**EXPERIMENT – 1**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include <iostream.h>

#include <graphics.h>

#include <math.h>

#include <conio.h> // Include for getch()

class pixel {

public:

float x, y, length, dx, dy;

int p;

};

class pixel1 : public pixel {

public:

void DDA(float, float, float, float);

void bresen(float, float, float, float);

int sign(float);

};

int pixel1::sign(float x) {

if (x < 0)

return -1;

else if (x == 0)

return 0;

else

return 1;

}

void pixel1::DDA(float x1, float y1, float x2, float y2) {

dx = fabs(x2 - x1);

dy = fabs(y2 - y1);

if (dx > dy)

length = dx;

else

length = dy;

dx = (x2 - x1) / length;

dy = (y2 - y1) / length;

x = x1 + 0.5 \* sign(dx);

y = y1 + 0.5 \* sign(dy);

// Draw axes

line(0, 240, 640, 240);

line(320, 0, 320, 480);

for (int i = 0; i < length; i++) {

x = x + dx;

y = y + dy;

putpixel((int)x, (int)y, WHITE); // Casting to int for putpixel

}

}

void pixel1::bresen(float x1, float y1, float x2, float y2) {

int temp, exchange\_flag = 0;

dx = fabs(x2 - x1);

dy = fabs(y2 - y1);

int s1 = sign(x2 - x1);

int s2 = sign(y2 - y1);

x = x1;

y = y1;

if (dy > dx) {

temp = dx;

dx = dy;

dy = temp;

exchange\_flag = 1;

} else {

exchange\_flag = 0;

}

p = 2 \* dy - dx;

// Draw axes

line(0, 240, 640, 240);

line(320, 0, 320, 480);

int i = 0;

while (i <= dx) {

putpixel((int)fabs(x), (int)fabs(y), WHITE); // Casting to int for putpixel

if (p >= 0) {

if (exchange\_flag == 1)

x = x + s1;

else

y = y + s2;

p = p - 2 \* dx;

}

if (exchange\_flag == 1)

y = y + s2;

else

x = x + s1;

p = p + 2 \* dy;

i++;

} // while end

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI"); // Specify the path to BGI folder

pixel1 s;

float x1, y1, x2, y2;

char ans;

int ch;

do {

cout << "\n\*\*\*\*MENU\*\*\*\*";

cout << "\n1.DDA ";

cout << "\n2.Bresenham ";

cout << "\n3.Exit ";

cin >> ch;

switch (ch) {

case 1:

cout << "\nEnter co-ordinates of line (x1, y1, x2, y2): ";

cin >> x1 >> y1 >> x2 >> y2;

x1 = x1 + 320;

y1 = 240 - y1;

x2 = x2 + 320;

y2 = 240 - y2;

s.DDA(x1, y1, x2, y2);

break;

case 2:

cout << "\nEnter co-ordinates of line (x1, y1, x2, y2): ";

cin >> x1 >> y1 >> x2 >> y2;

x1 = x1 + 320;

y1 = 240 - y1;

x2 = x2 + 320;

y2 = 240 - y2;

s.bresen(x1, y1, x2, y2);

break;

case 3:

break;

}

cout << "\nDo you want to continue...(y/n): ";

cin >> ans;

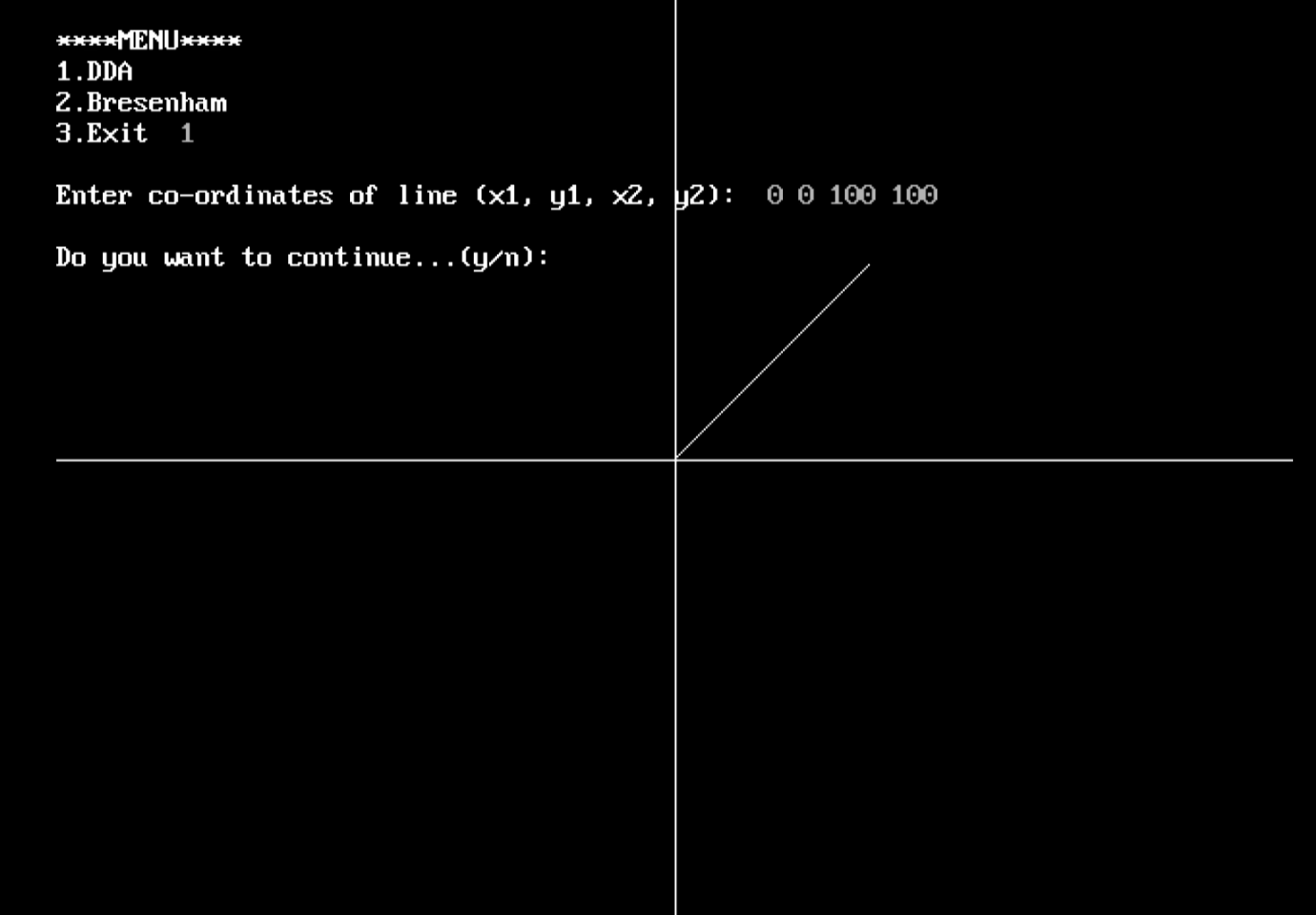
} while (ans == 'y' || ans == 'Y');

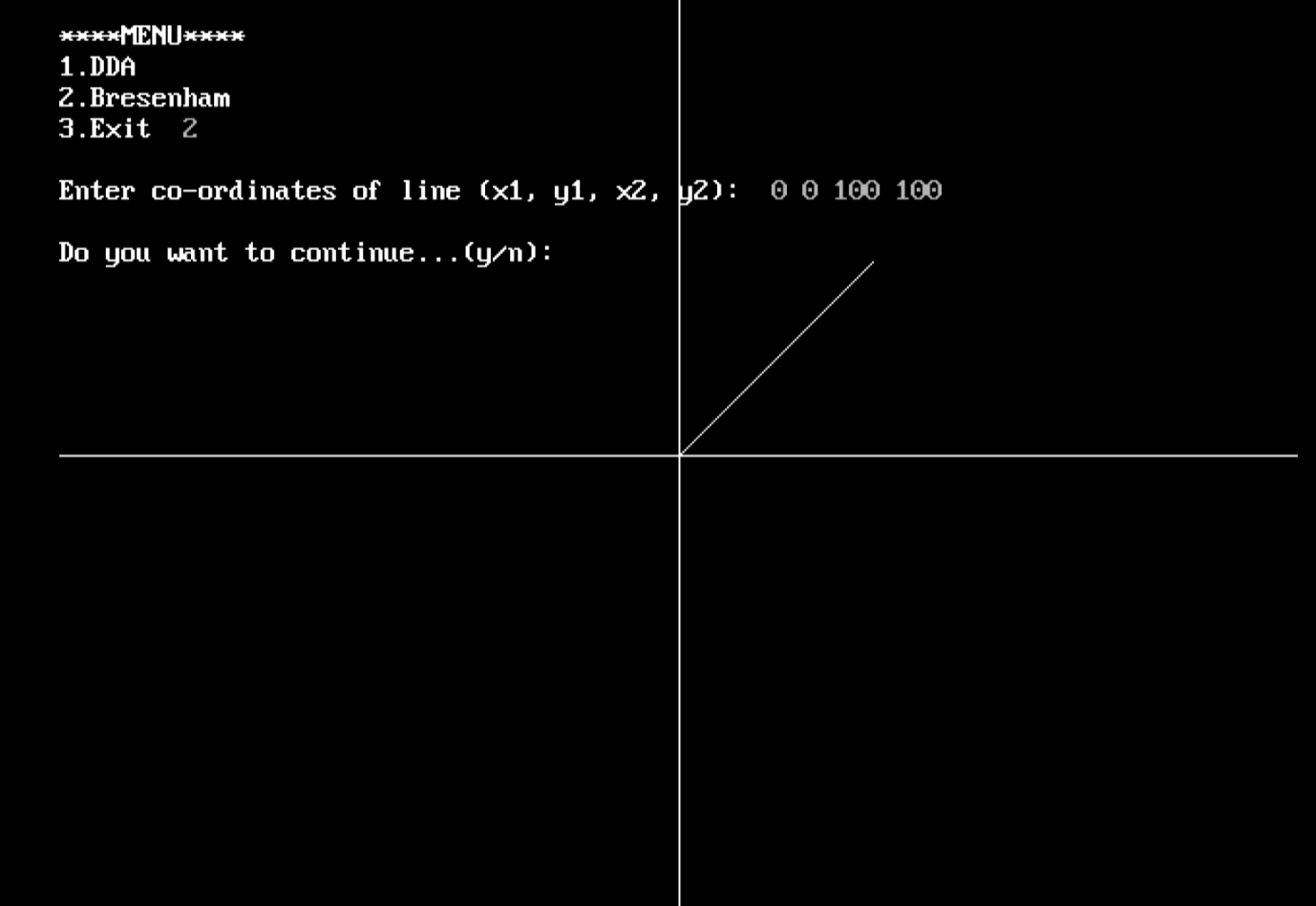
getch(); // Fix: Include conio.h for this

closegraph();

return 0;

}





**EXPERIMENT – 2**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include<iostream.h>

#include<graphics.h>

#include<conio.h>

// Class to hold coordinates and parameters

class CircleParams {

public:

int x, y, x1, y1, r, d;

};

// Class to implement Bresenham's Circle Drawing algorithm

class BresenhamCircle : public CircleParams {

public:

void getData(); // Function to get the input data

void draw(); // Function to draw the circle

};

// Function to get center coordinates and radius of the circle

void BresenhamCircle::getData() {

cout << "\nEnter center coordinates (x, y): ";

cin >> x1 >> y1;

x1 = x1 + 320; // Adjust for screen center

y1 = 240 - y1;

cout << "\nEnter radius of the circle: ";

cin >> r;

}

// Function to draw the circle using Bresenham's algorithm

void BresenhamCircle::draw() {

d = 3 - 2 \* r; // Initial decision parameter

x = 0;

y = r;

// Draw axes

line(320, 0, 320, 480);

line(0, 240, 640, 240);

// Start drawing the circle

do {

putpixel(x1 + x, y1 + y, 15);

putpixel(x1 + x, y1 - y, 15);

putpixel(x1 + y, y1 + x, 15);

putpixel(x1 + y, y1 - x, 15);

putpixel(x1 - x, y1 + y, 15);

putpixel(x1 - x, y1 - y, 15);

putpixel(x1 - y, y1 + x, 15);

putpixel(x1 - y, y1 - x, 15);

if (d < 0) {

d = d + 4 \* x + 6;

} else {

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

y--;

}

x++;

} while (x < y);

}

int main() {

int gd = DETECT, gm;

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

BresenhamCircle circle;

circle.getData();

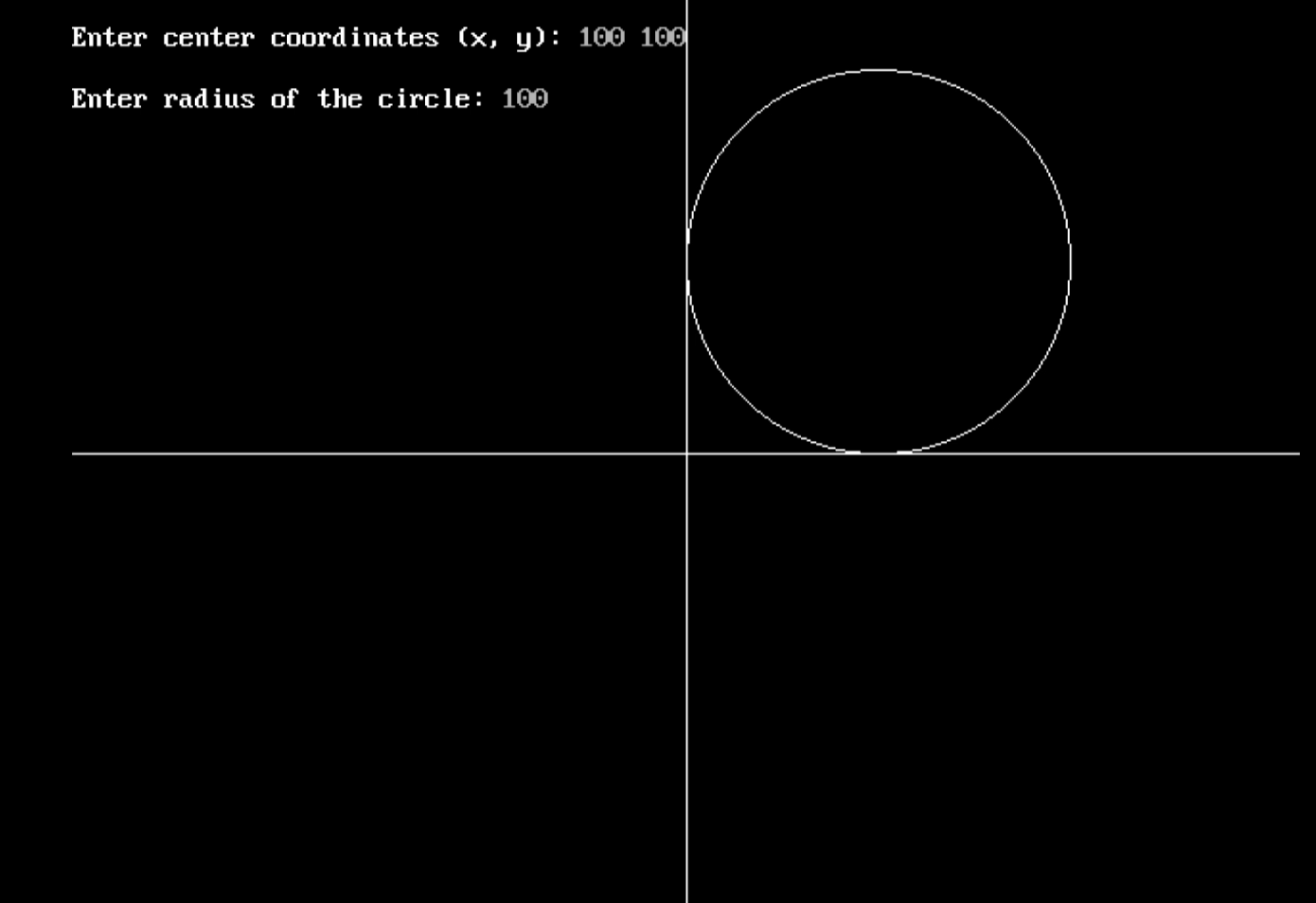
circle.draw();

getch(); // Wait for a key press

closegraph(); // Close the graphics mode

return 0;

}



**EXPERIMENT – 3**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include<iostream.h>

#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<dos.h>

class pixel1

{

public:

float x,y,length,dx,dy,height,width,X,Y;

void DDA(float,float,float,float);

void pattern();

int sign(float);

};

int pixel1::sign(float x)

{

if(x<0)

return -1;

else if(x==0)

return 0;

else

return 1;

}

void pixel1::DDA(float x1,float y1,float x2,float y2)

{

dx=abs(x2-x1);

dy=abs(y2-y1);

if(dx>dy)

length=dx;

else

length=dy;

dx=(x2-x1)/length;

dy=(y2-y1)/length;

x=x1+0.5\*sign(dx);

y=y1+0.5\*sign(dy);

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

{

delay(1);

x=x+dx;

y=y+dy;

putpixel(x,y,WHITE);

}

}

void pixel1:: pattern()

{

cout<<"\nEnter co-ordinates (X,Y)";

cin>>X>>Y;

cout<<"\nEnter width and Height";

cin>>width>>height;

DDA(X,Y,X+width,Y);

DDA(X,Y,X,Y+height);

DDA(X,Y+height,X+width,Y+height);

DDA(X+width,Y,X+width,Y+height);

DDA(X,Y+height/2,X+width/2,Y);

DDA(X,Y+height/2,X+width/2,Y+height);

DDA(X+width,Y+height/2,X+width/2,Y);

DDA(X+width,Y+height/2,X+width/2,Y+height);

DDA(X+width/4,Y+height/4,X+3\*width/4,Y+height/4);

DDA(X+width/4,Y+3\*height/4,X+3\*width/4,Y+3\*height/4);

DDA(X+width/4,Y+height/4,X+width/4,Y+3\*height/4);

DDA(X+3\*width/4,Y+height/4,X+3\*width/4,Y+3\*height/4);

}

int main()

{

int gd = DETECT, gm;

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

pixel1 s;

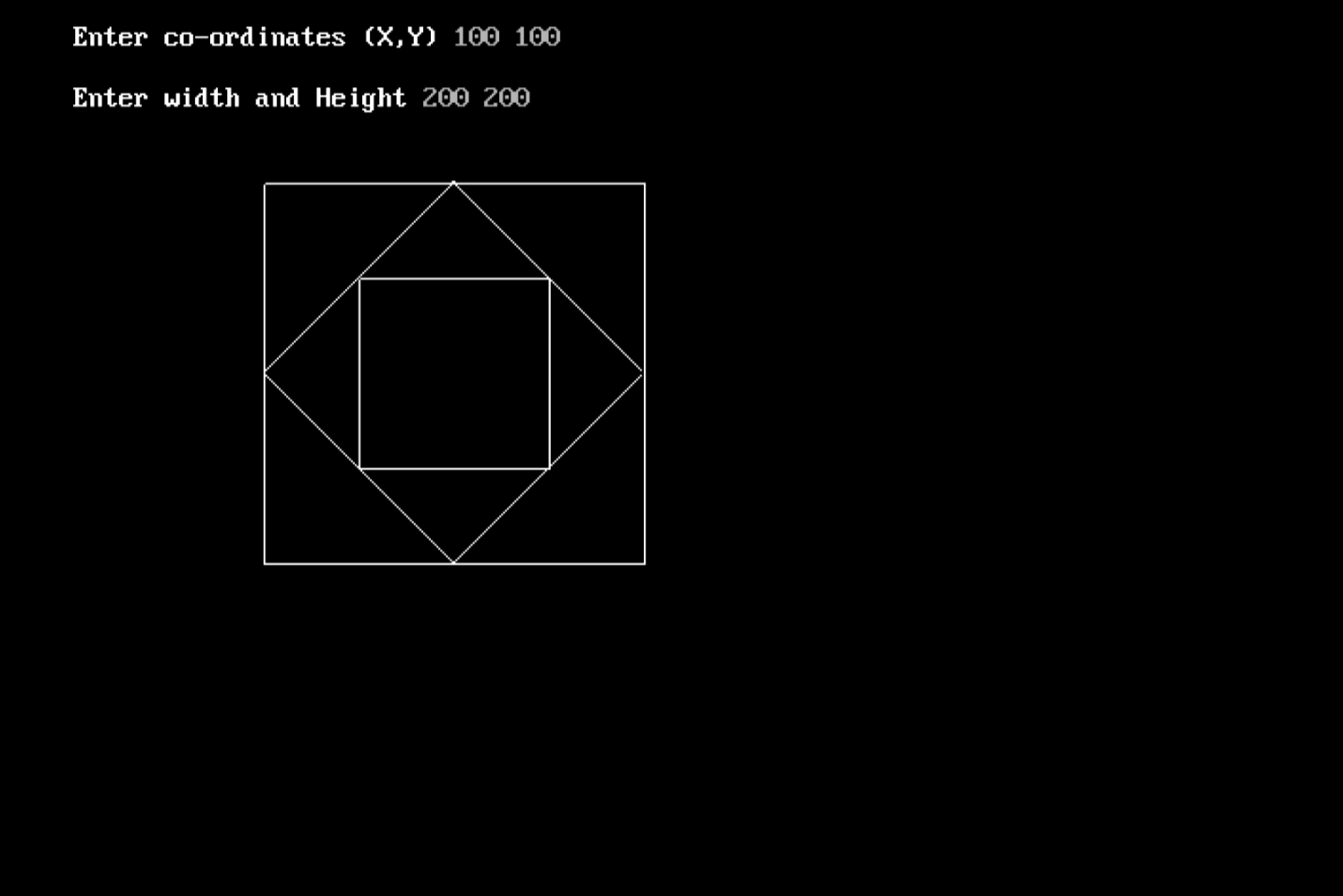
s.pattern();

delay(10000);

getch();

closegraph();

}



**EXPERIMENT – 4**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include<conio.h>

#include<graphics.h>

#include<math.h>

void ddaline(int x1, int y1, int x2, int y2) {

int s, m, dx, dy;

float xi, yi, x, y;

dx = x2 - x1;

dy = y2 - y1;

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

s = abs(dx); else

s = abs(dy);

xi = dx / (float) s;

yi = dy / (float) s;

x = x1;

y = y1;

putpixel(x1 + 0.5, y1 + 0.5, 15);

for (m = 0; m < s; m++) {

x += xi;

y += yi;

putpixel(x + 0.5, y + 0.5, 15);

}

}

void fill(int x, int y) {

int i, j;

for (i = x; i < (x + 50); i++)

ddaline(i, y, i, y + 50);

}

int main()

{

int i, j, c = 0;

int gd = DETECT, gm = DETECT;

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

cleardevice();

ddaline(100, 50, 100, 450);

ddaline(100, 50, 500, 50);

ddaline(500, 50, 500, 450);

ddaline(100, 450, 500, 450);

for (i = 100; i < 500; i += 50) {

for (j = 50; j < 450; j += 50) {

if (c % 2 == 0)

fill(i, j);

c++;

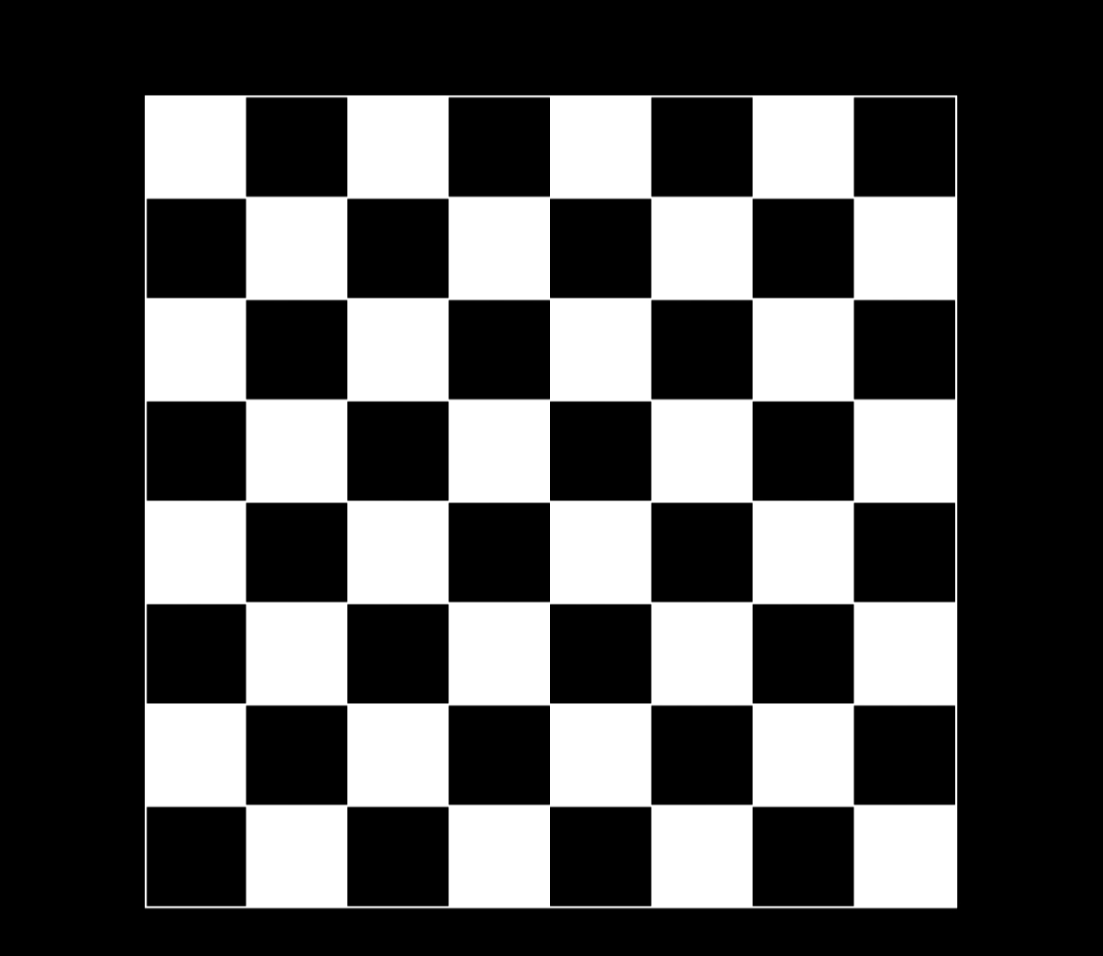
}

c++;

}

getch();

}



**EXPERIMENT – 5**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include <iostream.h>

#include <graphics.h>

#include <conio.h>

#include <dos.h> // Include dos.h for delay function

class sfill {

int x, y;

public:

void seedfill(int, int, int, int);

void draw();

};

void sfill::draw() {

int p[20][2], i, n;

line(320, 0, 320, 480);

line(0, 240, 640, 240);

cout << "\nEnter the number of vertices of the polygon: ";

cin >> n;

cout << "\nEnter all coordinates by <space>: ";

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

cin >> p[i][0] >> p[i][1];

p[i][0] = 320 + p[i][0];

p[i][1] = 240 - p[i][1];

}

p[n][0] = p[0][0];

p[n][1] = p[0][1];

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

line(p[i][0], p[i][1], p[i + 1][0], p[i + 1][1]);

}

line(p[i][0], p[i][1], p[0][0], p[0][1]);

}

void sfill::seedfill(int x, int y, int oldcolor, int newcolor) {

int color;

color = getpixel(x, y);

if (color == oldcolor && color != newcolor) {

putpixel(x, y, newcolor);

delay(2); // Delay for visual effect

seedfill(x + 1, y, oldcolor, newcolor);

seedfill(x - 1, y, oldcolor, newcolor);

seedfill(x, y + 1, oldcolor, newcolor);

seedfill(x, y - 1, oldcolor, newcolor);

}

}

int main() {

int gd = DETECT, gm;

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

sfill s;

s.draw();

int x, y;

cout << "\nEnter an inside point of the polygon: ";

cin >> x >> y;

x = x + 320;

y = 240 - y;

int oldcolor = getpixel(x, y);

int newcolor = 2; // Color to fill

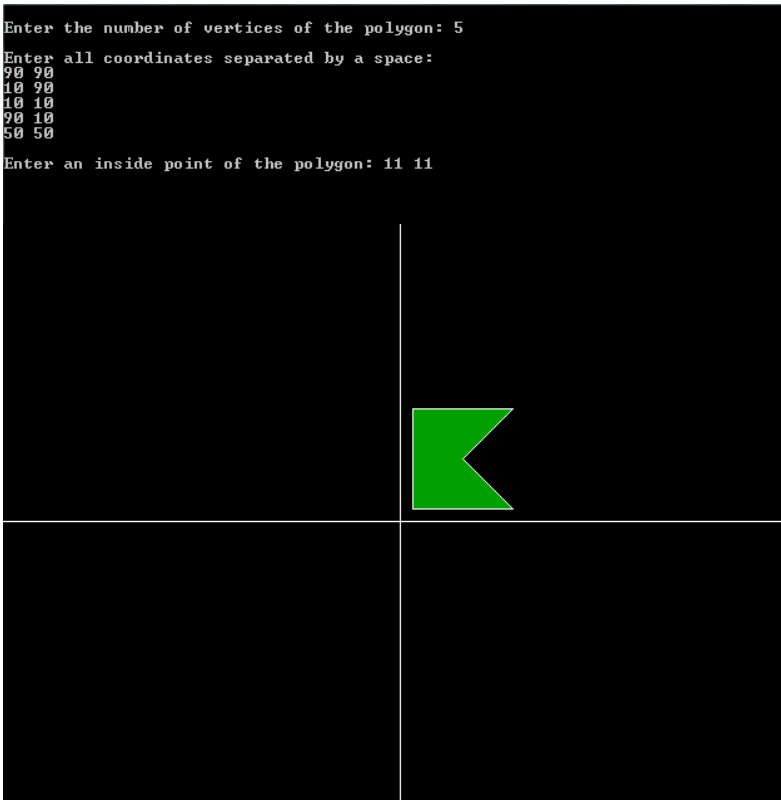
s.seedfill(x, y, oldcolor, newcolor);

delay(10000); // Delay to keep the window open

closegraph();

return 0;

}



**EXPERIMENT – 6**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include <conio.h> // For getch()

#include <graphics.h> // For graphics functions

#include <stdio.h> // For printf and scanf

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

int xmin, ymin, xmax, ymax;

// Function to compute outcode for a point (x, y)

int computeOutcode(int x, int y) {

int code = INSIDE;

if (x < xmin)

code |= LEFT;

else if (x > xmax)

code |= RIGHT;

if (y < ymin)

code |= BOTTOM;

else if (y > ymax)

code |= TOP;

return code;

}

// Cohen-Sutherland Line Clipping algorithm

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

int outcode1 = computeOutcode(x1, y1);

int outcode2 = computeOutcode(x2, y2);

int accept = 0; // Use int instead of bool

while (1) {

if (!(outcode1 | outcode2)) {

// Both endpoints are inside the window

accept = 1;

break;

} else if (outcode1 & outcode2) {

// Both endpoints are outside the window

break;

} else {

// Clipping is needed

int outcodeOut;

int x, y;

// Choose the point outside the window

if (outcode1 != 0)

outcodeOut = outcode1;

else

outcodeOut = outcode2;

// Find intersection point

if (outcodeOut & TOP) {

// Point is above the clip window

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

y = ymax;

} else if (outcodeOut & BOTTOM) {

// Point is below the clip window

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

y = ymin;

} else if (outcodeOut & RIGHT) {

// Point is to the right of clip window

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

x = xmax;

} else if (outcodeOut & LEFT) {

// Point is to the left of clip window

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

x = xmin;

}

// Replace the outside point with the intersection point

if (outcodeOut == outcode1) {

x1 = x;

y1 = y;

outcode1 = computeOutcode(x1, y1);

} else {

x2 = x;

y2 = y;

outcode2 = computeOutcode(x2, y2);

}

}

}

if (accept) {

// Draw the clipped line

setcolor(GREEN);

line(x1, y1, x2, y2);

}

}

// Main function

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI"); // Make sure BGI path is correct

// Input the window boundaries

printf("Enter xmin, ymin, xmax, ymax for clipping window: ");

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

// Draw the clipping window

rectangle(xmin, ymin, xmax, ymax);

// Input the line endpoints

int x1, y1, x2, y2;

printf("Enter the coordinates of the line (x1, y1, x2, y2): ");

scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

// Draw the original line in red

setcolor(RED);

line(x1, y1, x2, y2);

// Perform line clipping

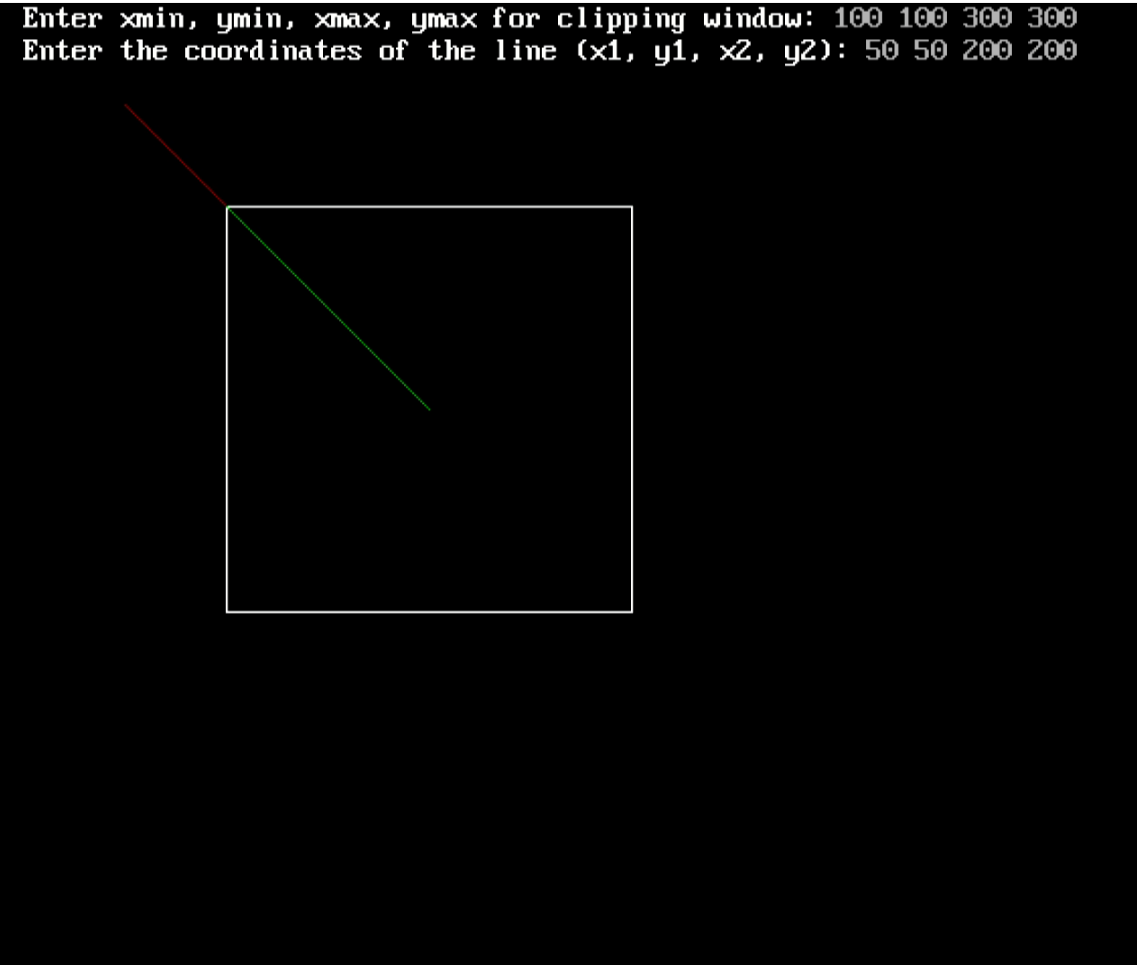
cohenSutherlandClip(x1, y1, x2, y2);

getch();

closegraph();

return 0;

}



**EXPERIMENT – 7**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include <iostream.h>

#include <math.h>

#include <graphics.h>

#include <conio.h>

#define M 10

class graph

{

private:

int A[M][M], ans[M][M], trans[3][3], color, n;

float rotat[2][2], scal[2][2];

public:

graph();

void drawline(float, float, float, float);

void polynomial();

int sign(float);

void Showpoly();

void translation();

void Scaling();

void Rotation();

void Display();

void tryi();

};

graph::graph()

{

color = RED;

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

{

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

{

if (i == j)

trans[i][j] = 1;

else

trans[i][j] = 0;

}

}

scal[0][0] = 1;

scal[0][1] = 0;

scal[1][0] = 0;

scal[1][1] = 1;

}

void graph::tryi()

{

int t;

cout << "Enter rotation angle value in degree : ";

cin >> t;

cout << sin(t \* 180 / 3.14) << " ";

}

void graph::drawline(float x1, float y1, float x2, float y2)

{

int dx, dy, length, x, y;

dx = abs(x2 - x1);

dy = abs(y2 - y1);

if (dx >= dy)

length = dx;

else

length = dy;

dx = (x2 - x1) / length;

dy = (y2 - y1) / length;

x = x1 + 0.5 \* sign(dx);

y = y1 + 0.5 \* sign(dy);

int i = 0;

while (i <= length)

{

x = x + dx;

y = y + dy;

putpixel(x, y, color);

i = i + 1;

}

}

int graph::sign(float a)

{

if (a < 0)

return -1;

else if (a == 0)

return 0;

else

return 1;

}

void graph::polynomial()

{

cout << "Enter number of vertex: ";

cin >> n;

cout << "\n -->> please Enter vertex coordinates Serially: \n";

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

{

cout << "\nX coordinate and Y coordinate for " << i + 1 << " point: ";

cin >> A[i][0] >> A[i][1];

A[i][2] = 1;

}

}

void graph::Showpoly()

{

color = RED;

for (int i = 0; i < n - 1; i++)

{

line(A[i][0], A[i][1], A[i + 1][0], A[i + 1][1]);

cout << A[i][0] << " " << A[i][1] << " " << A[i + 1][0] << " " << A[i + 1][1] << "\n";

}

line(A[0][0], A[0][1], A[n - 1][0], A[n - 1][1]);

}

void graph::translation()

{

int tx, ty;

color = GREEN;

cout << "Enter translation value tx, ty: ";

cin >> tx >> ty;

trans[2][0] = tx;

trans[2][1] = ty;

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

{

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

{

ans[i][j] = A[i][0] \* trans[0][j] + A[i][1] \* trans[1][j] + A[i][2] \* trans[2][j];

}

}

Display();

}

void graph::Scaling()

{

int sx, sy;

color = YELLOW;

cout << "Enter Scaling value sx, sy: ";

cin >> sx >> sy;

scal[0][0] = sx;

scal[1][1] = sy;

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

{

for (int j = 0; j < 2; j++)

{

ans[i][j] = A[i][0] \* scal[0][j] + A[i][1] \* scal[1][j];

}

}

Display();

}

void graph::Rotation()

{

int t;

cout << "Enter rotation angle value in degree: ";

cin >> t;

rotat[0][0] = cos(t \* 3.14 / 180);

rotat[0][1] = -sin(t \* 3.14 / 180);

rotat[1][0] = sin(t \* 3.14 / 180);

rotat[1][1] = cos(t \* 3.14 / 180);

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

{

for (int j = 0; j < 2; j++)

{

ans[i][j] = A[i][0] \* rotat[0][j] + A[i][1] \* rotat[1][j];

}

}

Display();

}

void graph::Display()

{

for (int i = 0; i < n - 1; i++)

{

line(ans[i][0], ans[i][1], ans[i + 1][0], ans[i + 1][1]);

}

line(ans[0][0], ans[0][1], ans[n - 1][0], ans[n - 1][1]);

}

int main()

{

graph obj;

int gd = DETECT, gm;

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

int ch;

while (1)

{

cout << "\n\nEnter choice:\n 1. Create polynomial\n";

cout << " 2. Show polynomial\n ";

cout << " 3. Translation\n 4. Scaling\n 5. Rotation\n 6. Exit program\n";

cout << "----> choice: ";

cin >> ch;

cout << endl;

switch (ch)

{

case 1:

obj.polynomial();

break;

case 2:

obj.Showpoly();

break;

case 3:

obj.translation();

break;

case 4:

obj.Scaling();

break;

case 5:

obj.Rotation();

break;

case 6:

closegraph();

return 0;

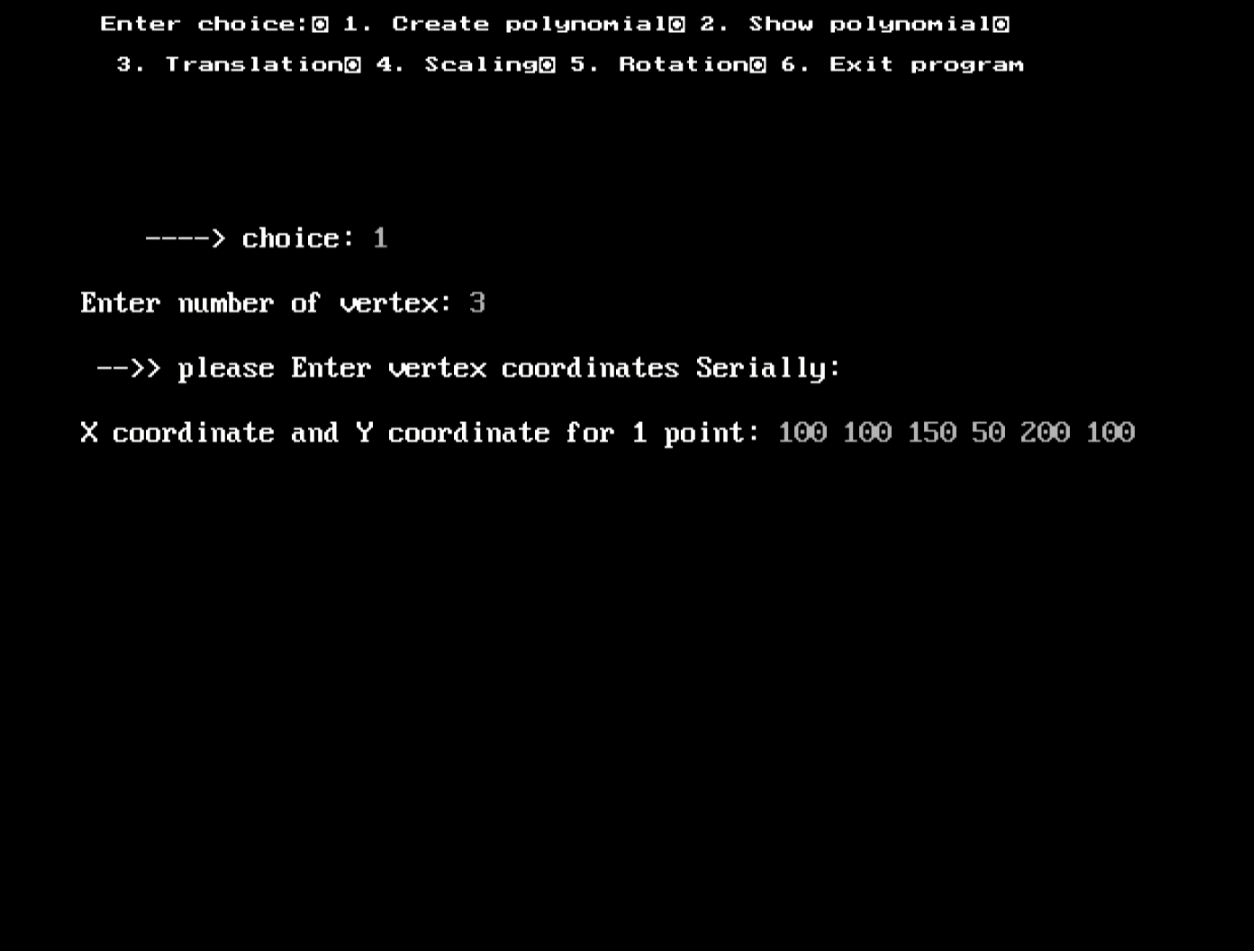
default:

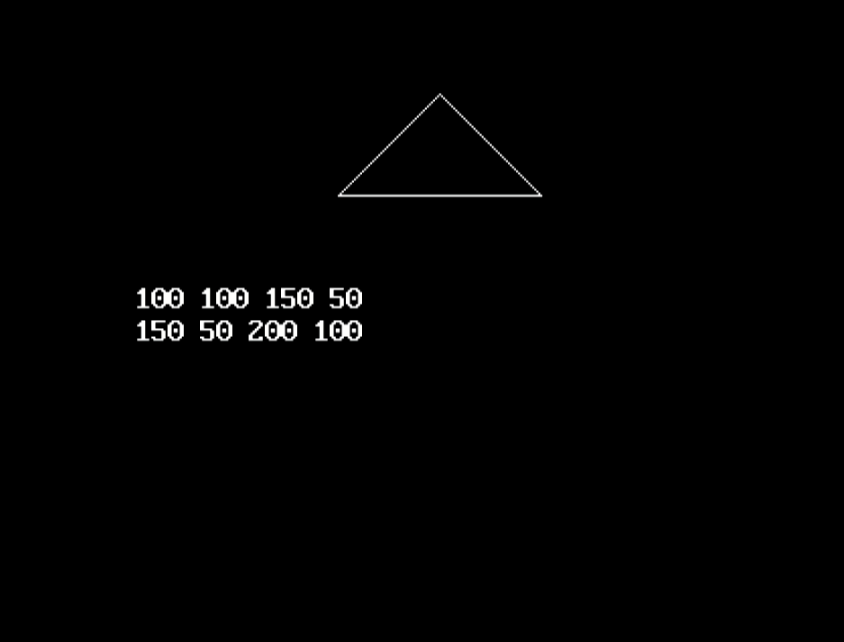
cout << "\nWrong Choice Please Try Again...";

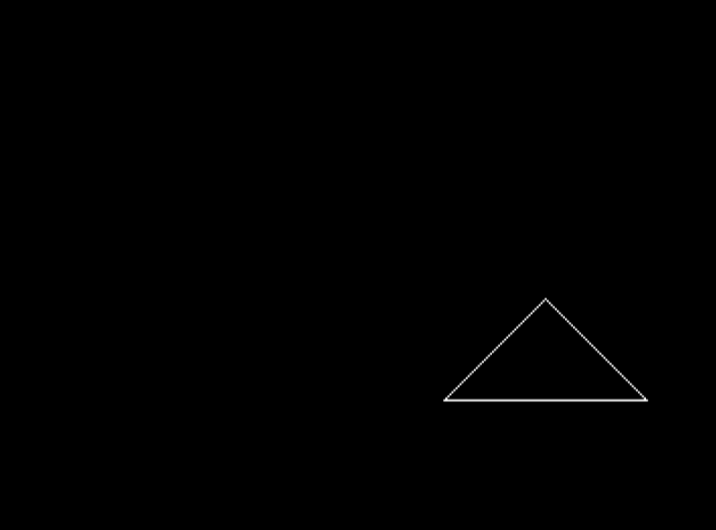
}

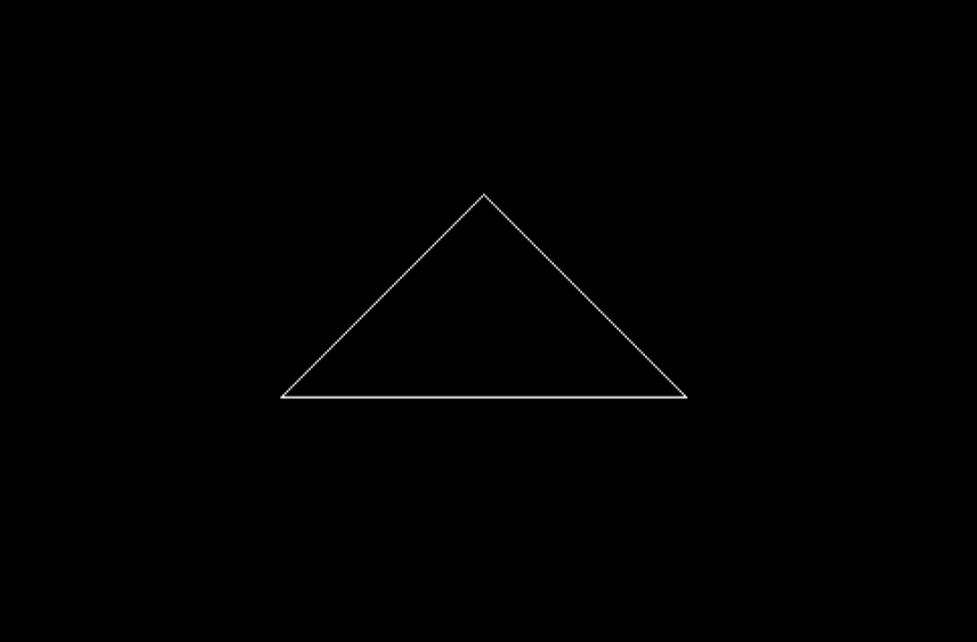
}

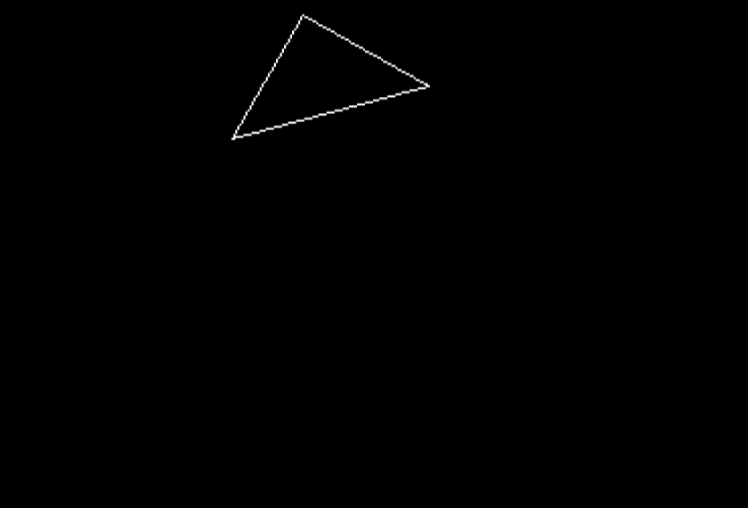
}











**EXPERIMENT – 8**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<iostream.h>

#include<dos.h>

float points[4][2];

int xa;

int ya;

void line1(float x2,float y2)

{

line(points[0][0], points[0][1], x2, y2);

points[0][0] = x2;

points[0][1] = y2;

}

void bezier(float xa, float ya, float xb, float yb, float xc, float yc, float xd, float yd, int n)

{

float xab, yab, xbc, ybc, xcd, ycd, xabc, yabc, xbcd, ybcd, xabcd, yabcd;

if(n == 0)

{

line1(xb, yb);

line1(xc, yc);

line1(xd, yd);

delay(100);

}

else

{

xab = (points[0][0] + xb) / 2;

yab = (points[0][1] + yb) / 2;

xbc = (xb + xc) / 2;

ybc = (yb + yc) / 2;

xcd = (xc + xd) / 2;

ycd = (yc + yd) / 2;

xabc = (xab + xbc) / 2;

yabc = (yab + ybc) / 2;

xbcd = (xbc + xcd) / 2;

ybcd = (ybc + ycd) / 2;

xabcd = (xabc + xbcd) / 2;

yabcd = (yabc + ybcd) / 2;

n = n - 1;

bezier(xa, ya, xab, yab, xabc, yabc, xabcd, yabcd, n);

bezier(xabcd, yabcd, xbcd, ybcd, xcd, ycd, xd, yd, n);

}

}

void main()

{

int gd = DETECT, gm, n, i;

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

cleardevice();

cout << "\nPlease! Enter n: ";

cin >> n;

cout << "\nEnter the values of all points coordinates: ";

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

{

cin >> points[i][0];

cin >> points[i][1];

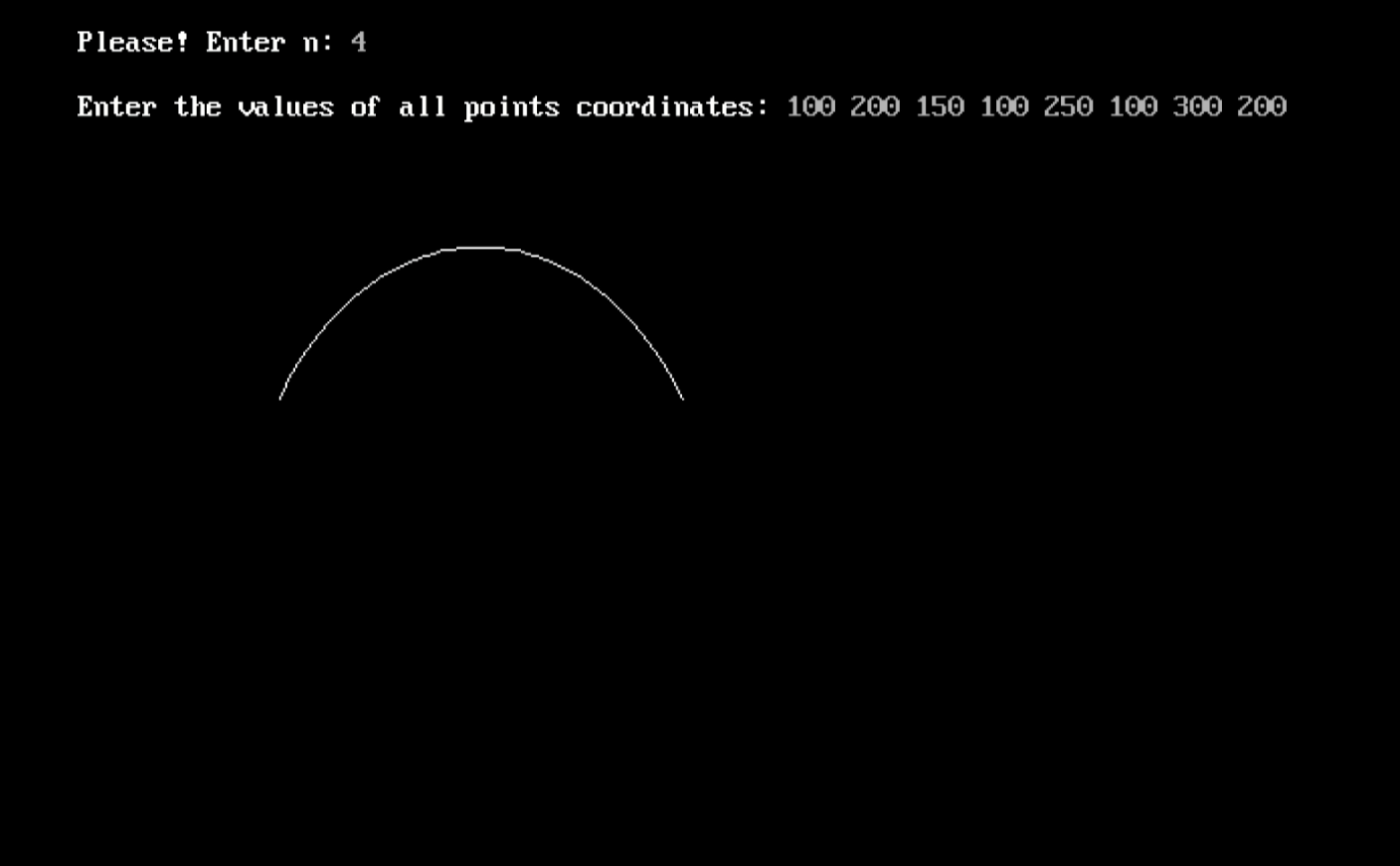
}

bezier(points[0][0], points[0][1], points[1][0], points[1][1], points[2][0], points[2][1], points[3][0], points[3][1], n);

getch();

closegraph();

}



**EXPERIMENT – 9**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include<iostream.h>

#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<dos.h>

#include<stdio.h>

void koch(int x1, int y1, int x2, int y2, int it)

{

float angle = 60 \* M\_PI / 180;

int x3 = (2 \* x1 + x2) / 3;

int y3 = (2 \* y1 + y2) / 3;

int x4 = (x1 + 2 \* x2) / 3;

int y4 = (y1 + 2 \* y2) / 3;

int x = x3 + (x4 - x3) \* cos(angle) + (y4 - y3) \* sin(angle);

int y = y3 - (x4 - x3) \* sin(angle) + (y4 - y3) \* cos(angle);

if (it > 0)

{

koch(x1, y1, x3, y3, it - 1);

koch(x3, y3, x, y, it - 1);

koch(x, y, x4, y4, it - 1);

koch(x4, y4, x2, y2, it - 1);

}

else

{

line(x1, y1, x3, y3);

line(x3, y3, x, y);

line(x, y, x4, y4);

line(x4, y4, x2, y2);

}

delay(100);

}

int main(void)

{

int gd = DETECT, gm;

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

int x1 = 100, y1 = 100, x2 = 400, y2 = 400;

int iteration;

cout << "\n Enter the number of iterations: ";

cin >> iteration;

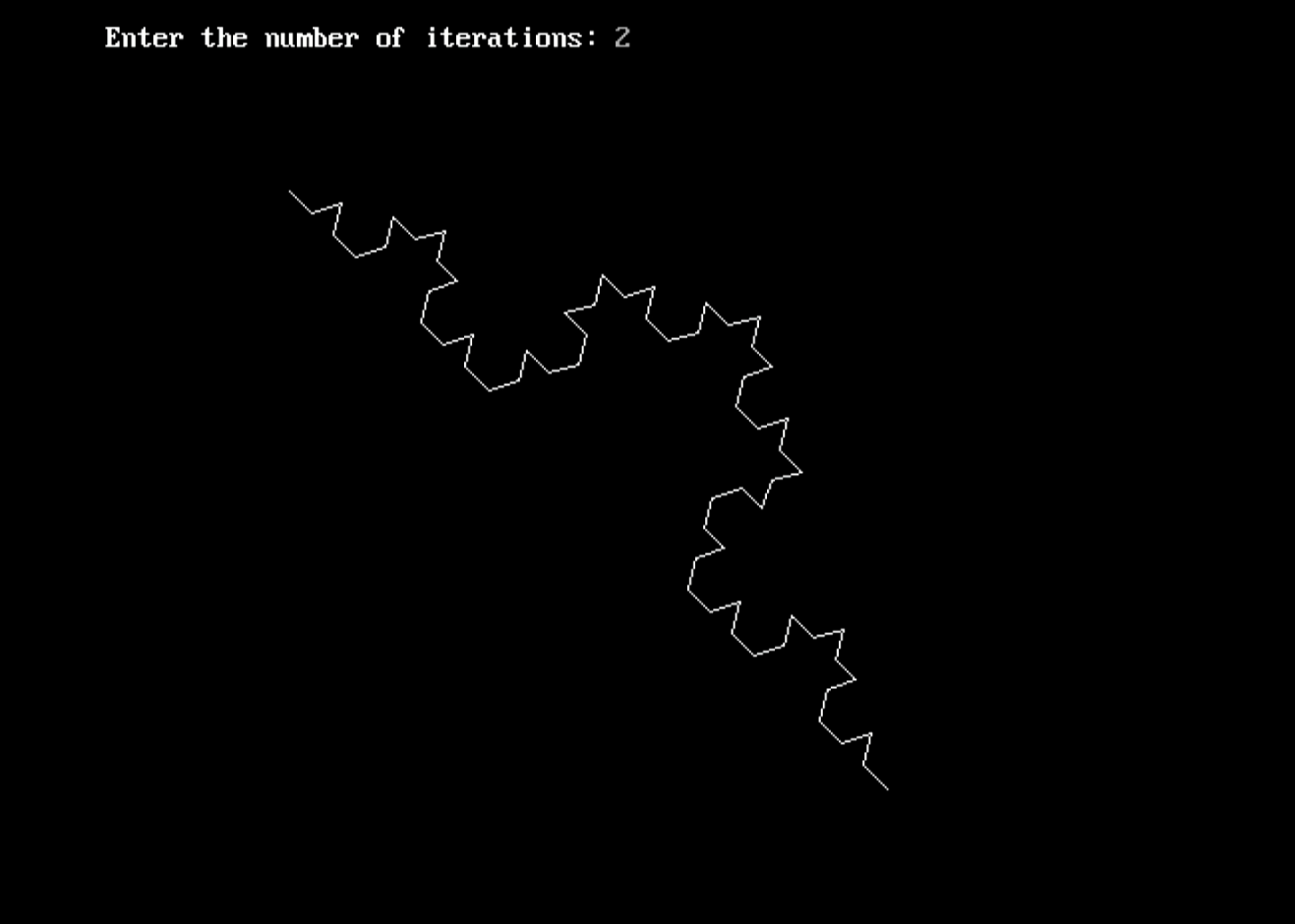
koch(x1, y1, x2, y2, iteration);

getch(); // Wait for key press before exiting

closegraph(); // Close the graphics window

return 0;

}



**EXPERIMENT – 10**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include <graphics.h>

#include <conio.h> // For getch()

#include <dos.h> // For delay()

int main() {

int gd = DETECT, gm;

int i, maxx, midy;

/\* initialize graphic mode \*/

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

/\* maximum pixel in horizontal axis \*/

maxx = getmaxx();

/\* mid pixel in vertical axis \*/

midy = getmaxy() / 2;

for (i = 0; i < maxx - 120; i = i + 4) {

/\* clears screen \*/

cleardevice();

/\* draw a white road \*/

setcolor(WHITE);

line(0, midy + 37, maxx, midy + 37);

/\* Draw Car \*/

setcolor(YELLOW);

line(i, midy + 23, i, midy);

line(i, midy, 40 + i, midy - 20);

line(40 + i, midy - 20, 80 + i, midy - 20);

line(80 + i, midy - 20, 100 + i, midy);

line(100 + i, midy, 120 + i, midy);

line(120 + i, midy, 120 + i, midy + 23);

line(0 + i, midy + 23, 18 + i, midy + 23);

arc(30 + i, midy + 23, 0, 180, 12);

line(42 + i, midy + 23, 78 + i, midy + 23);

arc(90 + i, midy + 23, 0, 180, 12);

line(102 + i, midy + 23, 120 + i, midy + 23);

/\* Draw Roof \*/

line(28 + i, midy, 43 + i, midy - 15);

line(43 + i, midy - 15, 57 + i, midy - 15);

line(57 + i, midy - 15, 57 + i, midy);

line(57 + i, midy, 28 + i, midy);

line(62 + i, midy - 15, 77 + i, midy - 15);

line(77 + i, midy - 15, 92 + i, midy);

line(92 + i, midy, 62 + i, midy);

line(62 + i, midy, 62 + i, midy - 15);

/\* Draw Wheels \*/

setcolor(BLUE);

circle(30 + i, midy + 25, 9);

circle(90 + i, midy + 25, 9);

floodfill(30 + i, midy + 25, BLUE);

floodfill(90 + i, midy + 25, BLUE);

/\* Add delay of 0.05 seconds \*/

delay(50);

}

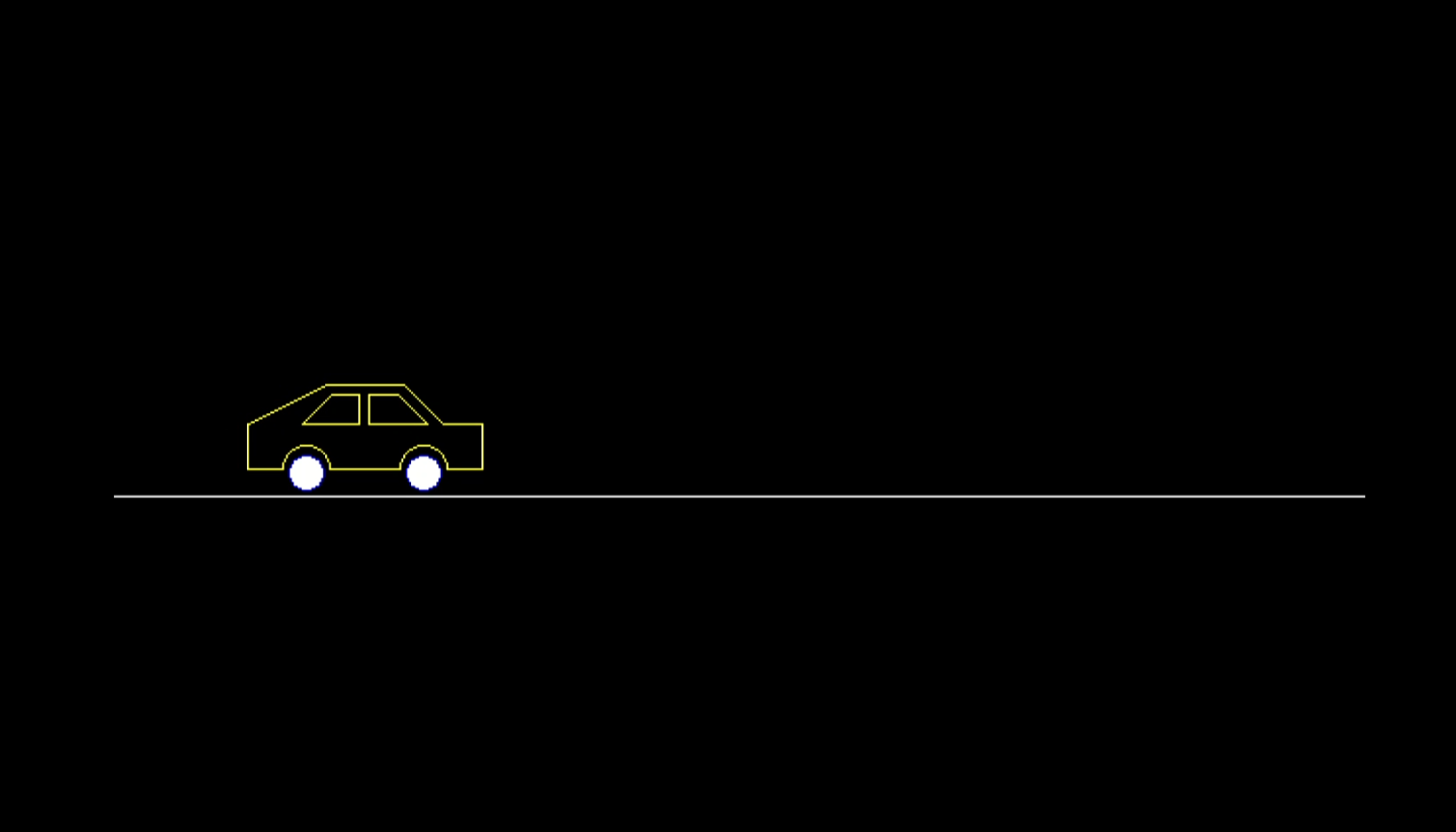
getch();

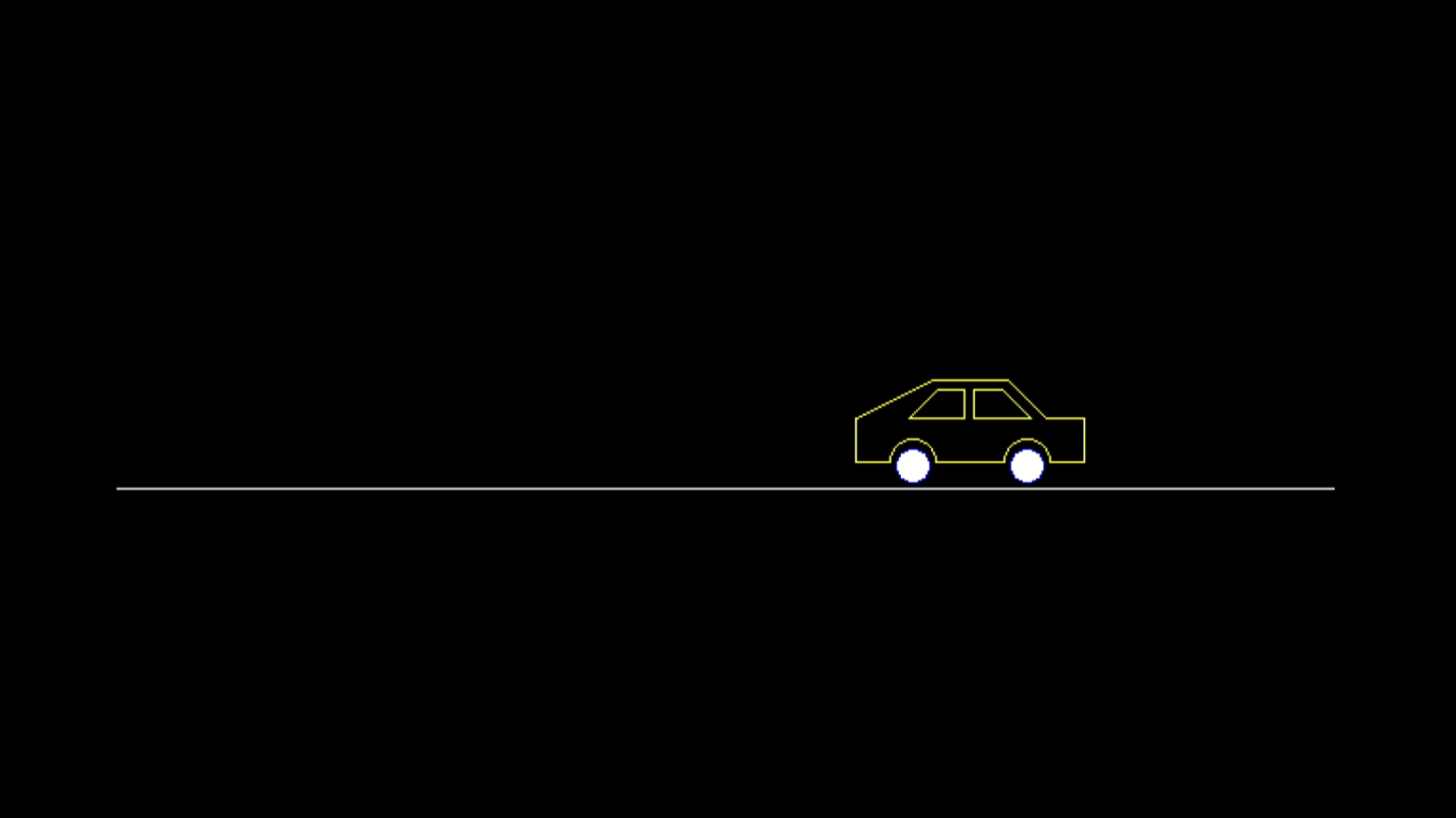
closegraph();

getch(); // Keep the screen open after closing graphics

return 0;

}





**EXPERIMENT – 11**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include<iostream>

#include<math.h>

#include<GL/glut.h>

using namespace std;

typedef float Matrix4 [4][4];

Matrix4 theMatrix;

static GLfloat input[8][3] =

{

{40,40,-50}, {90,40,-50}, {90,90,-50}, {40,90,-50},

{30,30,0}, {80,30,0}, {80,80,0}, {30,80,0}

};

float output[8][3];

float tx, ty, tz;

float sx, sy, sz;

float angle;

int choice, choiceRot;

void setIdentityM(Matrix4 m)

{

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

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

m[i][j] = (i == j);

}

void translate(int tx, int ty, int tz)

{

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

{

output[i][0] = input[i][0] + tx;

output[i][1] = input[i][1] + ty;

output[i][2] = input[i][2] + tz;

}

}

void scale(int sx, int sy, int sz)

{

theMatrix[0][0] = sx;

theMatrix[1][1] = sy;

theMatrix[2][2] = sz;

}

void RotateX(float angle) // Parallel to x

{

angle = angle \* 3.142 / 180;

theMatrix[1][1] = cos(angle);

theMatrix[1][2] = -sin(angle);

theMatrix[2][1] = sin(angle);

theMatrix[2][2] = cos(angle);

}

void RotateY(float angle) // Parallel to y

{

angle = angle \* 3.14 / 180;

theMatrix[0][0] = cos(angle);

theMatrix[0][2] = -sin(angle);

theMatrix[2][0] = sin(angle);

theMatrix[2][2] = cos(angle);

}

void RotateZ(float angle) // Parallel to z

{

angle = angle \* 3.14 / 180;

theMatrix[0][0] = cos(angle);

theMatrix[0][1] = sin(angle);

theMatrix[1][0] = -sin(angle);

theMatrix[1][1] = cos(angle);

}

void multiplyM()

{

// We Don't require 4th row and column in scaling and rotation

// [8][3] = [8][3] \* [3][3] // 4th not used

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

{

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

{

output[i][j] = 0;

for(int k = 0; k < 3; k++)

{

output[i][j] = output[i][j] + input[i][k] \* theMatrix[k][j];

}

}

}

}

void Axes(void)

{

glColor3f(0.0, 0.0, 0.0); // Set the color to BLACK

glBegin(GL\_LINES); // Plotting X-Axis

glVertex2s(-1000, 0);

glVertex2s(1000, 0);

glEnd();

glBegin(GL\_LINES); // Plotting Y-Axis

glVertex2s(0, -1000);

glVertex2s(0, 1000);

glEnd();

}

void draw(float a[8][3])

{

glBegin(GL\_QUADS);

glColor3f(0.7, 0.4, 0.5); // behind

glVertex3fv(a[0]);

glVertex3fv(a[1]);

glVertex3fv(a[2]);

glVertex3fv(a[3]);

glColor3f(0.8, 0.2, 0.4); // bottom

glVertex3fv(a[0]);

glVertex3fv(a[1]);

glVertex3fv(a[5]);

glVertex3fv(a[4]);

glColor3f(0.3, 0.6, 0.7); // left

glVertex3fv(a[0]);

glVertex3fv(a[4]);

glVertex3fv(a[7]);

glVertex3fv(a[3]);

glColor3f(0.2, 0.8, 0.2); // right

glVertex3fv(a[1]);

glVertex3fv(a[2]);

glVertex3fv(a[6]);

glVertex3fv(a[5]);

glColor3f(0.7, 0.7, 0.2); // up

glVertex3fv(a[2]);

glVertex3fv(a[3]);

glVertex3fv(a[7]);

glVertex3fv(a[6]);

glColor3f(1.0, 0.1, 0.1);

glVertex3fv(a[4]);

glVertex3fv(a[5]);

glVertex3fv(a[6]);

glVertex3fv(a[7]);

glEnd();

}

void init()

{

glClearColor(1.0, 1.0, 1.0, 1.0); // set background color to white

glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);

// Set the no. of Co-ordinates along X & Y axes and their gappings

glEnable(GL\_DEPTH\_TEST);

// To render the surfaces properly according to their depths

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

Axes();

glColor3f(1.0, 0.0, 0.0);

draw(input);

setIdentityM(theMatrix);

switch (choice)

{

case 1:

translate(tx, ty, tz);

break;

case 2:

scale(sx, sy, sz);

multiplyM();

break;

case 3:

switch (choiceRot)

{

case 1:

RotateX(angle);

break;

case 2:

RotateY(angle);

break;

case 3:

RotateZ(angle);

break;

default:

break;

}

multiplyM();

break;

}

draw(output);

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(1362, 750);

glutInitWindowPosition(0, 0);

glutCreateWindow("3D TRANSFORMATIONS");

init();

cout << "Enter your choice number:\n1.Translation\n2.Scaling\n3.Rotation\n=> ";

cin >> choice;

switch (choice)

{

case 1:

cout << "\nEnter Tx, Ty & Tz: \n";

cin >> tx >> ty >> tz;

break;

case 2:

cout << "\nEnter Sx, Sy & Sz: \n";

cin >> sx >> sy >> sz;

break;

case 3:

cout << "Enter your choice for Rotation about axis:\n1.parallel to X-axis (y & z)\n2.parallel to Y-axis (x & z)\n3.parallel to Z-axis (x & y)\n=> ";

cin >> choiceRot;

cout << "\nEnter Rotation angle: ";

cin >> angle;

break;

default:

break;

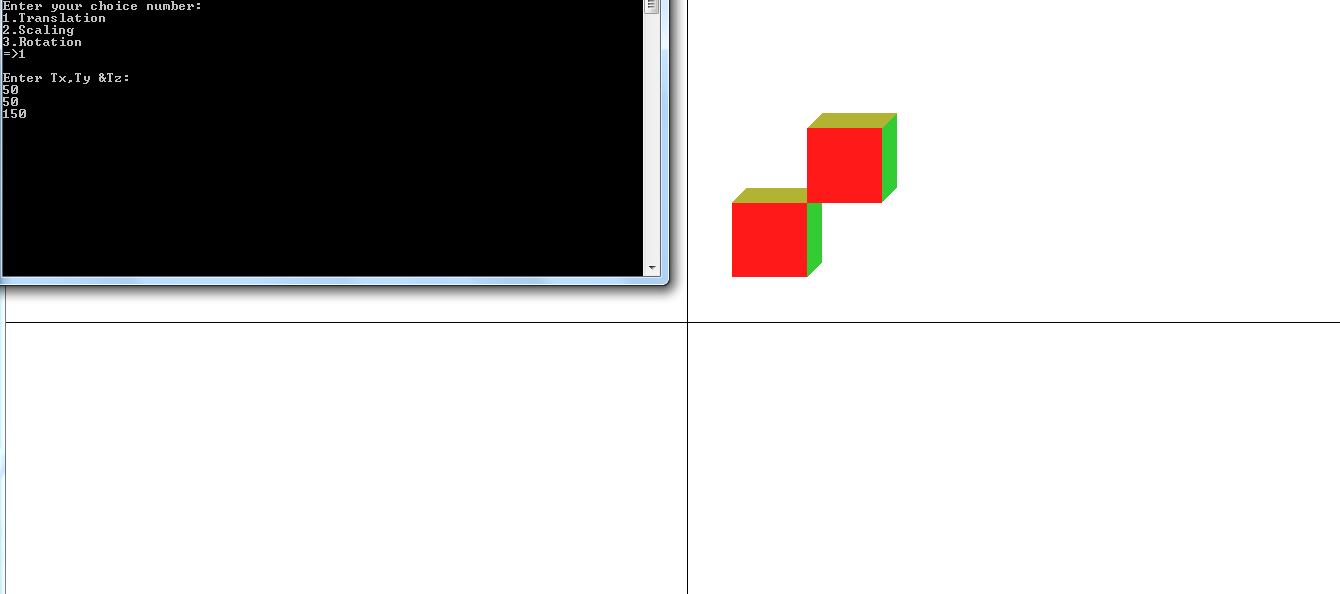
}

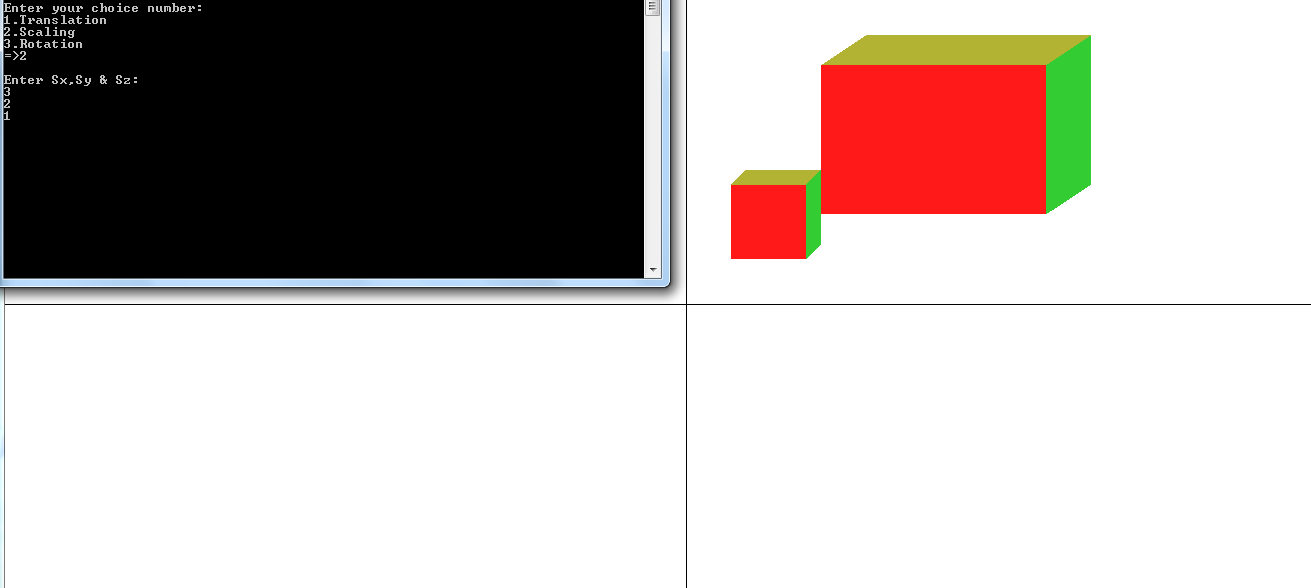
glutDisplayFunc(display);

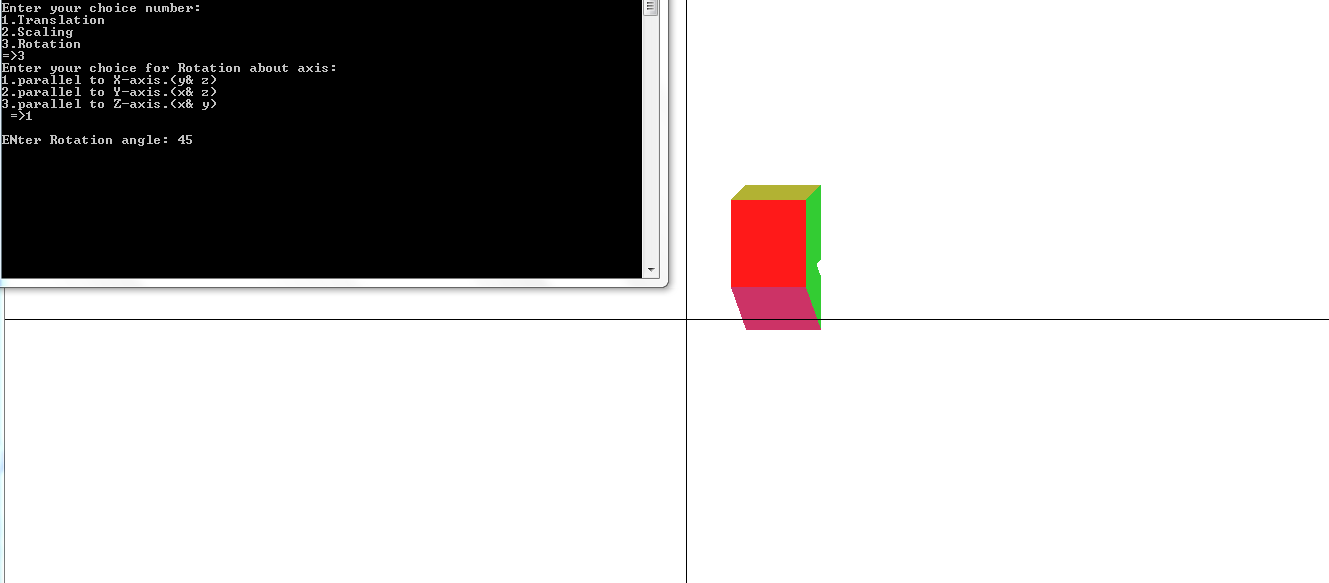
glutMainLoop();

return 0;

}







**EXPERIMENT – 12**

**// Name – Manthan Gajanan Kshirsagar**

**// Div – A**

**// Roll no – 58**

#include <stdio.h> // Use stdio.h for Turbo C++

#include <conio.h> // Use conio.h for getch()

char board[3][3] = { {'1', '2', '3'}, {'4', '5', '6'}, {'7', '8', '9'} };

// Function prototypes

void displayBoard();

int checkWin();

int checkDraw();

void makeMove(char player);

int main() {

char player = 'X';

int gameWon = 0, draw = 0;

// Game loop

while (!gameWon && !draw) {

displayBoard();

makeMove(player);

gameWon = checkWin();

draw = checkDraw();

if (!gameWon && !draw) {

// Switch player

player = (player == 'X') ? 'O' : 'X';

}

}

displayBoard();

if (gameWon) {

printf("Player %c wins!\n", player);

} else {

printf("It's a draw!\n");

}

getch(); // To pause the output screen

return 0;

}

// Function to display the current board

void displayBoard() {

printf(" | | \n");

printf(" %c | %c | %c \n", board[0][0], board[0][1], board[0][2]);

printf("\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\n");

printf(" | | \n");

printf(" %c | %c | %c \n", board[1][0], board[1][1], board[1][2]);

printf("\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\n");

printf(" | | \n");

printf(" %c | %c | %c \n", board[2][0], board[2][1], board[2][2]);

printf(" | | \n");

}

// Function to check if any player has won

int checkWin() {

// Check rows and columns

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

if (board[i][0] == board[i][1] && board[i][1] == board[i][2])

return 1;

if (board[0][i] == board[1][i] && board[1][i] == board[2][i])

return 1;

}

// Check diagonals

if (board[0][0] == board[1][1] && board[1][1] == board[2][2])

return 1;

if (board[0][2] == board[1][1] && board[1][1] == board[2][0])

return 1;

return 0;

}

// Function to check if the game is a draw

int checkDraw() {

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

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

if (board[i][j] != 'X' && board[i][j] != 'O') {

return 0; // There's still a move left

}

}

}

return 1; // No moves left

}

// Function to make a move for the current player

void makeMove(char player) {

int choice;

printf("Player %c, enter your move (1-9): ", player);

scanf("%d", &choice);

int row = (choice - 1) / 3;

int col = (choice - 1) % 3;

// Check if the chosen spot is valid

if (choice >= 1 && choice <= 9 && board[row][col] != 'X' && board[row][col] != 'O') {

board[row][col] = player;

} else {

printf("Invalid move! Try again.\n");

makeMove(player);

}

}

