

**INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

MID Lab Report

**Matric / ID No. :** C211237.

**ID No. (in words) :** C-Two-One-One-Two-Three-Seven.

**Name :** Anika Nawer.

**Semester :** 7th.

**Section :** 7BF.

**Course Code :** CSE-4742.

**Course Title :** Computer Graphics Lab.

**Course Teacher Name :** Tahsin Hossain.

***Submitted To:***

***Tahsin Hossain***

***Adjunct Lecturer***

***Department of Computer Science and Engineering International Islamic University Chittagong***

Topic Page No



1.A program to draw the following figure using built-in graphics function(5 different Shapes: Hexagon, House, House with a flag, Bi-cycle, Diamond Shape)…………………………………………………………………….04

2. Three different types of line algorithm: Basic line algorithm, DDA and Bresenham line algorithm………………………………………………...06

3. Two different types of circle algorithm: Bresenham circle, Midpoint Circle……………….……………...……………………………………...07

4.Translate a Line, Triangle and Rectangle about a point 5. Rotate a Line, Triangle and Rectangle about a point.........................................................10

5. Scale line, triangle and rectangle about a point………………………...12

6. Implement Cohen Southerland Line Clipping Algorithm. ……………..14

7. Implement Southerland Hodgeman Polygon Clipping Algorithm……..16

8. Write a program to draw a circle and when press ‘i’ on the keyboard the radius of the circle (increases by 15 pixel and when press ‘d’ decreases by

15 pixel) …………………………………………………………………...18

9. Write a program to draw a circle on the center of the screen. When press ‘l’ it moves on the left of the screen, when press ‘r’ moves on the right,when press ’u’ moves on the upward and when press ‘d’ moves on the downward…………………………………………………………………19

**Computer**

**graphics**

**LAB**

**REPORT**

1. A program to draw the following figure using built-in graphics function

(5 different Shapes: Hexagon, House, House with a flag, Bi-cycle, Diamond Shape)**:**

**Introduction:**

**Implementation:**

a.Hexagon:

#include <graphics.h>

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm, (char\*)"");

setcolor(RED);

line(200, 100, 300, 100);

line(300, 100, 350, 200);

line(350, 200, 300, 300);

line(300, 300, 200, 300);

line(200, 300, 150, 200);

line(150, 200, 200, 100);

setfillstyle(1,RED);

floodfill(250,150,RED);

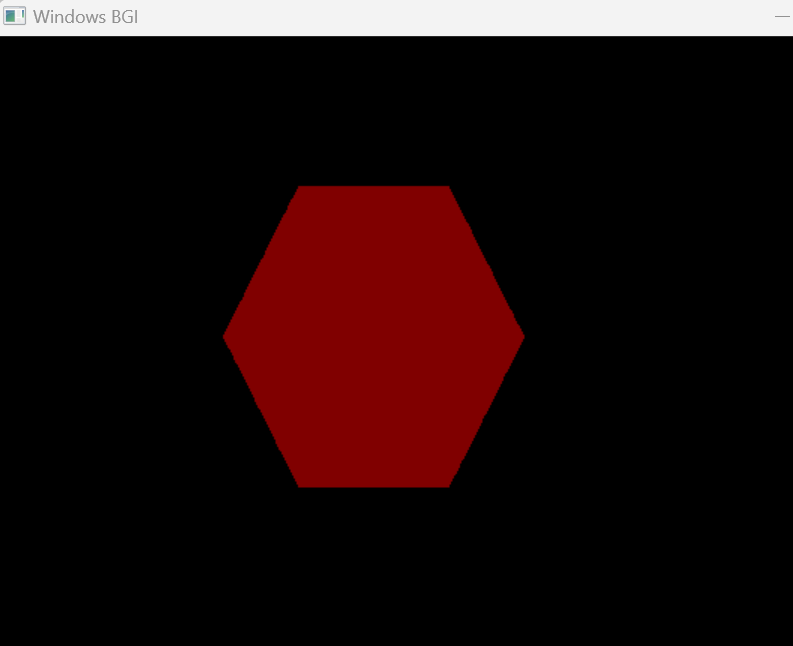
getchar();

closegraph();

return 0;

}

Output:



2.House:

#include <graphics.h>

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm, (char\*)"");

setbkcolor(WHITE);

cleardevice();

setcolor(BLUE);

rectangle(200, 50, 400, 100);

rectangle(150, 100, 450, 150);

rectangle(190,150,410,300);

rectangle(300, 300, 200, 300);

rectangle(275, 200, 325, 300);

rectangle(210, 180, 250, 220);

rectangle(360, 180, 400, 220);

//setfillstyle(1,GREEN);

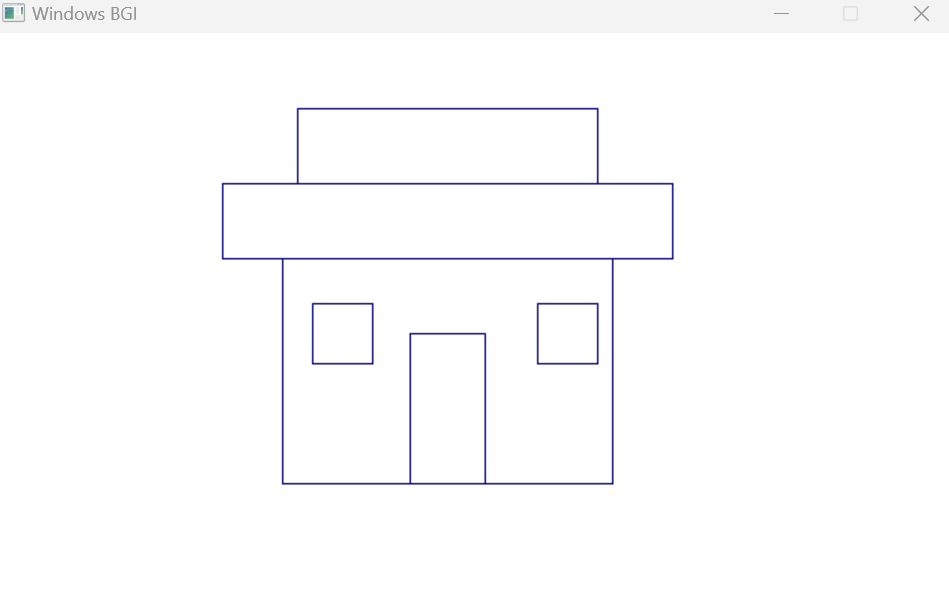
//floodfill(250,150,GREEN);

getchar();

closegraph();

return 0;

}



3.House with flag:

#include<graphics.h>

#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{

int gd,gm;

detectgraph( &gd,&gm);

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

setbkcolor(WHITE);

cleardevice();

setcolor(BLACK);

rectangle(300,50,330,70);

line(300,50,300,90);

line(300,90,240,160);

line(300,90,360,160);

line(240,160,360,160);

rectangle(140,160,460,190);

rectangle(70,190,530,450);

rectangle(70,190,165,450);

rectangle(70,190,165,290);

rectangle(165,190,191,450);

rectangle(191,190,286,450);

rectangle(191,190,286,290);

rectangle(286,190,314,450);

rectangle(314,190,409,450);

rectangle(314,190,409,290);

rectangle(409,190,435,450);

rectangle(435,190,530,450);

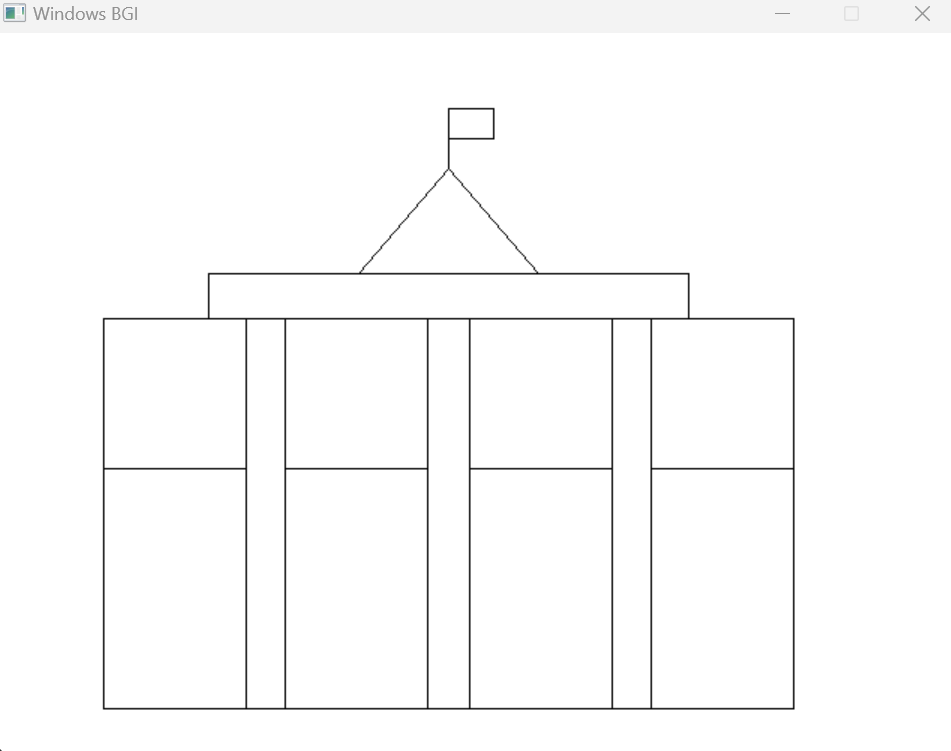
rectangle(435,190,530,290);

getchar();

return 0;

}

Output:



4.Cycle :

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

int main() {

int gd = DETECT, gm;

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

// Radius for circles

int radius\_left = 50;

int radius\_middle = 10;

int radius\_right = 50;

setlinestyle(0,0,7);

// Left circle

circle(200, 200, radius\_left);

// Middle circle

circle(300, 200, radius\_middle);

// Right circle

circle(400, 200, radius\_right);

// Line from left circle to middle circle

line(150 + radius\_left, 200, 300 - radius\_middle, 200);

line(150 + radius\_left, 200, 250, 120);

//line(250, 120, 300 , 200-radius\_middle);

line(235, 100, 300 , 200-radius\_middle);

line(300, 200-radius\_middle,350,120 );

line(338, 100,400,200 );

line(338, 100,360,90 );

line(220, 100, 250 , 100);

// Line from left circle to right circle

//line(150 + radius\_left, 200, 450 - radius\_right, 200);

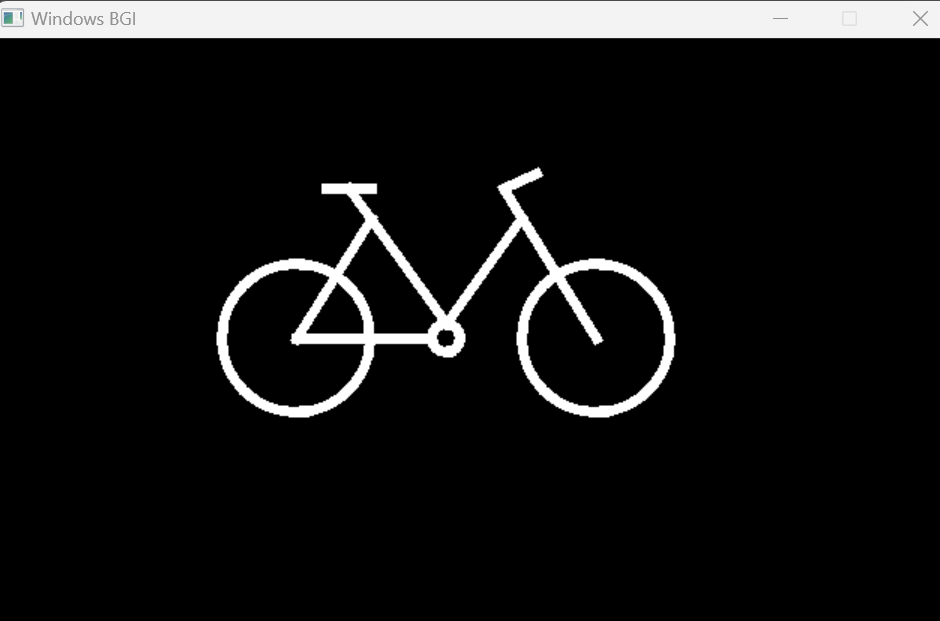
getchar();

closegraph();

return 0;

}

Output:



5.Dimond shape:

#include<graphics.h>

#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{

int gd,gm;

detectgraph( &gd,&gm);

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

setbkcolor(WHITE);

cleardevice();

setcolor(BLACK);

rectangle(100,50,400,450);

line(250,50,100,250);

line(250,50,400,250);

line(100,250,250,450);

line(250,450,400,250);

line(250,100,150,250);

line(250,100,350,250);

line(150,250,250,400);

line(250,400,350,250);

line(250,250,250,100);

line(250,250,400,250);

line(250,250,100,250);

line(250,250,250,400);

line(250,250,175,150);

line(250,250,300,175);

line(250,250,200,325);

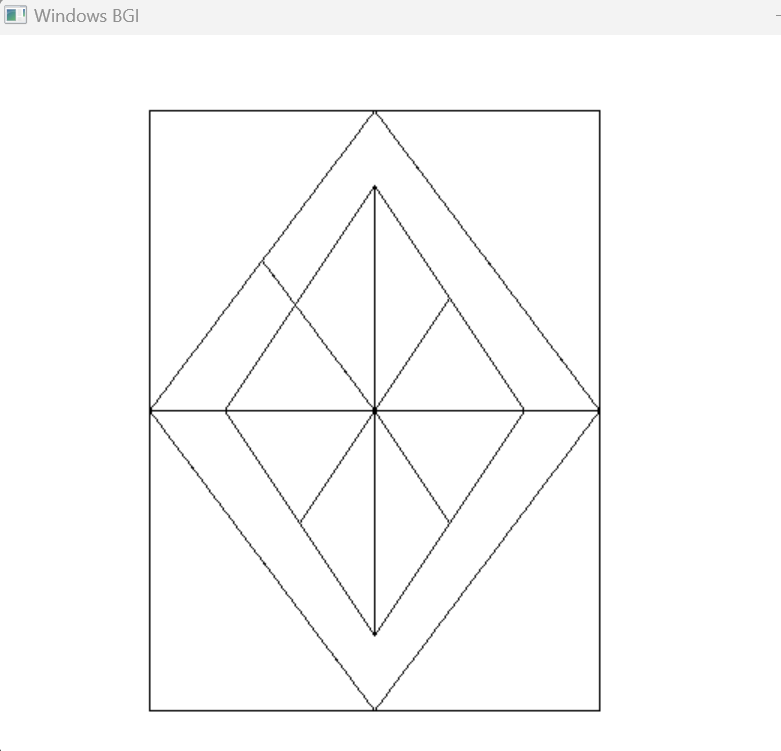
line(250,250,300,325);

getchar();

return 0;

}

Output:



**2.** Three different types of line algorithm: Basic line algorithm, DDA and Bresenham line algorithm**:**

**Introduction:**

a.direct line:

#include <graphics.h>

#include <iostream>

using namespace std;

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

{

float m = (float)(y2 - y1) / (x2 - x1);//slope

int dx = x2 - x1;

int dy = y2 - y1;

int x = x1,y = y1;

int gd = DETECT, gm;

initwindow(500, 500, "C211237");

putpixel(x, y, YELLOW);

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

{

x = x + 1;

y = y + m;

putpixel(x, y, YELLOW);

}

getch();

closegraph();

}

int main()

{

int x1, y1, x2, y2;

cout << "The coordinates of the First Endpoint (x1 y1): ";

cin >> x1 >> y1;

cout << "The coordinates of the Second Endpoint (x2 y2): ";

cin >> x2 >> y2;

drawLine(x1, y1, x2, y2);

return 0;

}

b.DDA:

#include<bits/stdc++.h>

#include<graphics.h>

#include<conio.h>

#include<stdio.h>

using namespace std;

int main()

{

int gd = DETECT,gm, i;

float x, y,dx,dy,steps;

int x0, x1, y0, y1;

initwindow(600, 600, "C211237");

setbkcolor(WHITE);

//x0 = 100 , y0 = 200, x1 = 500, y1 = 300;

cin>>x0>>y0>>x1>>y1;

dx = (float)(x1 - x0);

dy = (float)(y1 - y0);

if(dx>=dy)

{

steps = dx;

}

else

{

steps = dy;

}

dx = dx/steps;

dy = dy/steps;

x = x0;

y = y0;

i = 1;

while(i<= steps)

{

putpixel(x, y, RED);

x += dx;

y += dy;

i=i+1;

}

getchar();

closegraph();

}

c. Bresenham:

#include<bits/stdc++.h>

#include<graphics.h>

using namespace std;

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,WHITE);

p=p+2\*dy;

}

x=x+1;

}

}

int main()

{

int gdriver=DETECT, gmode, error, x0, y0, x1, y1;

initwindow(600, 600, "c211237");

cout<<"Enter co-ordinates of first point: ";

cin>>x0>>y0;

cout<<"Enter co-ordinates of second point: ";

cin>>x1>>y1;

drawline(x0, y0, x1, y1);

getchar();

return 0;

}

**3.** Two different types of circle algorithm:Bresenham circle, Midpoint Circle**:**

**Introduction:**

**Implementation:**

a.Bresenham circle :

#include <graphics.h>

#include <conio.h>

// Function to implement Bresenham's circle drawing algorithm

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

{

int x = 0, y = r;

int d = 3 - 2 \* r;

while (y >= x)

{

// For each point (x, y), plot the corresponding points in the other octants

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);

x++;

// Update the decision parameter based on the algorithm

if (d > 0)

{

y--;

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

}

else

{

d = d + 4 \* x + 6;

}

}

}

int main()

{

int gd = DETECT, gm;

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

int xc = 200, yc = 200, r = 100;

drawCircle(xc, yc, r);

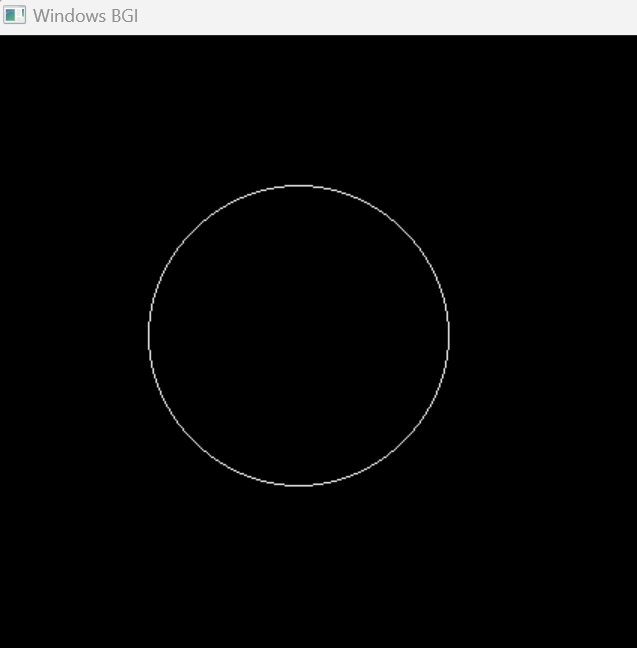
getchar();

closegraph();

return 0;

}

Output:



b. Midpoint Circle:

#include <graphics.h>

#include <conio.h>

// Function to plot the points in all eight octants

void plotPoints(int xc, int yc, int x, int 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);

}

// Function to implement Midpoint Circle Drawing Algorithm

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

{

int x = 0, y = r;

int d = 1 - r; // Decision parameter

plotPoints(xc, yc, x, y); // Initial point (0, r)

while (x <= y)

{

x++;

if (d < 0)

{

d = d + 2 \* x + 1;

}

else

{

y--;

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

}

plotPoints(xc, yc, x, y);

}

}

int main()

{

int gd = DETECT, gm;

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

int xc = 200, yc = 200, r = 100;

midpointCircleDrawing(xc, yc, r);

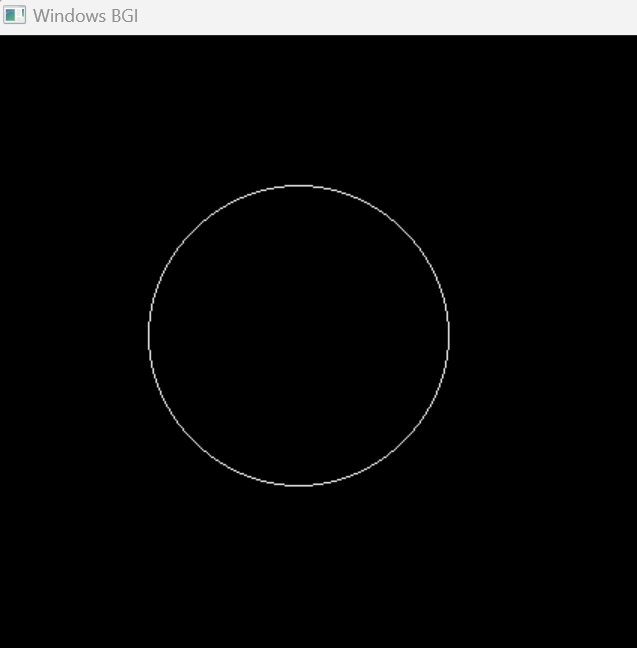
getchar();

closegraph();

return 0;

}

Output:



**4.** Translate a Line, Triangle and Rectangle about a point**:**

**Introduction:**

**Implementation:**

#include <graphics.h>

#include <conio.h>

// Function to translate a point

void translate(int &x, int &y, int tx, int ty)

{

x = x + tx;

y = y + ty;

}

int main()

{

int gd = DETECT, gm;

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

setbkcolor(WHITE);

cleardevice();

// Translation parameters

int tx = 50, ty = 50;

// Original and translated coordinates for line

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

line(x1, y1, x2, y2);

translate(x1, y1, tx, ty);

translate(x2, y2, tx, ty);

setcolor(RED);

line(x1, y1, x2, y2);

// Original and translated coordinates for triangle

int x3 = 150, y3 = 150, x4 = 250, y4 = 150, x5 = 200, y5 = 250;

line(x3, y3, x4, y4);

line(x4, y4, x5, y5);

line(x5, y5, x3, y3);

translate(x3, y3, tx, ty);

translate(x4, y4, tx, ty);

translate(x5, y5, tx, ty);

setcolor(GREEN);

line(x3, y3, x4, y4);

line(x4, y4, x5, y5);

line(x5, y5, x3, y3);

// Original and translated coordinates for rectangle

int x6 = 300, y6 = 150, x7 = 400, y7 = 150, x8 = 400, y8 = 250, x9 = 300, y9 = 250;

rectangle(x6, y6, x8, y8);

translate(x6, y6, tx, ty);

translate(x7, y7, tx, ty);

translate(x8, y8, tx, ty);

translate(x9, y9, tx, ty);

setcolor(BLUE);

rectangle(x6, y6, x8, y8);

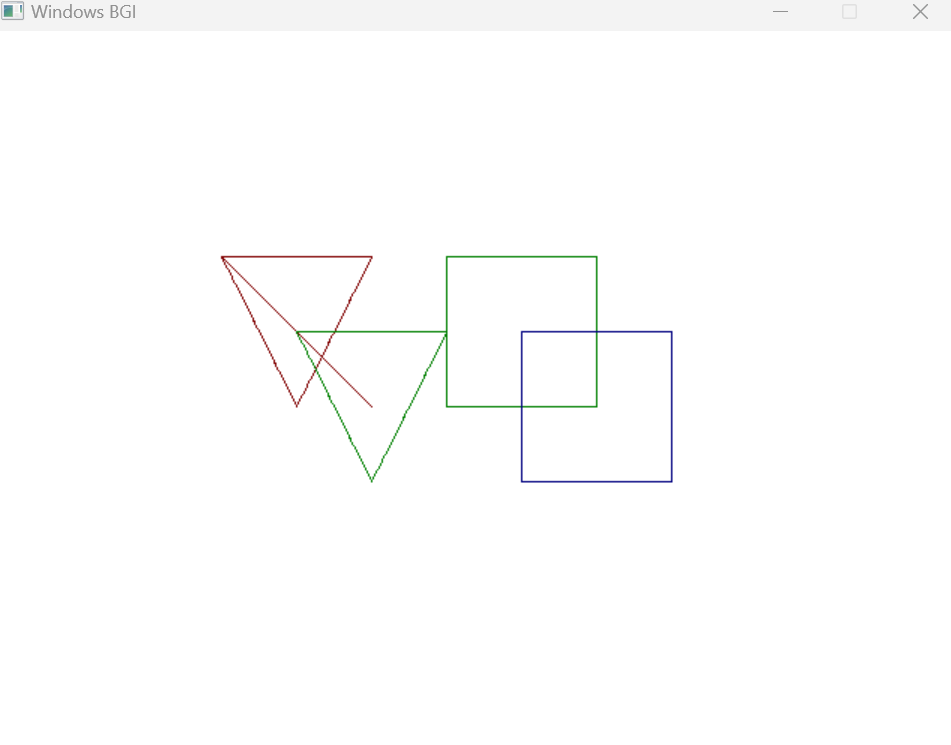
getchar();

closegraph();

return 0;

}

Output:



**5.** Rotate a Line, Triangle and Rectangle about a point:

**Introduction:**

**Implementation:**

#include <graphics.h>

#include <conio.h>

#include <math.h>

#define PI 3.14159265

// Function to rotate a point around a pivot

void rotate(int &x, int &y, int xr, int yr, float angle)

{

int tempX = x - xr;

int tempY = y - yr;

float rad = angle \* PI / 180.0;

x = xr + (tempX \* cos(rad) - tempY \* sin(rad));

y = yr + (tempX \* sin(rad) + tempY \* cos(rad));

}

int main()

{

int gd = DETECT, gm;

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

setbkcolor(WHITE);

cleardevice();

// Rotation parameters

int xr = 200, yr = 200; // Rotation center

float angle = 45.0; // Rotation angle in degrees

// Draw and rotate a line

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

line(x1, y1, x2, y2);

rotate(x1, y1, xr, yr, angle);

rotate(x2, y2, xr, yr, angle);

setcolor(RED);

line(x1, y1, x2, y2);

// Draw and rotate a triangle

int x3 = 150, y3 = 150, x4 = 250, y4 = 150, x5 = 200, y5 = 250;

line(x3, y3, x4, y4);

line(x4, y4, x5, y5);

line(x5, y5, x3, y3);

rotate(x3, y3, xr, yr, angle);

rotate(x4, y4, xr, yr, angle);

rotate(x5, y5, xr, yr, angle);

setcolor(GREEN);

line(x3, y3, x4, y4);

line(x4, y4, x5, y5);

line(x5, y5, x3, y3);

// Draw and rotate a rectangle

int x6 = 300, y6 = 150, x7 = 400, y7 = 150, x8 = 400, y8 = 250, x9 = 300, y9 = 250;

rectangle(x6, y6, x8, y8);

rotate(x6, y6, xr, yr, angle);

rotate(x7, y7, xr, yr, angle);

rotate(x8, y8, xr, yr, angle);

rotate(x9, y9, xr, yr, angle);

setcolor(BLUE);

line(x6, y6, x7, y7);

line(x7, y7, x8, y8);

line(x8, y8, x9, y9);

line(x9, y9, x6, y6);

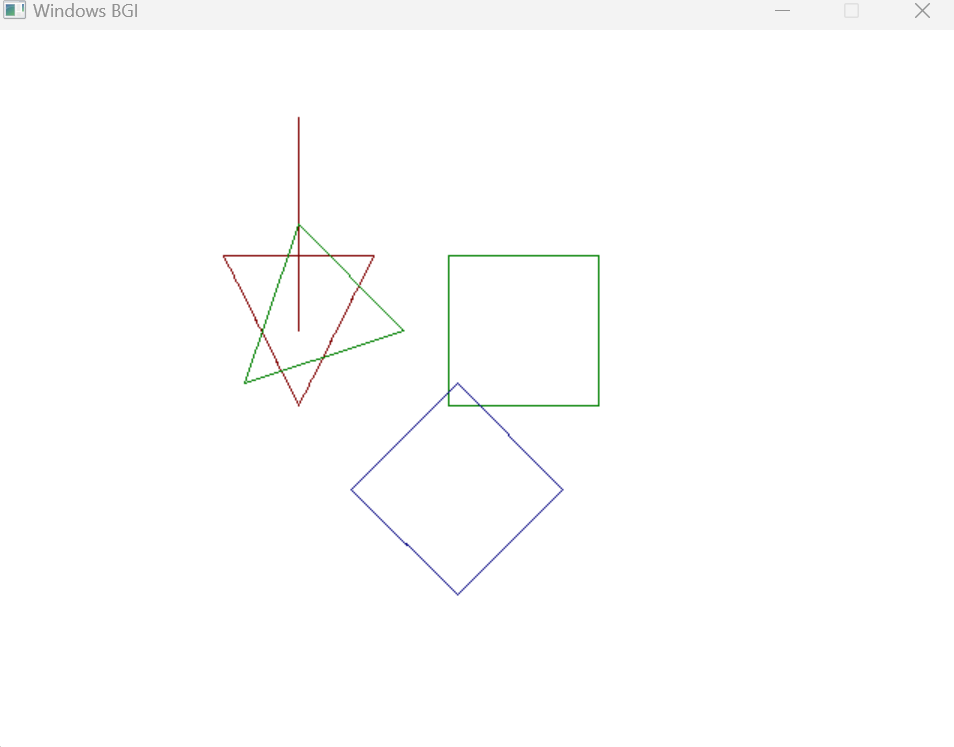
getchar();

closegraph();

return 0;

}

Output:



**6.** Scale line, triangle and rectangle about a point**:**

**Introduction:**

**Implementation:**

#include <graphics.h>

#include <conio.h>

// Function to scale a point around a pivot

void scale(int &x, int &y, int xs, int ys, float scaleFactor)

{

x = xs + (x - xs) \* scaleFactor;

y = ys + (y - ys) \* scaleFactor;

}

int main()

{

int gd = DETECT, gm;

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

setbkcolor(WHITE);

cleardevice();

// Scaling parameters

int xs = 200, ys = 200; // Scaling center

float scaleFactor = 1.5; // Scaling factor

// Draw and scale a line

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

line(x1, y1, x2, y2);

scale(x1, y1, xs, ys, scaleFactor);

scale(x2, y2, xs, ys, scaleFactor);

setcolor(RED);

line(x1, y1, x2, y2);

// Draw and scale a triangle

int x3 = 150, y3 = 150, x4 = 250, y4 = 150, x5 = 200, y5 = 250;

line(x3, y3, x4, y4);

line(x4, y4, x5, y5);

line(x5, y5, x3, y3);

scale(x3, y3, xs, ys, scaleFactor);

scale(x4, y4, xs, ys, scaleFactor);

scale(x5, y5, xs, ys, scaleFactor);

setcolor(GREEN);

line(x3, y3, x4, y4);

line(x4, y4, x5, y5);

line(x5, y5, x3, y3);

// Draw and scale a rectangle

int x6 = 300, y6 = 150, x7 = 400, y7 = 150, x8 = 400, y8 = 250, x9 = 300, y9 = 250;

rectangle(x6, y6, x8, y8);

scale(x6, y6, xs, ys, scaleFactor);

scale(x7, y7, xs, ys, scaleFactor);

scale(x8, y8, xs, ys, scaleFactor);

scale(x9, y9, xs, ys, scaleFactor);

setcolor(BLUE);

line(x6, y6, x7, y7);

line(x7, y7, x8, y8);

line(x8, y8, x9, y9);

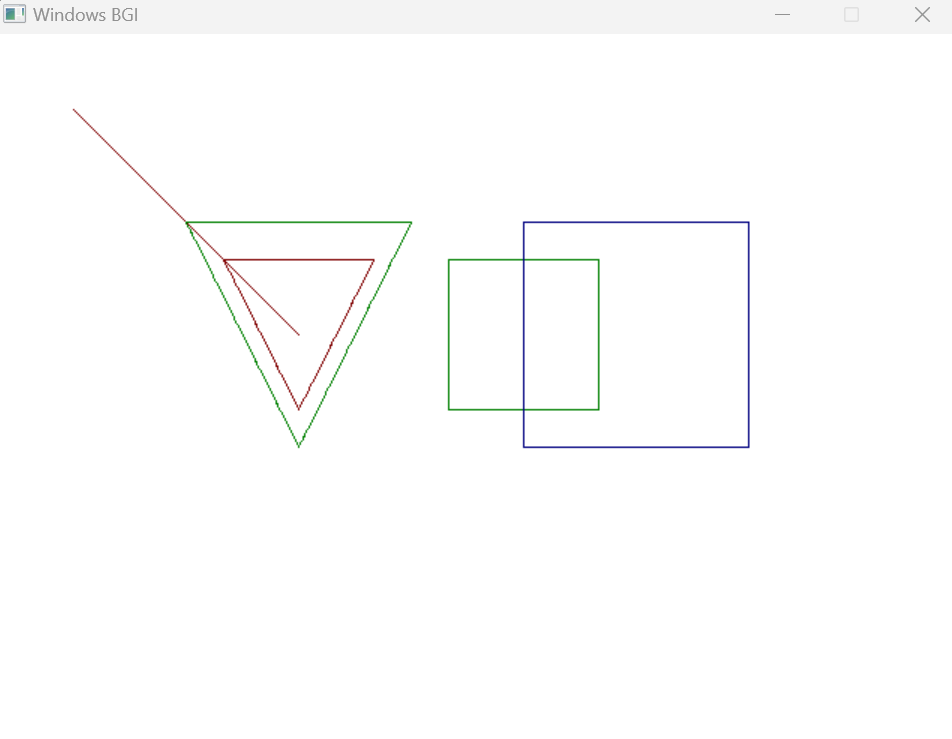
line(x9, y9, x6, y6);

getchar();

closegraph();

return 0;

}



**7.** Implement Cohen Southerland Line Clipping Algorithm**:**

**Introduction:**

**Implementation:**

#include <graphics.h>

#include <conio.h>

const int INSIDE = 0; // 0000

const int LEFT = 1; // 0001

const int RIGHT = 2; // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8; // 1000

// Window boundaries

const int x\_max = 300;

const int y\_max = 300;

const int x\_min = 100;

const int y\_min = 100;

// Function to compute the region code for a point (x, y)

int computeCode(int x, int y)

{

int code = INSIDE;

if (x < x\_min)

code |= LEFT;

else if (x > x\_max)

code |= RIGHT;

if (y < y\_min)

code |= BOTTOM;

else if (y > y\_max)

code |= TOP;

return code;

}

// Cohen-Sutherland Line Clipping Algorithm

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

{

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

bool accept = false;

while (true)

{

if ((code1 == 0) && (code2 == 0))

{

// If both endpoints lie within the rectangle

accept = true;

break;

}

else if (code1 & code2)

{

// If both endpoints share an outside zone (both are outside the rectangle)

break;

}

else

{

// Some segment of the line lies within the rectangle

int code\_out;

int x, y;

// At least one endpoint is outside the rectangle, pick it

if (code1 != 0)

code\_out = code1;

else

code\_out = code2;

// Find the intersection point using formulas

if (code\_out & TOP)

{

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

y = y\_max;

}

else if (code\_out & BOTTOM)

{

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

y = y\_min;

}

else if (code\_out & RIGHT)

{

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

x = x\_max;

}

else if (code\_out & LEFT)

{

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

x = x\_min;

}

// Replace the outside point with the intersection point and update the code

if (code\_out == code1)

{

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

}

else

{

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

if (accept)

{

setcolor(GREEN);

rectangle(x\_min, y\_min, x\_max, y\_max);

setcolor(RED);

line(x1, y1, x2, y2);

}

}

int main()

{

int gd = DETECT, gm;

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

// Example lines

int x1 = 50, y1 = 50, x2 = 400, y2 = 400;

setcolor(WHITE);

line(x1, y1, x2, y2);

// Perform Cohen-Sutherland Line Clipping

cohenSutherlandClip(x1, y1, x2, y2);

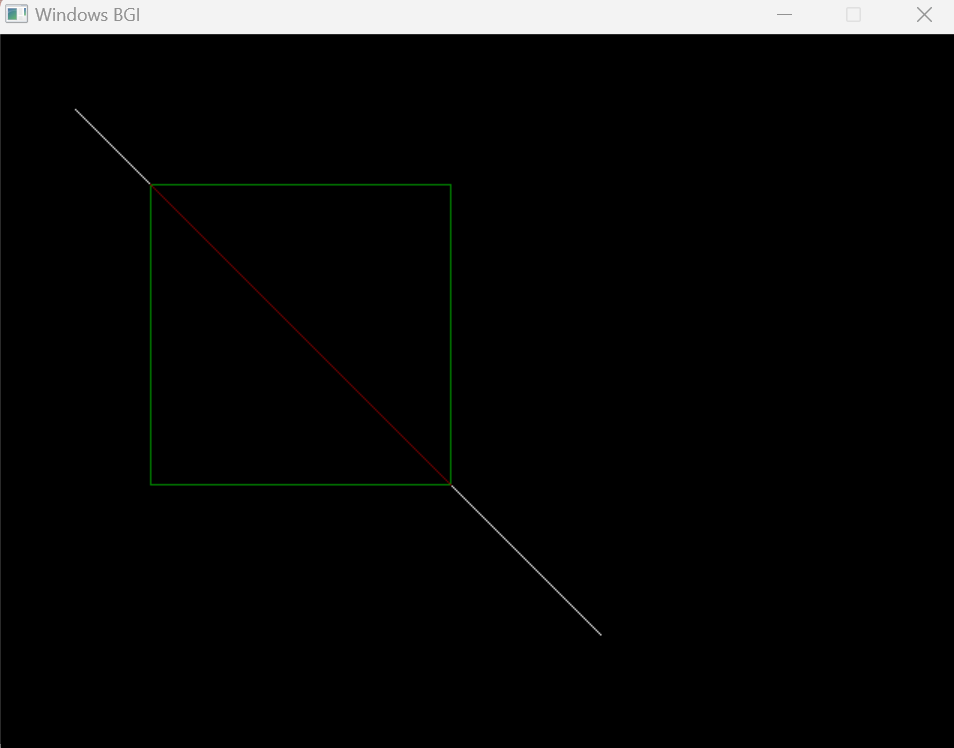
getchar();

closegraph();

return 0;

}

Output:



**7.** Implement Southerland Hodgeman Polygon Clipping Algorithm**:**

**Introduction**

**Implementation:**

#include<bits/stdc++.h>

#include <graphics.h>

#include <conio.h>

#include <math.h>

#include <process.h>

#define TRUE 1

#define FALSE 0

typedef unsigned int outcode; // alias outcode create kore for the data type unsigned int

//outcode

int CompOutCode(float x,float y);

enum { TOP = 0x1, //outcodes er jnno enumeration define korte hbe.

BOTTOM = 0x2, //These values represent whether a point is above, below, to the right, or to the left of a clipping window edge.

RIGHT = 0x4,

LEFT = 0x8

};

float xmin,xmax,ymin,ymax;

int clip(float x0,float y0,float x1,float y1)

{

outcode outcode0,outcode1,outcodeOut;//These variables will store the outcodes for the two

//endpoints and the result outcode during clipping.

int accept = FALSE,done = FALSE; //These variables are used to determine whether the line segment

//is accepted after clipping and whether the clipping process is done.

outcode0 = CompOutCode(x0,y0); //two endpoints of the line segment using the CompOutCode function.

outcode1 = CompOutCode(x1,y1);

do

{

if(!(outcode0|outcode1)) //If both outcodes are zero (meaning both endpoints are inside

{

//the clipping window), set accept to TRUE and done to TRUE to accept the line segment.

accept = TRUE;

done = TRUE;

}

else if(outcode0 & outcode1) //If both outcodes have a common set bit, set done to TRUE, indicating that

done = TRUE; //the line is entirely outside the window.

else // If neither of the above conditions is met, it means the line needs clipping.

{

float x,y;

outcodeOut = outcode0?outcode0:outcode1; //Inside this block, the code calculates the intersection point

//of the line with the clipping window and updates the appropriate

//endpoint (x0, y0, or x1, y1) and their corresponding outcodes (outcode0 or outcode1).

if(outcodeOut & TOP)

{

x = x0+(x1-x0)\*(ymax-y0)/(y1-y0);

y = ymax;

}

else if(outcodeOut & BOTTOM)

{

x = x0+(x1-x0)\*(ymin-y0)/(y1-y0);

y = ymin;

}

else if(outcodeOut & RIGHT)

{

y = y0+(y1-y0)\*(xmax-x0)/(x1-x0);

x = xmax;

}

else

{

y = y0+(y1-y0)\*(xmin-x0)/(x1-x0);

x = xmin;

}

if(outcodeOut==outcode0)

{

x0 = x;

y0 = y;

outcode0 = CompOutCode(x0,y0);

}

else

{

x1 = x;

y1 = y;

outcode1 = CompOutCode(x1,y1);

}

}

}

while(done==FALSE);// TRUE na howa prjnto loop lgbe, indicating the

//line is either accepted or entirely outside the window

if(accept) //If the line is accepted after clipping, draw it using the line function.

line(x0,y0,x1,y1);

outtextxy(150,20,"POLYGON AFTER CLIPPING");//Display a text message on the graphics window.

rectangle(xmin,ymin,xmax,ymax);

}

//outcode

int CompOutCode(float x,float y)

{

outcode code = 0;

if(y>ymax)

code|=TOP;

else if(y<ymin)

code|=BOTTOM;

if(x>xmax)

code|=RIGHT;

else if(x<xmin)

code|=LEFT;

return code;

}

int main( )

{

float x1,y1,x2,y2;

/\* request auto detection \*/

int n,poly[14],i;

printf("Enter the no of sides of polygon:");

scanf("%d",&n);

printf("\nEnter the coordinates of polygon\n");

for( i=0; i<2\*n; i++) //x,y er value er jnno 2n prjnto

{

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

}

poly[2\*n]=poly[0];

poly[2\*n+1]=poly[1];

printf("Enter the rectangular coordinates of clipping window\n");

scanf("%f%f%f%f",&xmin,&ymin,&xmax,&ymax);

/\* initialize graphics and local variables \*/

initwindow(900, 900, "c211237");

outtextxy(150,20,"POLYGON BEFORE CLIPPING");

drawpoly(n+1,poly);

rectangle(xmin,ymin,xmax,ymax);

getch( );

cleardevice( );

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

clip(poly[2\*i],poly[(2\*i)+1],poly[(2\*i)+2],poly[(2\*i)+3]);

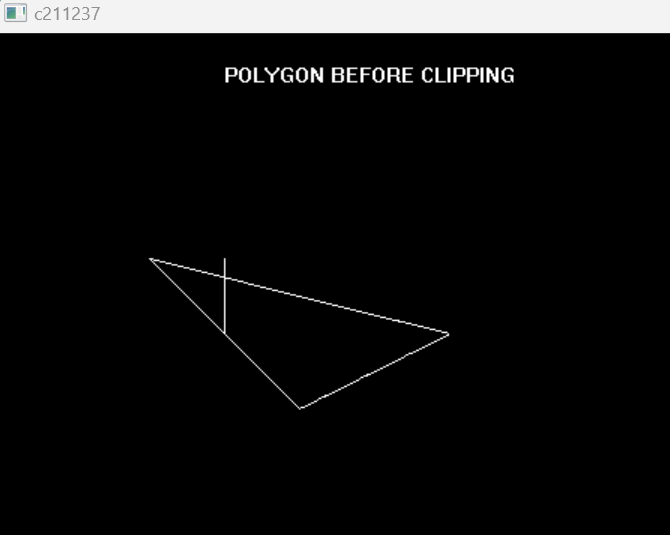
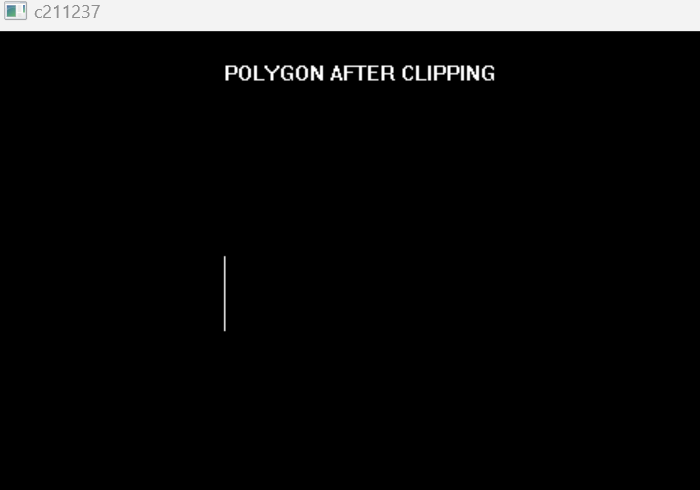
getch( );

restorecrtmode( );

return 0;

}

Output:

****

**9.** Write a program to draw a circle and when press ‘i’ on the keyboard the radius of the circle increases by 15 pixel and when press ‘d’ decreases by

15 pixel**:**

**Introduction:**

**Implementation:**

#include <graphics.h>

#include <conio.h>

int main()

{

int gd = DETECT, gm;

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

int x = getmaxx() / 2;

int y = getmaxy() / 2;

int radius = 50; // Initial radius

char ch;

do

{

cleardevice(); // Clear the screen

circle(x, y, radius); // Draw the circle

ch = getchar(); // Get a key press

if (ch == 'i')

{

radius += 15; // Increase radius

}

else if (ch == 'd')

{

radius -= 15; // Decrease radius

}

// Ensure radius does not become negative

if (radius < 0)

{

radius = 0;

}

}

while (ch != 'q'); // Press 'q' to quit

closegraph();

return 0;

}

**10.** Write a program to draw a circle on the center of the screen. When press ‘l’ it moves on the left of the screen, when press ‘r’ moves on the right,when press ’u’ moves on the upward and when press ‘d’ moves on the downward**:**

**Introduction:**

**Implementation:**

#include <graphics.h>

#include <conio.h>

int main()

{

int gd = DETECT, gm;

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

int x = getmaxx() / 2;

int y = getmaxy() / 2;

int radius = 50; // Radius of the circle

char ch;

do

{

cleardevice(); // Clear the screen

circle(x, y, radius); // Draw the circle

ch = getchar(); // Get a key press

if (ch == 'l')

{

x -= 10; // Move left

}

else if (ch == 'r')

{

x += 10; // Move right

}

else if (ch == 'u')

{

y -= 10; // Move up

}

else if (ch == 'd')

{

y += 10; // Move down

}

}

while (ch != 'q'); // Press 'q' to quit

closegraph();

return 0;

}

End