

**FAR-WESTERN UNIVERSITY**

**SCHOOL OF ENGINEERING**

**NEPAL**

A PRACTICAL REPORT OF

“COMPUTER GRAPHICS.”

**SUMITTED BY:**

NAME: SANTOSH UPADHYAY

ROLL NO.: 41

YEAR/SEM: 3**RD** Year/ 5**TH** Sem

**SUBMITTED TO:**

ER. BHIM BDR. PUN

DEPARTMENT OF ENGINEERING

SOE, FWU

|  |
| --- |
| **INDEX** |

|  |
| --- |
| NAME: SANTOSH UPADHYAY  FACULTY: COMPUTER ENGINEERING ROLL NO.: 41  SUBJECT: COMPUTER GRAPHICS SYMBOL NO.: 8050143  UNIVERSITY: FAR-WESTERN UNIVERSITY, SoE |

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Title | Date | Remarks |
| 1 | To draw line using DDA Algorithm. | 2080/11/13 |  |
| 2 | To draw a moving car using inbuilt functions. | 2080/11/13 |  |
| 3 | To draw a line using Bresenham’s Algorithm. | 2080/11/14 |  |
| 4 | To draw a circle using Mid-Point circle drawing Algorithm. | 2080/11/14 |  |
| 5 | To draw an ellipse using Mid-Point ellipse drawing Algorithm. | 2080/11/15 |  |
| 6 | To perform 2D Translation of point & line. | 2080/11/16 |  |
| 7 | To perform Rotation & Scaling of 2D object. | 2080/11/18 |  |
| 8 | Parallel Projection & Perspective projection of 3D object. | 2080/11/18 |  |
| 9 | To draw a 2D object using Graphics. | 2080/11/15 |  |

1. **DDA ALGORITHM**

#include <graphics.h>

#include <iostream>

#include<conio.h>

#include<math.h>

using namespace std;

int main( )

{

float x,y,x0,y0,xm,ym,dx,dy,s;

int i,gd=DETECT,gm;

cout<<"Enter the value of x0 and y0 : ";

cin>>x0>>y0;

cout<<"Enter the value of xmax and ymax: ";

cin>>xm>>ym;

dx=abs(xm-x0);

dy=abs(ym-y0);

if(dx>=dy)

s=dx;

else

s=dy;

dx=dx/s;

dy=dy/s;

x=x0;

y=y0;

initgraph(&gd,&gm,NULL);

setcolor(YELLOW);

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

{

putpixel(x,y,YELLOW);

x=x+dx;

y=y+dy;

}

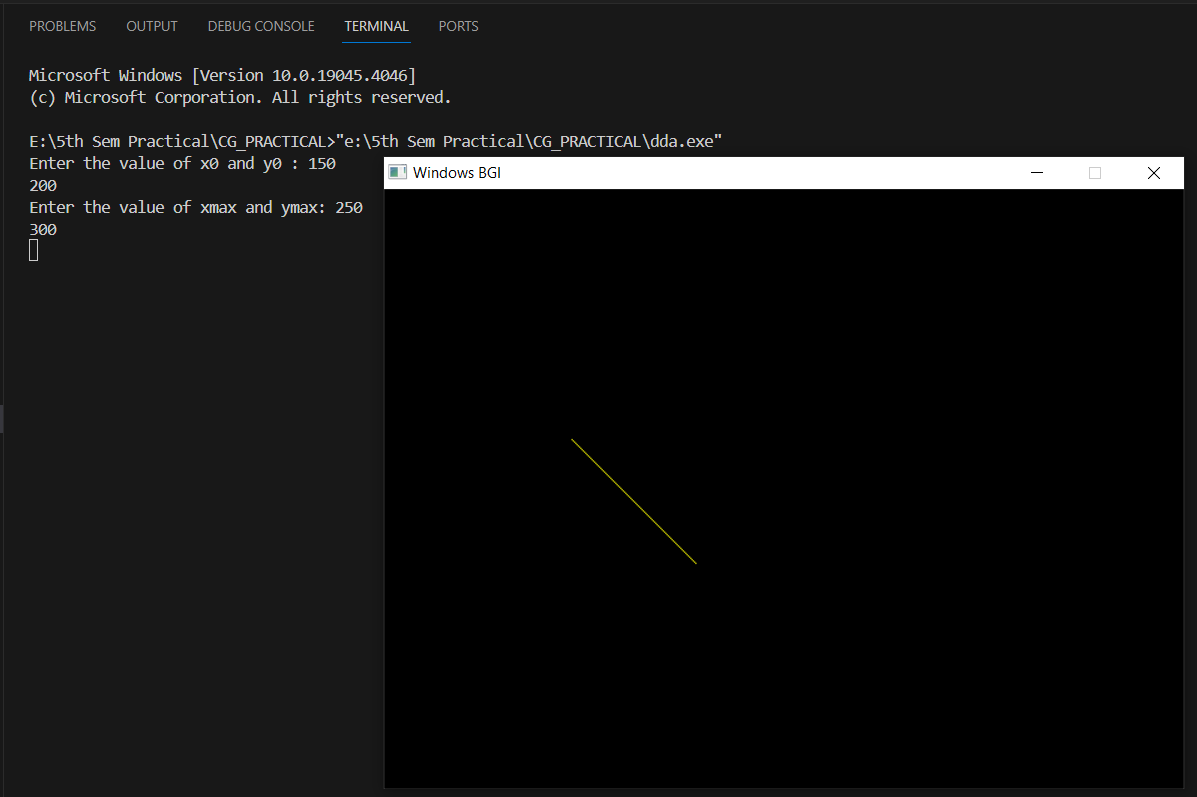
getche();

closegraph();

return 0;

}

**OUTPUT:**



1. **DRAW CAR**

#include <graphics.h>

#include <stdio.h>

void draw\_moving\_car(void) {

int i, j = 0, gd = DETECT, gm;

initgraph(&gd, &gm,NULL);

for (i = 0; i <= 420; i = i + 10) {

setcolor(YELLOW);

line(0 + i, 300, 210 + i, 300);

line(50 + i, 300, 75 + i, 270);

line(75 + i, 270, 150 + i, 270);

line(150 + i, 270, 165 + i, 300);

line(0 + i, 300, 0 + i, 330);

line(210 + i, 300, 210 + i, 330);

circle(65 + i, 330, 15);

circle(65 + i, 330, 2);

circle(145 + i, 330, 15);

circle(145 + i, 330, 2);

line(0 + i, 330, 50 + i, 330);

line(80 + i, 330, 130 + i, 330);

line(210 + i, 330, 160 + i, 330);

delay(100);

cleardevice();

}

getch();

closegraph();

}

int main()

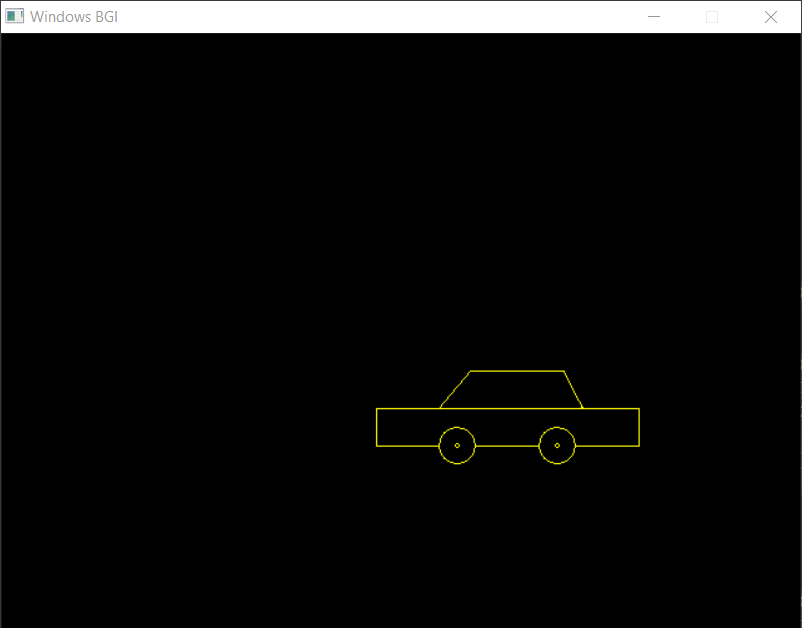
{

draw\_moving\_car();

return 0;

}

**OUTPUT:**



1. **BRESENHAM’S ALGORITHM**

#include<iostream>

#include <stdio.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, YELLOW);

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;

int x0,y0,x1,y1;

cout<<"Enter the Values of starting Point:";

cin>>x0>>y0;

cout<<"Enter the Values of End Point:";

cin>>x1>>y1;

initgraph(&gdriver, &gmode, NULL);

drawline(x0, y0, x1, y1);

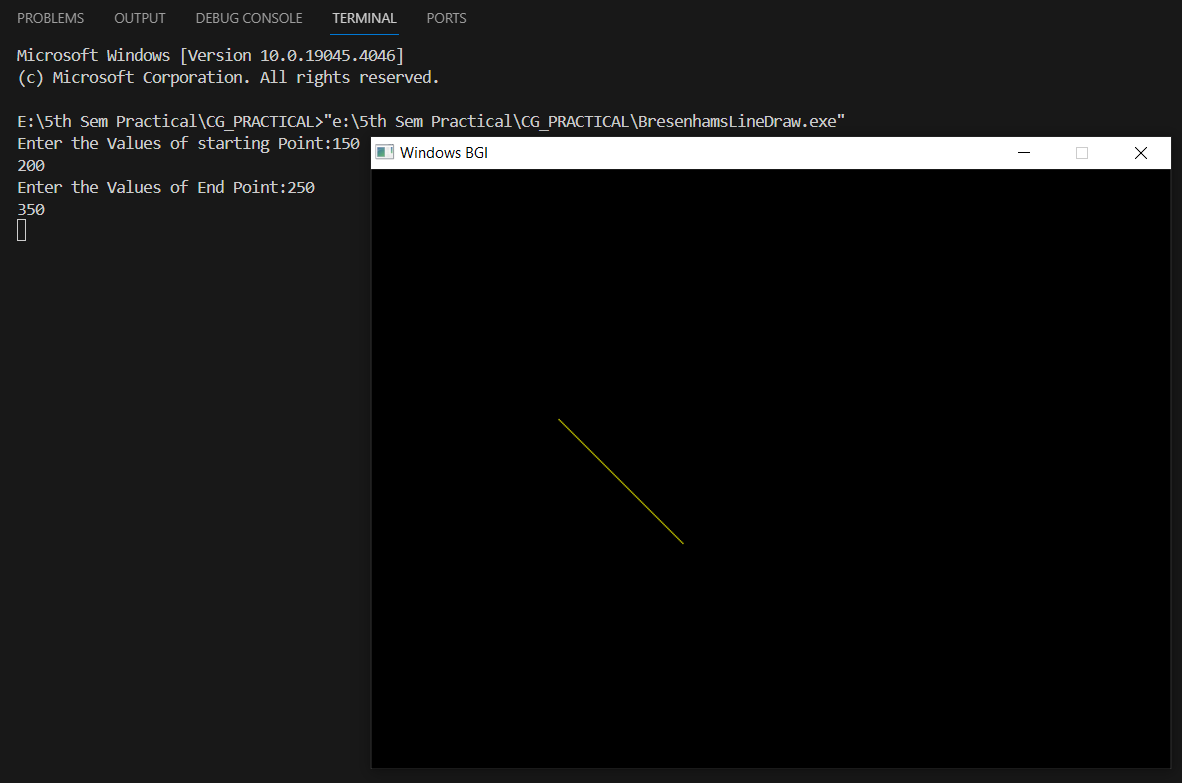
getch();

closegraph();

return 0;

}

**OUTPUT:**



1. **MID-POINT CIRCLE DRAWING ALGORITHM**

#include<graphics.h>

#include<iostream>

using namespace std;

class MidpointCircle

{

private:

int x\_center, y\_center, radius;

public:

MidpointCircle(){

cout<<"Enter value of centere (x,y) and Radius :";

cin>>x\_center>>y\_center>>radius;

}

void drawCircle()

{

int x = radius;

int y = 0;

int p = 1 - radius;

while (x > y)

{

putPixel(x + x\_center, y + y\_center);

putPixel(y + x\_center, x + y\_center);

putPixel(-y + x\_center, x + y\_center);

putPixel(-x + x\_center, y + y\_center);

putPixel(-x + x\_center, -y + y\_center);

putPixel(-y + x\_center, -x + y\_center);

putPixel(y + x\_center, -x + y\_center);

putPixel(x + x\_center, -y + y\_center);

y++;

if (p <= 0)

{

p = p + 2 \* y + 1;

}

else

{

x--;

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

}

}

}

private:

void putPixel(int x, int y)

{

putpixel(x, y, WHITE);

}

};

int main()

{

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int x=200, y=300, r=80;

MidpointCircle circle;

circle.drawCircle();

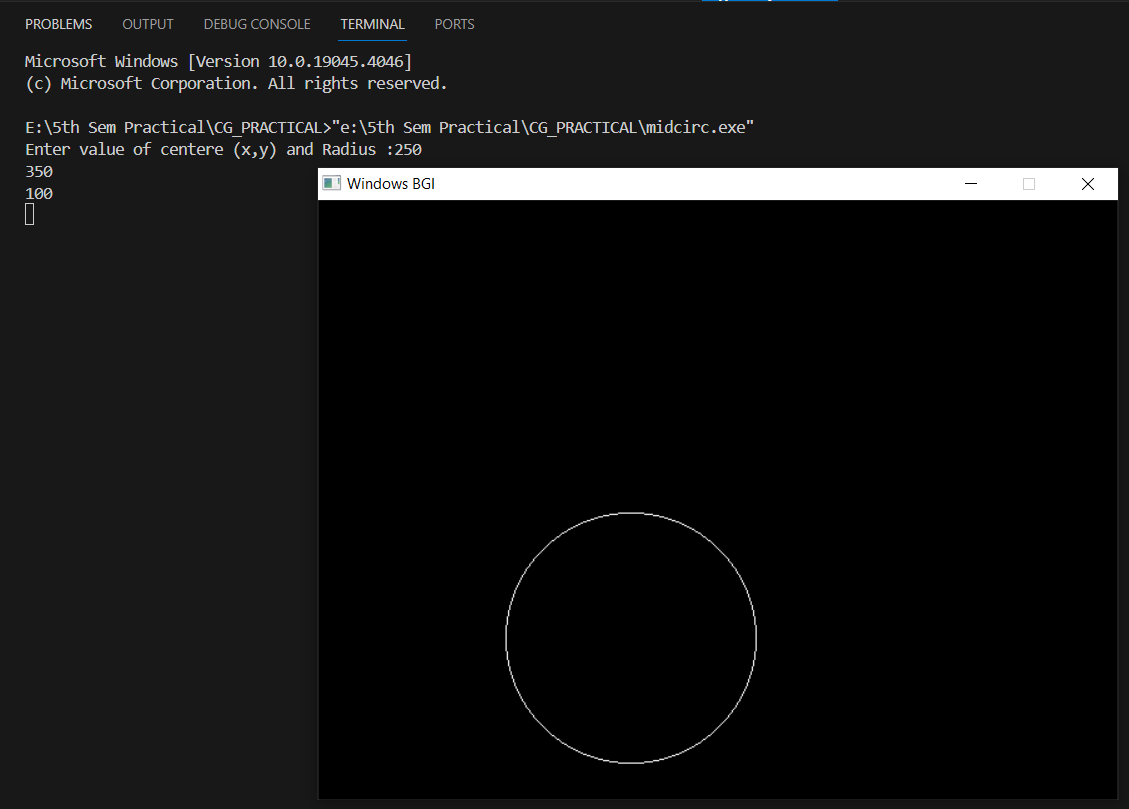
delay(5000);

closegraph();

return 0;

}

**OUTPUT:**



1. **MID-POINT ELLIPSE DRAWING ALGORITHM**

#include <graphics.h>

#include <iostream>

using namespace std;

class MidpointEllipse

{

private:

int x\_center, y\_center, a, b;

public:

MidpointEllipse(int x, int y, int majorAxis, int minorAxis) : x\_center(x), y\_center(y), a(majorAxis), b(minorAxis) {}

void drawEllipse()

{

int x = 0, y = b;

int a\_squared = a \* a;

int b\_squared = b \* b;

int two\_a\_squared = 2 \* a\_squared;

int two\_b\_squared = 2 \* b\_squared;

int four\_a\_squared = 4 \* a\_squared;

int four\_b\_squared = 4 \* b\_squared;

int decision\_parameter;

// Region 1

decision\_parameter = b\_squared - a\_squared \* b + 0.25 \* a\_squared;

while (four\_b\_squared \* x <= four\_a\_squared \* y)

{

putPixel(x, y);

x++;

if (decision\_parameter < 0)

decision\_parameter += two\_b\_squared \* x + b\_squared;

else

{

y--;

decision\_parameter += two\_b\_squared \* x - two\_a\_squared \* y + a\_squared + b\_squared;

}

}

// Region 2

decision\_parameter = b\_squared \* (x + 0.5) \* (x + 0.5) + a\_squared \* (y - 1) \* (y - 1) - a\_squared \* b\_squared;

while (y >= 0)

{

putPixel(x, y);

y--;

if (decision\_parameter > 0)

decision\_parameter += a\_squared - two\_a\_squared \* y;

else

{

x++;

decision\_parameter += two\_b\_squared \* x - two\_a\_squared \* y + a\_squared + b\_squared;

}

}

}

private:

void putPixel(int x, int y)

{

putpixel(x + x\_center, y + y\_center, WHITE);

putpixel(-x + x\_center, y + y\_center, WHITE);

putpixel(x + x\_center, -y + y\_center, WHITE);

putpixel(-x + x\_center, -y + y\_center, WHITE);

}

};

int main()

{

int gd = DETECT, gm;

int x, y, majorAxis, minorAxis;

cout<<"Enter the Value of X Coordinate of centre:";

cin>>x;

cout<<"Enter the Value of Y Coordinate of centre:";

cin>>y;

cout<<"Enter the Value of Major axis:";

cin>>majorAxis;

cout<<"Enter the Value of Minor Axis:";

cin>>minorAxis;

initgraph(&gd, &gm, NULL);

MidpointEllipse ellipse(x, y, majorAxis, minorAxis);

ellipse.drawEllipse();

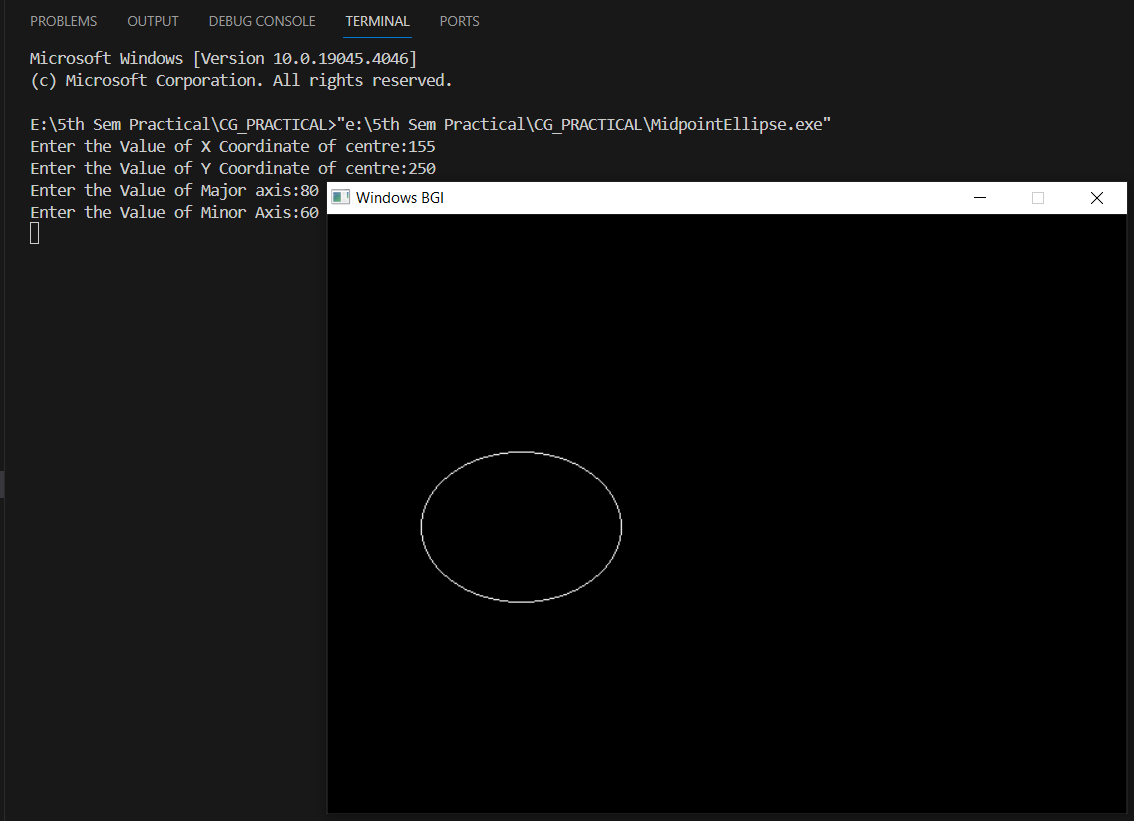
delay(5000);

getch();

closegraph();

return 0;

}

**OUTPUT:** 

1. **TRANSLATION OF POINT & LINE**

**POINT:**

#include<iostream>

#include<graphics.h>

using namespace std;

void translatePoint(int P[], int T[])

{

int gd=DETECT, gm;

initgraph(&gd,&gm, NULL);

cout<<"Original Coordinate:("<<P[0]<<","<<P[1]<<")";

putpixel(P[0],P[1],GREEN);

P[0]=P[0]+T[0];

P[1]=P[1]+T[1];

cout<<"\n Translated Coordinates: ("<<P[0]<<","<<P[1]<<")";

putpixel(P[0],P[1],YELLOW);

getch();

closegraph();

}

int main()

{

int P[2]={50, 80};

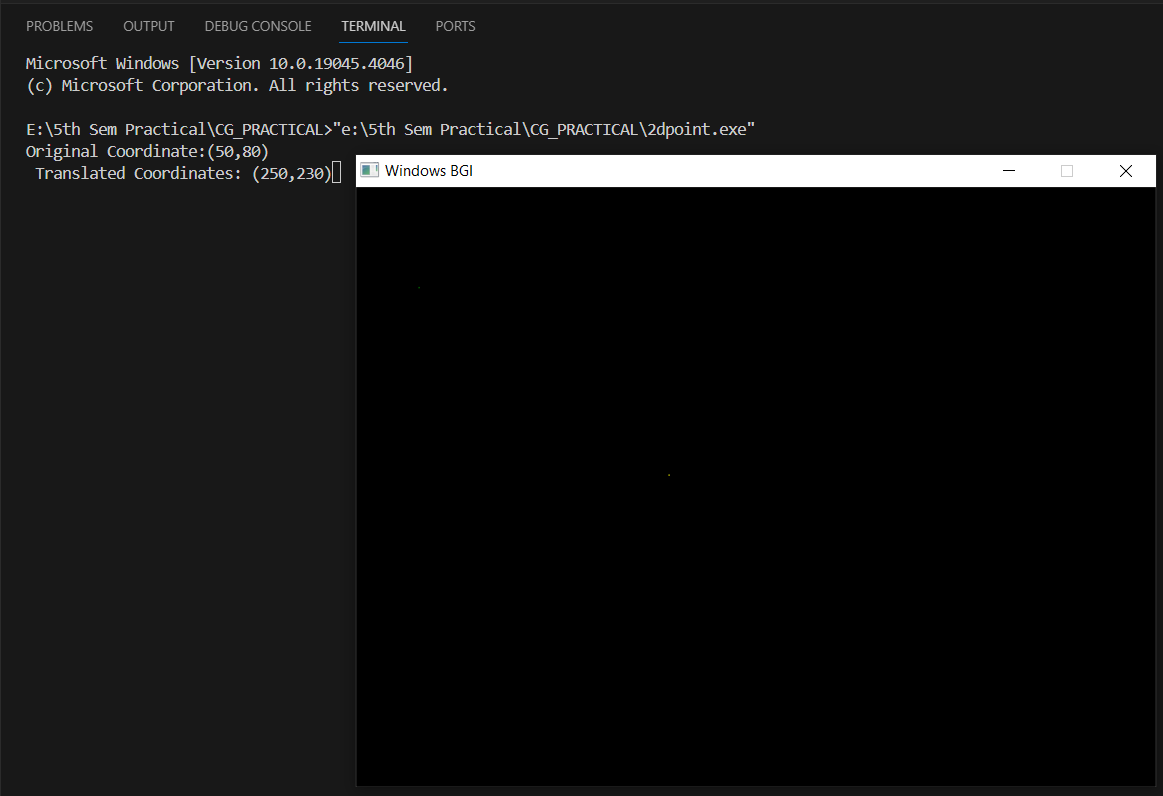
int T[2]={200, 150};

translatePoint(P, T);

return 0;

}

**OUTPUT:**



**LINE:**

#include<iostream>

#include<graphics.h>

using namespace std;

void translatePoint(int P[], int T[])

{ int gd=DETECT, gm;

initgraph(&gd,&gm, NULL);

cout<<"Original Coordinate:("<<P[0]<<","<<P[1]<<"),("<<P[2]<<","<<P[3]<<")";

line(P[0],P[1],P[2],P[3]);

P[0]=P[0]+T[0];

P[1]=P[1]+T[1];

P[2]=P[2]+T[0];

P[3]=P[3]+T[1];

cout<<"\n Translated Coordinates: ("<<P[0]<<","<<P[1]<<"),("<<P[2]<<","<<P[3]<<")";

setcolor(YELLOW);

line(P[0],P[1],P[2],P[3]);

getch();

closegraph();

}

int main()

{

int P[4];

int T[2];

cout<<"Enter x1, y1, x2 & y2 :";

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

{ cin>>P[i]; }

cout<<"Enter tx & ty :";

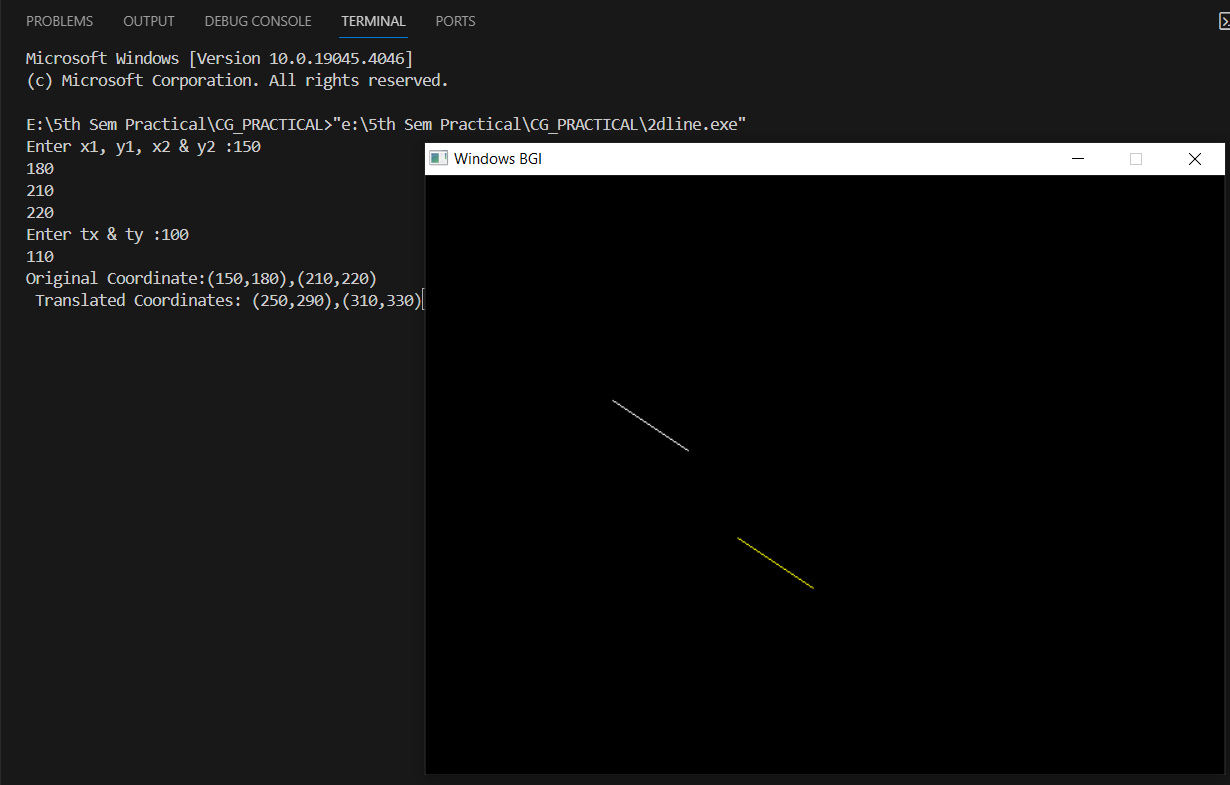
for(int i=0;i<=1;i++)

{ cin>>T[i]; }

translatePoint(P, T);

return 0;}

**OUTPUT:**



1. **ROTATION & SCALING OF 2D OBJECT**

**// File name: drawobj.cpp**

#include <stdio.h>

#include <math.h>

#include<graphics.h>

#include<iostream>

#include<graphics.h>

#include<iostream>

#define PI 3.14

using namespace std;

struct Point

{

double x;

double y;

};

void drawObject(struct Point point[],int s)

{

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

{

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

}

line(point[s-1].x,point[s-1].y,point[0].x,point[0].y);

}

void print(struct Point point[],int s)

{

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

{

cout<<"\npoint"<<i<<"\t\tx="<<point[i].x<<"\t\ty="<<point[i].y;

}

}

**ROTATION:**

**// File name: rotat.cpp**

#include"drawobj.cpp"

void rotatePoint(struct Point point[],int s, double angleDegrees)

{

int gd=DETECT,gm;

printf("\nBefore Rotation:");

print(point,s);

initgraph(&gd,&gm,NULL);

setcolor(GREEN);

drawObject(point,s);

double angleRadians = angleDegrees \* PI / 180.0;

double cosTheta = cos(angleRadians);

double sinTheta = sin(angleRadians);

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

{

double newX = point[i].x \* cosTheta - point[i].y \* sinTheta;

double newY = point[i].x \* sinTheta + point[i].y \* cosTheta;

point[i].x = newX;

point[i].y = newY;

}

printf("\nAfter Rotation:");

print(point,s);

setcolor(RED);

drawObject(point,s);

getch();

closegraph();

}

**// File name: rotate2d.cpp**

#include"rotat.cpp"

#include<graphics.h>

using namespace std;

int main()

{

int n,i,gd=DETECT,gm;

struct Point myp[100];

double sx,sy,x,y,ang;

cout<<"Enter no of points :";

cin>>n;

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

{

cout<<"Enter cordinates of p"<<i<<" :";

cin>>x>>y;

myp[i]={x,y};

}

cout<<"\nEnter angle of rotation :";

cin>>ang;

initgraph(&gd,&gm,NULL);

rotatePoint(myp,n,ang);

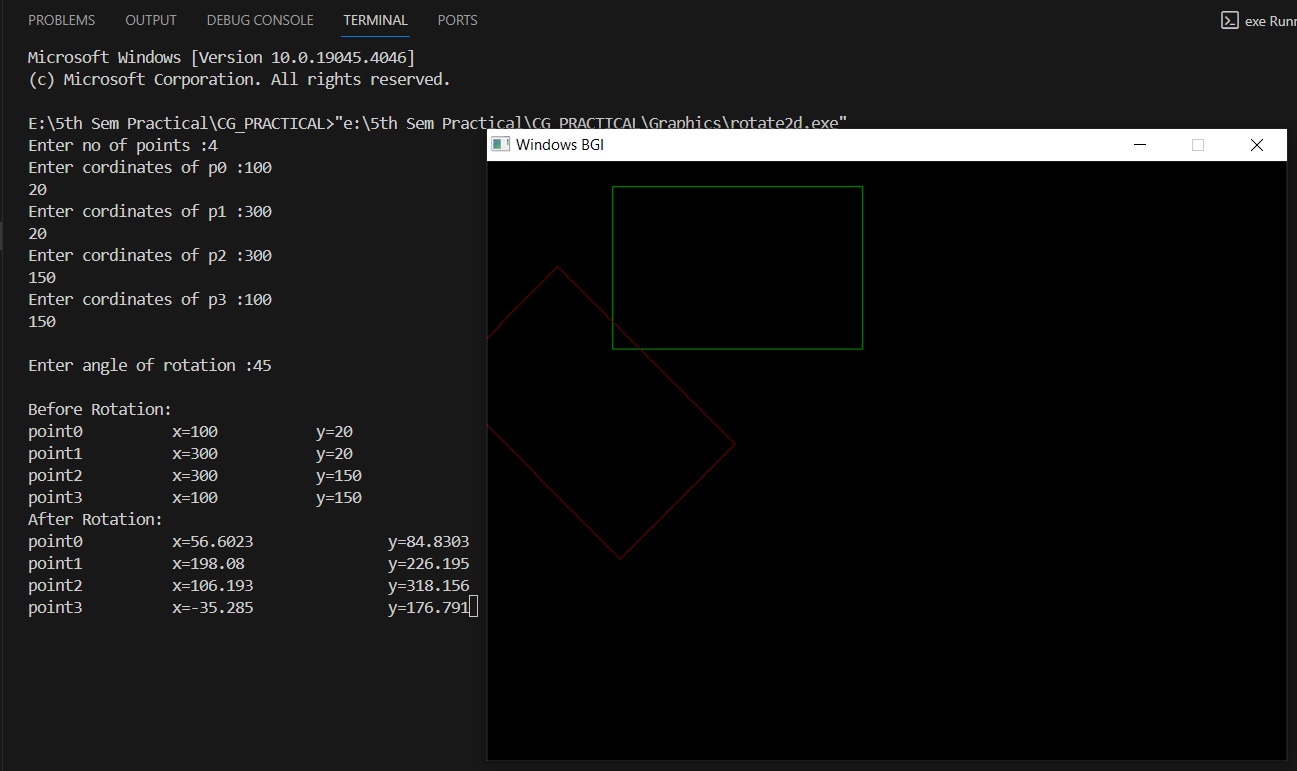
getch();

closegraph();

return 0;

}

**OUTPUT:**



**SCALING:**

**// File name: scale.cpp**

#include"drawobj.cpp"

void scalePoint(struct Point point[],int s, double scaleX, double scaleY)

{

int gd=DETECT,gm;

cout<<"\nBefore Scaling:";

print(point,s);

initgraph(&gd,&gm,NULL);

setcolor(GREEN);

drawObject(point,s);

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

{

point[i].x \*= scaleX;

point[i].y \*= scaleY;

}

cout<<"\nAfter Scaling:";

print(point,s);

setcolor(RED);

drawObject(point,s);

getch();

closegraph();

}

**//File name: Scaling2D.cpp**

#include"scale.cpp"

#include<graphics.h>

using namespace std;

int main()

{

int i,n;

struct Point myp[100];

double sx,sy,x,y;

cout<<"Enter no of points :";

cin>>n;

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

{

cout<<"Enter cordinates of p"<<i<<" :";

cin>>x>>y;

myp[i]={x,y};

}

cout<<"Enter sx & sy:";

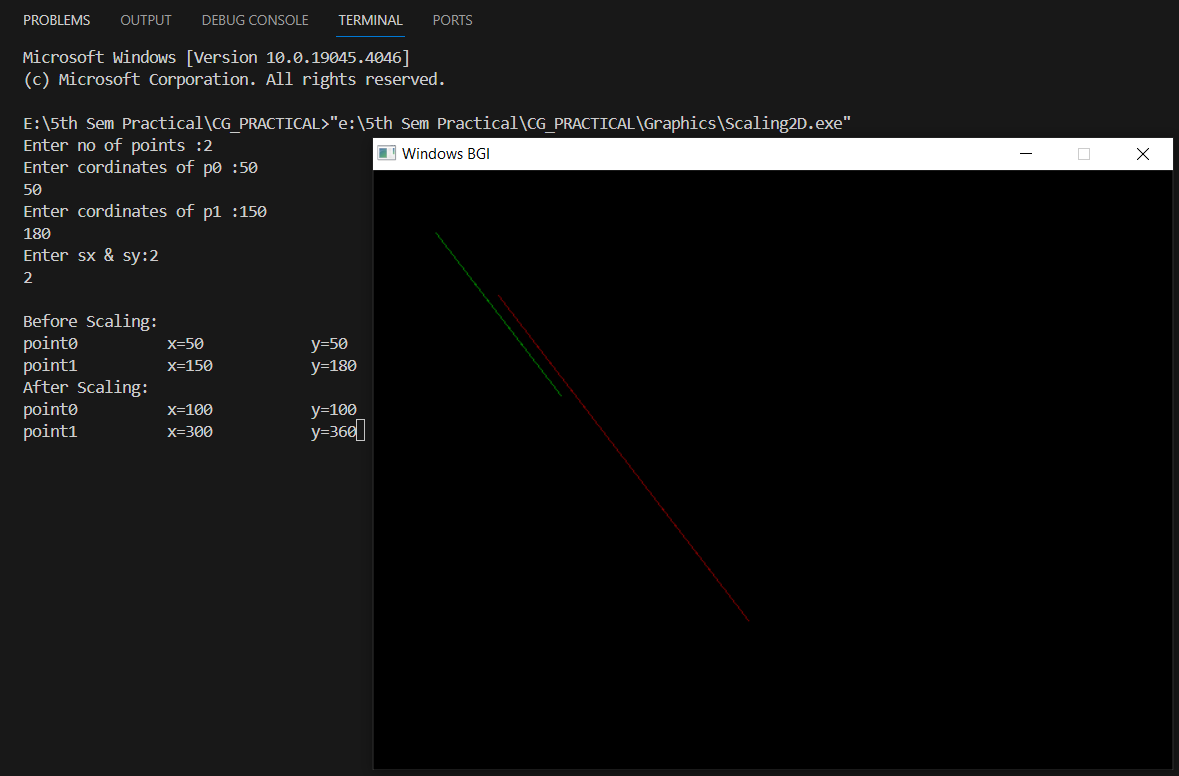
cin>>sx>>sy;

scalePoint(myp,n,sx,sy);

return 0;

}

**OUTPUT:**



1. **PARALLEL & PERSPECTIVE PROJECTION**

**//File name: parallelprojection.cpp**

#include <stdio.h>

#include"drawobj.cpp" // defined before scaling & rotation mentioned above

struct Point3D

{

double x;

double y;

double z;

};

struct Point2D

{

double x;

double y;

};

void parallelProjection(struct Point3D point3d[], struct Point point2d[],int s)

{

int gd=DETECT,gm;

cout<<"\nBefore Projetion:";

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

{

cout<<"\npoint"<<i<<"\t\tx="<<point3d[i].x<<"\t\ty="<<point3d[i].y<<"\t\tz="<<point3d[i].z;

}

initgraph(&gd,&gm,NULL);

setcolor(GREEN);

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

{

point2d[i].x = point3d[i].x;

point2d[i].y = point3d[i].y;

}

cout<<"\nAfter Projection:";

print(point2d,s);

setcolor(RED);

drawObject(point2d,s);

getch();

closegraph();

}

int main()

{

int i,n;

struct Point3D myp3d[100];

struct Point myPoint2D[100];

cout<<"Enter point count:";

cin>>n;

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

{

cout<<"cordinates of p"<<i<<" :";

cin>>myp3d[i].x>>myp3d[i].y>>myp3d[i].z;

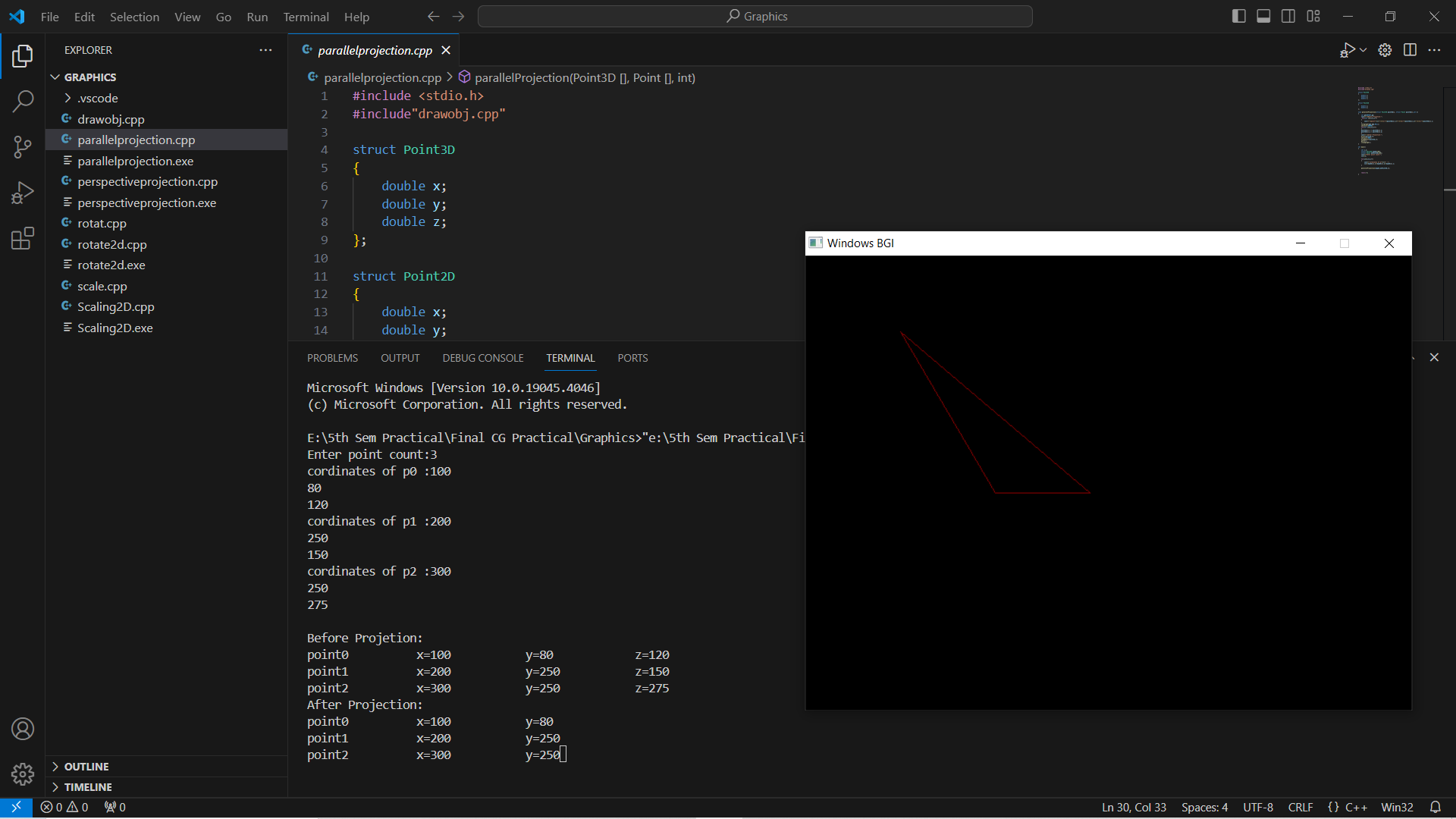
}

parallelProjection(myp3d,myPoint2D,n);

return 0;

}

**OUTPUT:**



**//File name: perspectiveprojection.cpp**

#include"drawobj.cpp"

#include<graphics.h>

struct Point3D

{

double x;

double y;

double z;

};

struct Point2D

{

double x;

double y;

};

void perspectiveProjection(struct Point3D point3d[], struct Point point2d[],int s,double f)

{

int gd=DETECT,gm;

cout<<"\nBefore Projetion:";

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

{

cout<<"\npoint"<<i<<"\t\tx="<<point3d[i].x<<"\t\ty="<<point3d[i].y<<"\t\tz="<<point3d[i].z;

}

initgraph(&gd,&gm,NULL);

setcolor(GREEN);

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

{

point2d[i].x = point3d[i].x\*f/point3d[i].z;

point2d[i].y = point3d[i].y\*f/point3d[i].z;

}

cout<<"\nAfter Projection:";

print(point2d,s);

setcolor(RED);

drawObject(point2d,s);

getch();

closegraph();

}

int main()

{

int i,n;

double f;

struct Point3D myp3d[100];

struct Point myp2d[100];

cout<<"points count:";

cin>>n;

cout<<"focus :";

cin>>f;

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

{

cout<<"cordinates of p"<<i<<" :";

cin>>myp3d[i].x>>myp3d[i].y>>myp3d[i].z;

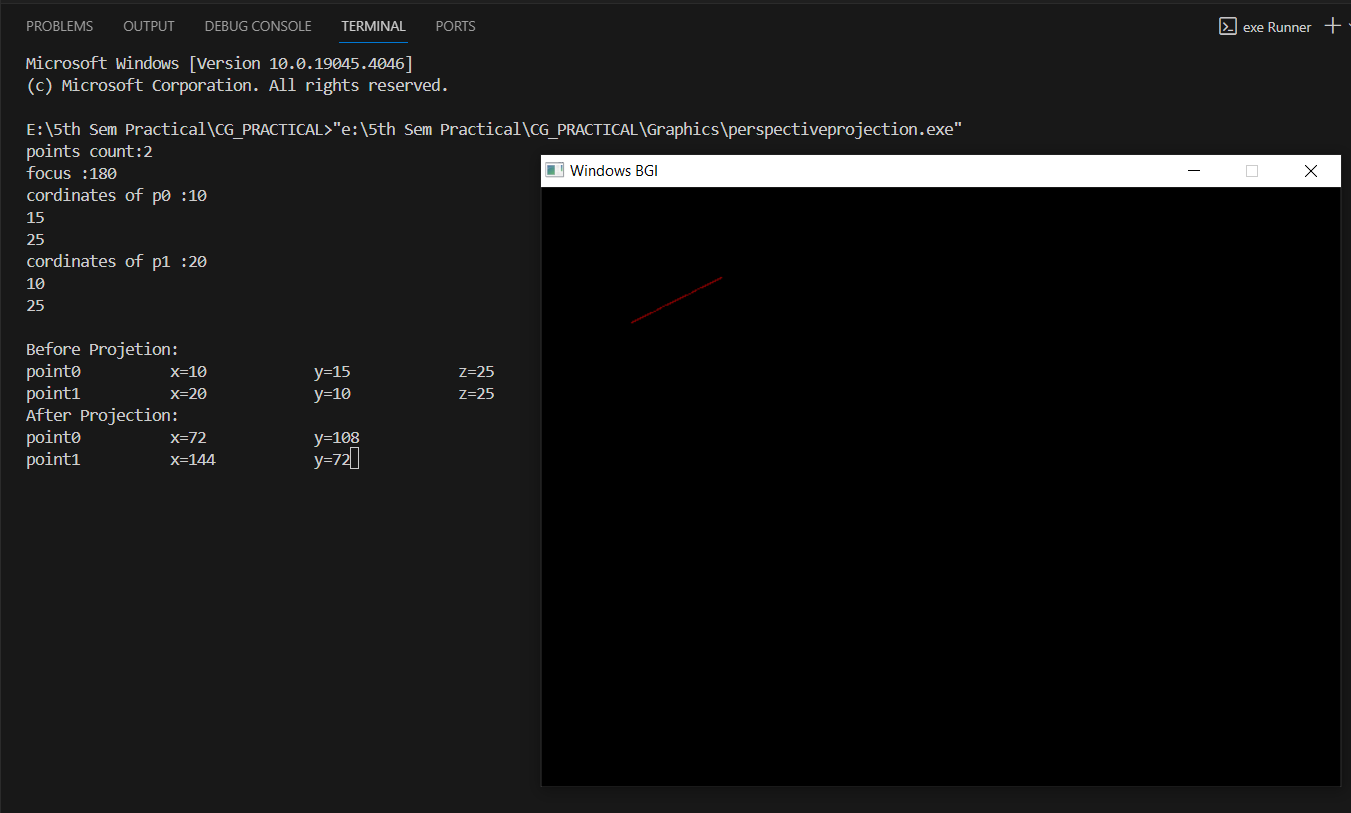
}

perspectiveProjection(myp3d,myp2d,n,f);

return 0;

}

**OUTPUT:**



1. **DRAW OBJECT**
2. #include<graphics.h>
3. #include<conio.h>
4. int main()
5. {
6. int gd=DETECT,gm;
7. initgraph (&gd,&gm,NULL);
8. //setbkcolor(GREEN);
9. printf("\t\t\t\n\nLINE");
10. line(50,40,190,40);
11. printf("\t\t\n\n\n\nRECTANGLE");
12. rectangle(120,110,215,165);
13. printf("\t\t\t\n\n\n\n\n\n\nARC");
14. arc(120,200,180,0,30);
15. printf("\t\n\n\n\nCIRCLE");
16. circle(120,270,30);
17. printf("\t\n\n\n\nECLIPSE");
18. ellipse(120,350,0,360,30,20);
19. getch();
20. closegraph();
21. return 0;
22. }

**OUTPUT:**

