



KALINDI COLLEGE
UNIVERSITY OF DELHI

NAME: PRACHI AGGARWAL

COURSE: BSC HONS COMPUTER SCIENCE

COLLEGE ROLL NUMBER: 21570015

UNIVERSITY ROLL NUMBER: 21033570042

SUBJECT: COMPUTER GRAPHICS

SUBMITTED TO: MS SHALINI SHARMA

INDEX

S.No.	Practical	Page No.
1.	Write a program to implement Bresenham's line drawing algorithm.	3
2.	Write a program to implement mid-point circle drawing algorithm.	6
3.	Write a program to clip a line using Cohen and Sutherland line clipping algorithm.	8
4.	Write a program to clip a polygon using Sutherland Hodgeman algorithm.	13
5.	Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates)	18
6.	Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.	35
7.	Write a program to draw Hermite /Bezier curve.	56
8.	Dda line algorithm	61
9.	Mid-point line algorithm	64

Question 1: Write a program to implement Bresenham's line drawing algorithm

Code:

```
#include<iostream>
#include<graphics.h>
#include<stdlib.h>
using namespace std;
void midPoint(int x1,int y1,int x2,int y2){
    int dx,dy,xe;
    dx=x2-x1;
    dy=y2-y1;
    int d= 2*dy - dx;
    int x = x1,y=y1;
    if(x1>x2){
        x=x2;
        y=y2;
        xe=x1;
    }
    else{
        x=x1;
        y=y1;
        xe=x2;
    }
    cout<<"("<<x<<','<<y<<")"<<"\n";
    while(x<xe){
        putpixel(x,y,7);
        x++;
        if(d<0){
            d= d+2*dy;
            putpixel(x,y,7);
```

```

    }
    else{
        d += 2*dy-2*dx;
        y++;
        putpixel(x,y,7);
    }
    cout<<"("<<x<<','<<y<<")"<<"\n";
}
}

int main(){
    system("cls");
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    int x1,x2,y1,y2;
    cout<<"Enter coordinates in sequence x1 y1 x2 y2"<<endl;
    cin>>x1>>y1>>x2>>y2;
    midPoint(x1,y1,x2,y2);
    getch();
    return 0;
}

```

Output:



Enter coordinates in sequence x1 y1 x2 y2

20

30

40

50

(20,30)

(21,31)

(22,32)

(23,33)

(24,34)

(25,35)

(26,36)

(27,37)

(28,38)

(29,39)

(30,40)

(31,41)

(32,42)

(33,43)

(34,44)

(35,45)

(36,46)

(37,47)

(38,48)

(39,49)

(40,50)

Question 2: Write a program to implement mid-point circle drawing algorithm.

Code:

```
#include<iostream>
#include<graphics.h>
using namespace std;
void drawcircle(int x0, int y0, int radius)
{
    int x = radius;
    int y = 0;
    int err = 0;
    while (x >= y)
    {
        putpixel(x0 + x, y0 + y, 7);
        putpixel(x0 + y, y0 + x, 7);
        putpixel(x0 - y, y0 + x, 7);
        putpixel(x0 - x, y0 + y, 7);
        putpixel(x0 - x, y0 - y, 7);
        putpixel(x0 - y, y0 - x, 7);
        putpixel(x0 + y, y0 - x, 7);
        putpixel(x0 + x, y0 - y, 7);
        if (err <= 0)
        {
            y += 1;
            err += 2*y + 1;
        }
        if (err > 0)
        {
            x -= 1;
            err -= 2*x + 1;
        }
    }
}
```

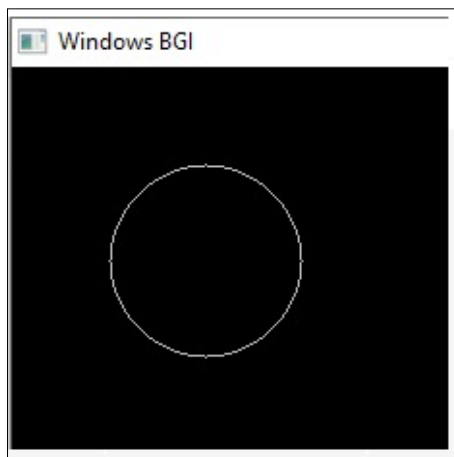
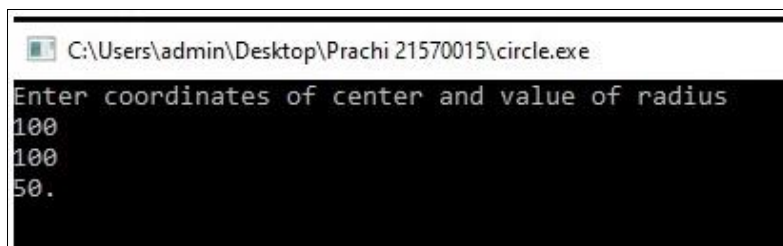
```

}
}
}

int main(){
    system("cls");
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    int x,y,r;
    cout<<"Enter coordinates of center and value of radius"<<endl;
    cin>>x>>y>>r;
    drawcircle(x,y,r);
    getch();
    return 0;
}

```

Output:



Question 3: Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

Code:

```
#include<graphics.h>
#include<iostream>
#include<math.h>
using namespace std;
const int FRAME = 0;
const int LEFT = 1 ;
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;
const int xmin = 40;
const int xmax = 100;
const int ymin = 40;
const int ymax = 80;
int coordinateCode(double x, double y){
    int code = FRAME;
    if(x<xmin){
        code = LEFT;
    }
    else if(x>xmax){
        code = RIGHT;
    }
    if(y<ymin){
        code = BOTTOM;
    }
    else if(y>ymax){
        code = TOP;
    }
}
```



```

        return code;
    }

void cohenSutherlandLineClipping(double x1,double y1, double x2, double y2){
    int code1 = coordinateCode(x1,y1);
    int code2 = coordinateCode(x2,y2);
    bool valid = false;
    line(xmin+200,ymax+200,xmax+200,ymax+200);
    line(xmax+200,ymax+200,xmax+200,ymin+200);
    line(xmax+200,ymin+200,xmin+200,ymin+200);
    line(xmin+200,ymin+200,xmin+200,ymax+200);
    while(true){
        if((code1==0)&&(code2==0)){
            valid = true;
            break;
        }
        else if(code1 & code2){
            break;
        }
        else{
            int code_out;
            double a,b;

            if(code1 != 0){
                code_out = code1;
            }
            else{
                code_out = code2;
            }

            if(code_out & TOP){

```

```

        a = x1 + (x2-x1)*(ymax-y1)/(y2-y1);
        b = ymax;
    }
    else if(code_out & BOTTOM){
        a = x1 + (x2-x1)*(ymin-y1)/(y2-y1);
        b = ymin;
    }
    else if(code_out & RIGHT){
        b = y1+(y2-y1)*(xmax-x1)/(x2-x1);
        a = xmax;
    }
    else if(code_out & LEFT){
        b = y1 + (y2-y1)*(xmin-x1)/(x2-x1);
        a = xmin;
    }

    if(code_out == code1){
        x1=a;
        y1=b;
        code1= coordinateCode(x1,y1);
    }
    else{
        x2=a;
        y2=b;
        code2 = coordinateCode(x2,y2);
    }
}
}
if(valid){

```

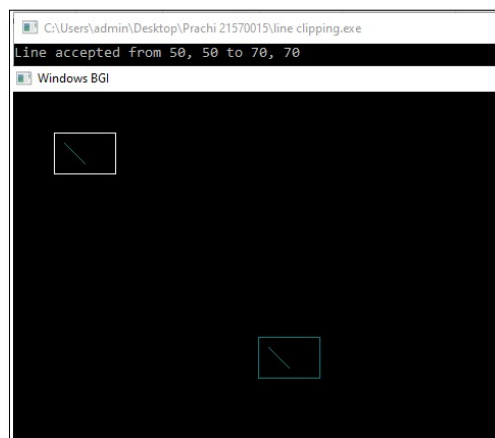
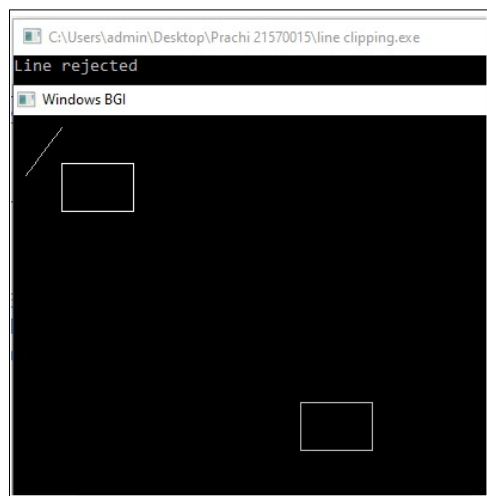
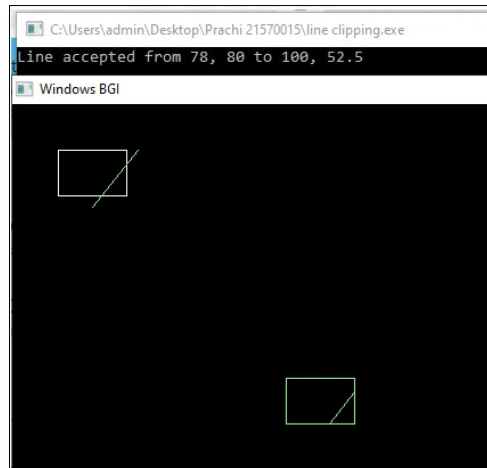
```

cout << "Line accepted from " << x1 << ", "<< y1 << " to " << x2 << ", " << y2 << endl;
        line(x1+200,y1+200,x2+200,y2+200);
    }
    else
        cout << "Line rejected" << endl;
}

int main(){
    system("cls");
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    line(xmin,ymax,xmax,ymax);
line(xmax,ymax,xmax,ymin);
line(xmax,ymin,xmin,ymin);
line(xmin,ymin,xmin,ymax);
    setcolor(3);
    line(50,50,70,70);
    setcolor(7);
    line(10,50,40,10);
    setcolor(10);
    line(70,90,110,40);
    cohenSutherlandLineClipping(50,50,70,70);
    cohenSutherlandLineClipping(10,50,40,10);
    cohenSutherlandLineClipping(70,90,110,40);
    getch();
    return 0;
}

```

Output:



Question 4: Write a program to clip a polygon using Sutherland Hodgeman algorithm.

Code:

```
#include<graphics.h>
#include<iostream>
#include<math.h>
using namespace std;
const int FRAME = 0;
const int LEFT = 1 ;
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;
const int xmin = 60;
const int xmax = 120;
const int ymin = 60;
const int ymax = 100;
int coordinateCode(double x, double y){
    int code = FRAME;
    if(x<xmin){
        code = LEFT;
    }
    else if(x>xmax){
        code = RIGHT;
    }
    if(y<ymin){
        code = BOTTOM;
    }
    else if(y>ymax){
        code = TOP;
    }
    return code;
}
```

```

}

void cohenSutherlandLineClipping(double x1,double y1, double x2, double y2){
    int code1 = coordinateCode(x1,y1);
    int code2 = coordinateCode(x2,y2);
    bool valid = false;
    line(xmin+200,ymax+200,xmax+200,ymax+200);
    line(xmax+200,ymax+200,xmax+200,ymin+200);
    line(xmax+200,ymin+200,xmin+200,ymin+200);
    line(xmin+200,ymin+200,xmin+200,ymax+200);
    while(true){
        if((code1==0)&&(code2==0)){
            valid = true;
            break;
        }
        else if(code1 & code2){
            break;
        }
        else{
            int code_out;
            double a,b;
            if(code1 != 0){
                code_out = code1;
            }
            else{
                code_out = code2;
            }
            if(code_out & TOP){
                a = x1 + (x2-x1)*(ymax-y1)/(y2-y1);
                b = ymax;
            }
        }
    }
}

```

```

else if(code_out & BOTTOM){
    a = x1 + (x2-x1)*(ymin-y1)/(y2-y1);
    b = ymin;
}
else if(code_out & RIGHT){
    b = y1+(y2-y1)*(xmax-x1)/(x2-x1);
    a = xmax;
}
else if(code_out & LEFT){
    b = y1 + (y2-y1)*(xmin-x1)/(x2-x1);
    a = xmin;
}
if(code_out == code1){
    x1=a;
    y1=b;
    code1= coordinateCode(x1,y1);
}
else{
    x2=a;
    y2=b;
    code2 = coordinateCode(x2,y2);
}
}
}
if(valid){
    cout << "Line accepted from " << x1 << ", " << y1 << " to " << x2 << ", " << y2 << endl;
    line(x1+200,y1+200,x2+200,y2+200);
}
else
    cout << "Line rejected" << endl;

```

```

}

void polygonclipping(int n){
    int arr[n][2][2];
    cout<<"Enter coordinates of your polygon: "<<endl;
    for(int i=0; i<n; i++){
        cout<<"Edge"<<i<<endl;
        for(int j=0; j<2; j++){
            cout<<"Endpoint"<<j<<":";
            for(int k=0; k<2; k++){
                cin>>arr[i][j][k];
            }
        }
        line(arr[i][0][0],arr[i][0][1],arr[i][1][0],arr[i][1][1]);
        cout<<endl;
    }
    for(int i=0; i<n; i++){
        cohenSutherlandLineClipping(arr[i][0][0],arr[i][0][1],arr[i][1][0],arr[i][1][1]);
    }
}

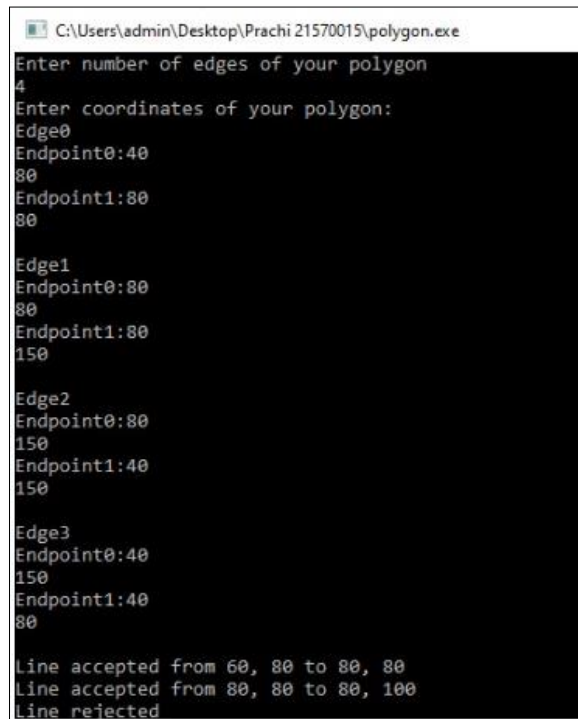
int main(){
    system("cls");
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    line(xmin,ymax,xmax,ymax);
    line(xmax,ymax,xmax,ymin);
    line(xmax,ymin,xmin,ymin);
    line(xmin,ymin,xmin,ymax);
    int n;
    cout<<"Enter number of edges of your polygon"<<endl;
    cin>>n;

```

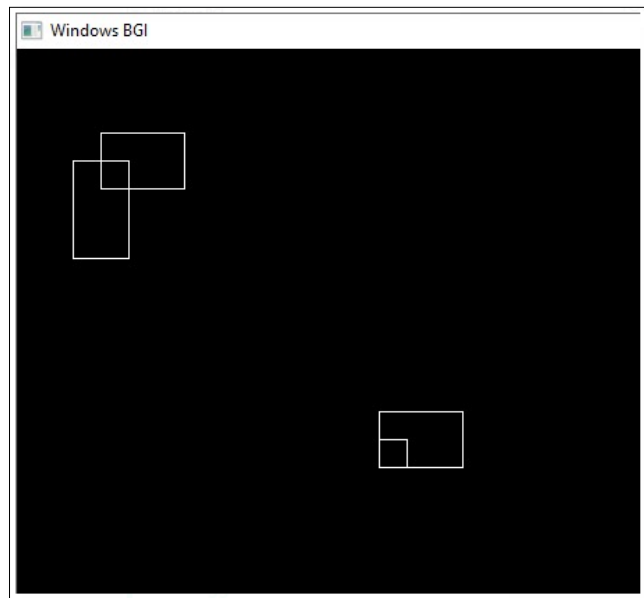


```
    polygonclipping(n);  
    getch();  
    return 0;  
}
```

Output:



```
C:\Users\admin\Desktop\Prachi 21570015\polygon.exe  
Enter number of edges of your polygon  
4  
Enter coordinates of your polygon:  
Edge0  
Endpoint0:40  
80  
Endpoint1:80  
80  
  
Edge1  
Endpoint0:80  
80  
Endpoint1:80  
150  
  
Edge2  
Endpoint0:80  
150  
Endpoint1:40  
150  
  
Edge3  
Endpoint0:40  
150  
Endpoint1:40  
80  
  
Line accepted from 60, 80 to 80, 80  
Line accepted from 80, 80 to 80, 100  
Line rejected
```



Question 5: Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates)

Code:

```
#include<iostream>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#define pi 3.14285714
using namespace std;

class transformations
{
double vertices[3][3];
double t_matrix[3][3];
double result[3][3];

public:
transformations(){ };

void get_vertices();
void display_triangle();
void display_triangle_result();

void multiplication();
void copyback();

void rotation(double angle,double m,double n);
void reflection(double m,double c);
void scaling(double a,double d);
```

```

void shearing(double b,double c);
void translation(double tx, double ty);
};

void transformations::get_vertices()
{
int i=0;

for(i=0;i<3;i++)
{
    cout<<"\nEnter vertex "<<i+1<<":";
    cout<<"\nx1 : ";
    cin>>vertices[i][0];
    result[i][0]=vertices[i][0];

    cout<<"y1 : ";
    cin>>vertices[i][1];
    result[i][1]=vertices[i][1];

    vertices[i][2]=result[i][2]=1;
}
}

void transformations::display_triangle()
{
int i=0;
for(i=0;i<2;i++)
    line(vertices[i][0],vertices[i][1],vertices[i+1][0],vertices[i+1][1]);
    line(vertices[i][0],vertices[i][1],vertices[0][0],vertices[0][1]);
}

```

```

void transformations::display_triangle_result()
{
int i=0;
for(i=0;i<2;i++)
    line(result[i][0],result[i][1],result[i+1][0],result[i+1][1]);
    line(result[i][0],result[i][1],result[0][0],result[0][1]);
}

```

```

void transformations::copyback()
{
int i=0,j=0;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
result[i][j]=vertices[i][j];
}

```

```

void transformations::multiplication()
{
double r[3][3];
int i=0,j=0,k=0;

for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
r[i][j]=0;
for(k=0;k<3;k++)
r[i][j]+=result[i][k]*t_matrix[k][j];
}
}
}

```

```

    }
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    result[i][j]=r[i][j];
}

void transformations::translation(double tx, double ty){
    copyback();
    cleardevice();

    setcolor(RED);
    display_triangle();
    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[2][0]=tx;
    t_matrix[2][1]=ty;
    t_matrix[2][2]=1;

    multiplication();
    setcolor(YELLOW);
    display_triangle_result();
    getch();
}

void transformations::rotation(double angle,double m,double n)
{

```

```
angle=((pi/180)*angle);
```

```
copyback();
```

```
cleardevice();
```

```
setcolor(RED);
```

```
display_triangle();
```

```
t_matrix[0][0]=1;
```

```
t_matrix[0][1]=0;
```

```
t_matrix[0][2]=0;
```

```
t_matrix[1][0]=0;
```

```
t_matrix[1][1]=1;
```

```
t_matrix[1][2]=0;
```

```
t_matrix[2][0]=(m*(-1));
```

```
t_matrix[2][1]=(n*(-1));
```

```
t_matrix[2][2]=1;
```

```
multiplication();
```

```
t_matrix[0][0]=cos(angle);
```

```
t_matrix[0][1]=sin(angle);
```

```
t_matrix[1][0]=(sin(angle)*(-1));;
```

```
t_matrix[1][1]=cos(angle);
```

```
t_matrix[2][0]=0;
```

```
t_matrix[2][1]=0;
```

```
multiplication();
```

```
t_matrix[0][0]=1;
```

```
t_matrix[0][1]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=1;
t_matrix[2][0]=m;
t_matrix[2][1]=n;
    multiplication();
```

```
setcolor(YELLOW);
display_triangle_result();
getch();
}
```

```
void transformations::reflection(double m,double c)
{
double angle=atan(m);
copyback();
cleardevice();
```

```
double x1=0,y1=c,x2=400,y2=(m*x2)+c;
setcolor(YELLOW);
line(x1,y1,x2,y2);
```

```
setcolor(RED);
display_triangle();
```

```
t_matrix[0][0]=1;
t_matrix[0][1]=0;
t_matrix[0][2]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=1;
```

```
t_matrix[1][2]=0;
t_matrix[2][0]=0;
t_matrix[2][1]=(c*(-1));
t_matrix[2][2]=1;
```

```
multiplication();
```

```
t_matrix[0][0]=cos(-1*angle);
t_matrix[0][1]=sin(-1*angle);
t_matrix[1][0]=(sin(-1*angle)*(-1));;
t_matrix[1][1]=cos(-1*angle);
t_matrix[2][0]=0;
t_matrix[2][1]=0;
```

```
multiplication();
```

```
t_matrix[0][0]=1;
t_matrix[0][1]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=-1;
t_matrix[2][0]=0;
t_matrix[2][1]=0;
```

```
multiplication();
```

```
t_matrix[0][0]=cos(angle);
t_matrix[0][1]=sin(angle);
t_matrix[1][0]=(sin(angle)*(-1));;
t_matrix[1][1]=cos(angle);
t_matrix[2][0]=0;
```



```
t_matrix[2][1]=0;
```

```
    multiplication();
```

```
t_matrix[0][0]=1;
```

```
t_matrix[0][1]=0;
```

```
t_matrix[1][0]=0;
```

```
t_matrix[1][1]=1;
```

```
t_matrix[2][0]=0;
```

```
t_matrix[2][1]=c;
```

```
    multiplication();
```

```
setcolor(YELLOW);
```

```
display_triangle_result();
```

```
getch();
```

```
}
```

```
void transformations::scaling(double a,double d)
```

```
{
```

```
copyback();
```

```
cleardevice();
```

```
setcolor(RED);
```

```
display_triangle();
```

```
t_matrix[0][0]=a;
```

```
t_matrix[0][1]=0;
```

```
t_matrix[0][2]=0;
```

```
t_matrix[1][0]=0;
```

```
t_matrix[1][1]=d;
```

```
t_matrix[1][2]=0;
t_matrix[2][0]=0;
t_matrix[2][1]=0;
t_matrix[2][2]=1;
```

```
    multiplication();
```

```
setcolor(YELLOW);
display_triangle_result();
getch();
}
```

```
void transformations::shearing(double b,double c)
```

```
{
copyback();
cleardevice();
```

```
setcolor(RED);
display_triangle();
```

```
t_matrix[0][0]=1;
t_matrix[0][1]=b;
t_matrix[0][2]=0;
t_matrix[1][0]=c;
t_matrix[1][1]=1;
t_matrix[1][2]=0;
t_matrix[2][0]=0;
t_matrix[2][1]=0;
t_matrix[2][2]=1;
```

```

        multiplication();

setcolor(YELLOW);
display_triangle_result();
getch();
}

int main()
{
system("cls");
int gd=DETECT,gm;
int choice;
transformations t1;
char ch1,ch2;
double angle,m,n,slope,intercept,a,b,c,d,tx,ty;

do
{
cout<<"\n\n\tTWO DIMENSIONAL TRANSFORMATIONS\n";
cout<<"\nEnter the details of a triangle(i.e. 2-D object)";
    t1.get_vertices();

    do
    {
        initgraph(&gd,&gm,(char*)"");
        cout<<"\n.....MENU.....";
        cout<<"\n1.Rotation.";
        cout<<"\n2.Reflection.";
        cout<<"\n3.Scaling.";
        cout<<"\n4.Shearing.";
        cout<<"\n5.Translation.";
    }
}

```

```

cout<<"\n.....";
cout<<"\n\nEnter your choice :: ";
cin>>choice;
switch(choice)
{
    case 1:cout<<"\n\nFOR ROTATION.....";
            cout<<"\nEnter the angle of rotation :: ";
            cin>>angle;

            cout<<"\nEnter the point about which rotation is performed :: ";
            cout<<"\nx coordinate : ";
            cin>>m;
            cout<<"y coordinate : ";
            cin>>n;

            t1.rotation(angle,m,n);
            break;

    case 2:cout<<"\n\nFOR REFLECTION.....";
            cout<<"\nTo enter the line in slope-intercept form(i.e.  $y=mx+b$ )....";
            cout<<"\nEnter slope(m) : ";
            cin>>slope;
            cout<<"Enter y-intercept(b) : ";
            cin>>intercept;

            t1.reflection(slope,intercept);
            break;

    case 3:cout<<"\n\nFOR SCALING.....";
            cout<<"\nEnter the factor of scaling...";

```

```

        cout<<"\nAlong the x-axis : ";
        cin>>a;
        cout<<"Along the y-axis : ";
        cin>>d;

        t1.scaling(a,d);
        break;

    case 4:cout<<"\n\nFOR SHEARING.....";
        cout<<"\nEnter the factor of shearing...";
        cout<<"\nAlong the x-axis : ";
        cin>>c;
        cout<<"Along the y-axis : ";
        cin>>b;

        t1.shearing(b,c);
        break;

    case 5: cout<<"\n\nFOR TRANSLATION.....";
        cout<<"\nEnter the factor of shearing...";
        cout<<"\nAlong the x-axis : ";
        cin>>tx;
        cout<<"Along the y-axis : ";
        cin>>ty;
        t1.translation(tx,ty);
        break;

    default:cout<<"\n\n\t!!!INVALID CHOICE!!!";
        getch();
}

```

```

closegraph();

cout<<"\nDo you want to try another transformation?(Y/N)...";
cin>>ch2;

}while(ch2=='y' || ch2=='Y');

cout<<"\n\nDo you want to try with a triangle with different dimensions(Y/N)? ";
cin>>ch1;
}while(ch1=='y' || ch1=='Y');
getch();
return 0;
}

```

Output:

```

                TWO DIMENSIONAL TRANSFORMATIONS

Enter the details of a triangle(i.e. 2-D object)
Enter vertex 1:
x1 : 100
y1 : 100

Enter vertex 2:
x1 : 140
y1 : 50

Enter vertex 3:
x1 : 180
y1 : 100

.....MENU.....
1.Rotation.
2.Reflection.
3.Scaling.
4.Shearing.
5.Translation.
.....

```

- Rotation

```
Enter your choice :: 1

FOR ROTATION.....
Enter the angle of rotation :: 60

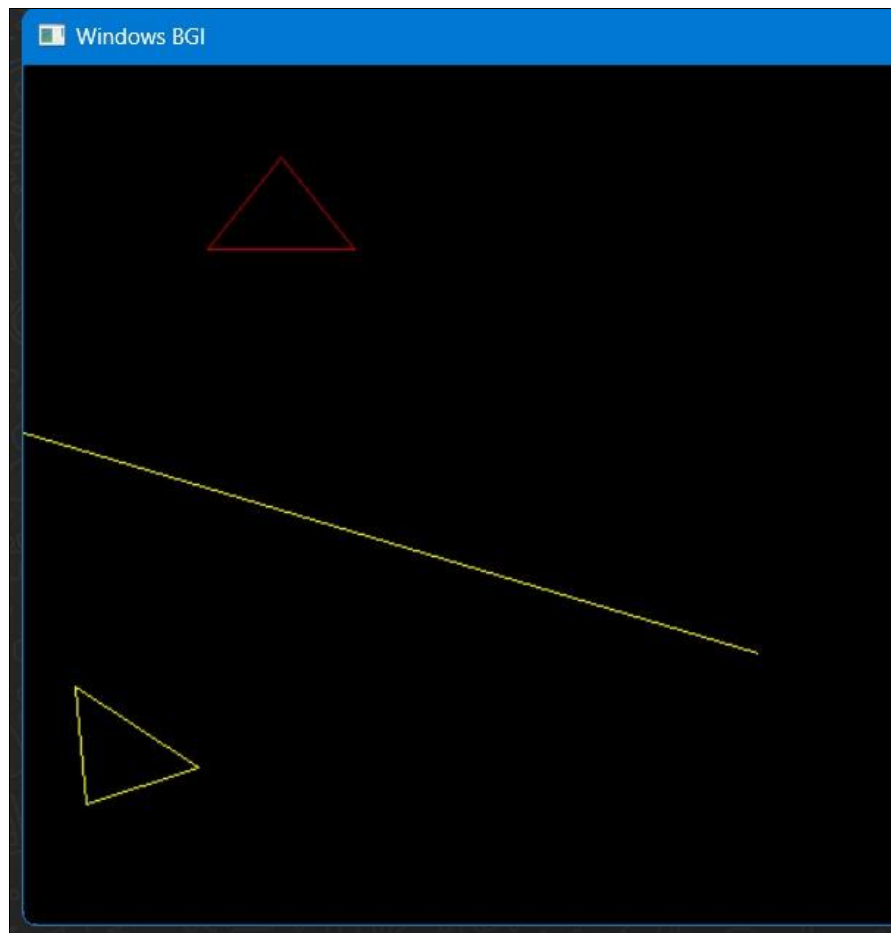
Enter the point about which rotation is performed ::
x coordinate : 180
y coordinate : 100
|
```



- Reflection

```
Enter your choice :: 2

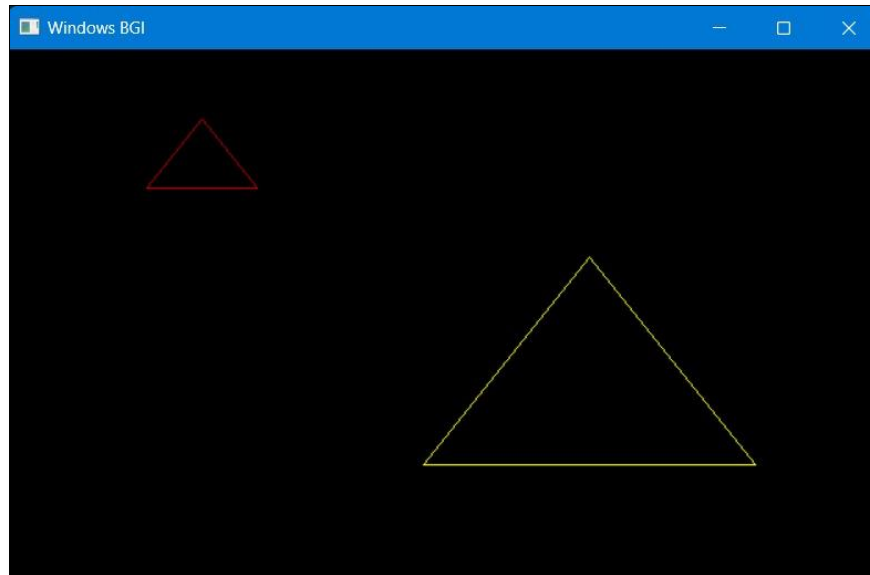
FOR REFLECTION.....
To enter the line in slope-intercept form(i.e.  $y=mx+b$ )....
Enter slope(m) : 0.3
Enter y-intercept(b) : 200
|
```



- Scaling

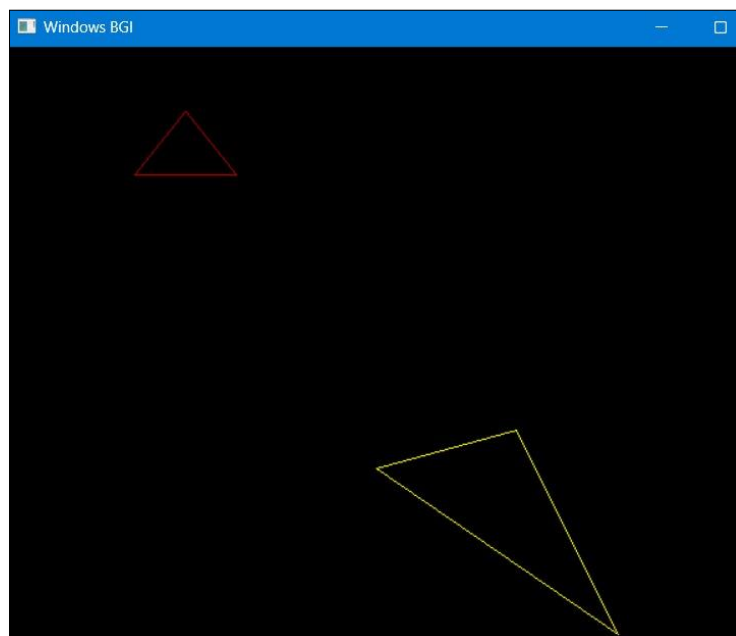
```
Enter your choice :: 3

FOR SCALING.....
Enter the factor of scaling...
Along the x-axis : 3
Along the y-axis : 3
|
```

- Shearing

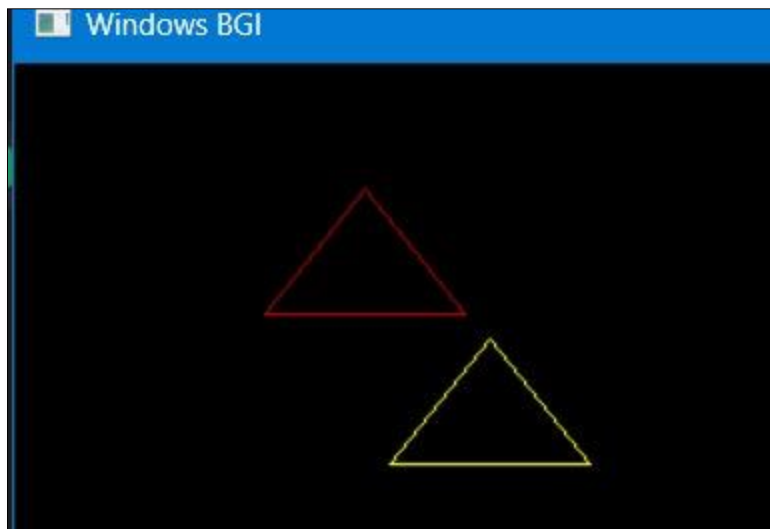
```
Enter your choice :: 4  
  
FOR SHEARING.....  
Enter the factor of shearing...  
Along the x-axis : 3  
Along the y-axis : 2  
|
```



- Translation

```
Enter your choice :: 5
```

```
FOR TRANSLATION.....  
Enter the factor of shearing...  
Along the x-axis : 50  
Along the y-axis : 60  
|
```



Question 6: Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

Code:

```
#include<iostream>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#define pi 3.14285714
using namespace std;
class projections
{
    double vertices[8][4];
    double t_matrix[4][4];
    double result[8][4];

    public:
    projections(){ };

    void get_vertices();
    void display_cube();

    void multiplication();
    void copyback();

    void orthographic();
    void axonometric(double angle_x,double angle_y);
    void cavalier(double angle);
    void cabinet(double angle);
    void single_point(double r);
```

```

    void two_point(double r);
    void three_point(double r);
};

void projections::get_vertices()
{
    for(int i=0;i<8;i++)
    {
        cout<<"\nEnter vertex "<<i+1<<":";
        cout<<"\nx : ";
        cin>>vertices[i][0];
        result[i][0]=vertices[i][0];

        cout<<"y : ";
        cin>>vertices[i][1];
        result[i][1]=vertices[i][1];

        cout<<"z : ";
        cin>>vertices[i][2];
        result[i][2]=vertices[i][2];

        vertices[i][3]=result[i][3]=1;
    }
}

void projections::display_cube()
{
    int i=0;

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

```

```

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

for(i=4;i<7;i++)
line(result[i][0],result[i][1],result[i+1][0],result[i+1][1]);
line(result[i][0],result[i][1],result[4][0],result[4][1]);

for(i=0;i<4;i++)
line(result[i][0],result[i][1],result[i+4][0],result[i+4][1]);
}

```

```

void projections::copyback()

```

```

{
    int i=0,j=0;

    for(i=0;i<8;i++)
    for(j=0;j<4;j++)
    result[i][j]=vertices[i][j];
}

```

```

void projections::multiplication()

```

```

{
    double r[8][4];
    int i=0,j=0,k=0;

    for(i=0;i<8;i++)
    {
        for(j=0;j<4;j++)
        {
            r[i][j]=0;

```

```

        for(k=0;k<4;k++)
            r[i][j]+=result[i][k]*t_matrix[k][j];
    }
}

for(i=0;i<8;i++)
for(j=0;j<4;j++)
result[i][j]=r[i][j];
}

void projections::orthographic()
{
    cleardevice();
    clearviewport();
    cout<<"\tORTHOGRAPHIC PROJECTION";

    copyback();

    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=0;
    t_matrix[2][1]=0;

```

```

t_matrix[2][2]=0;
t_matrix[2][3]=0;

t_matrix[3][0]=0;
t_matrix[3][1]=0;
t_matrix[3][2]=0;
t_matrix[3][3]=1;

multiplication();

setcolor(YELLOW);
display_cube();
delay(20);
getch();
}

void projections::axonometric(double angle_x,double angle_y)
{
    angle_x=((pi/180)*angle_x);
    angle_y=((pi/180)*angle_y);

    cleardevice();
    clearviewport();
    cout<<"\tAXONOMETRIC PROJECTION";

    copyback();

    t_matrix[0][0]=cos(angle_y);
    t_matrix[0][1]=(sin(angle_y))*(sin(angle_x));
    t_matrix[0][2]=0;

```

```
t_matrix[0][3]=0;
```

```
t_matrix[1][0]=0;
```

```
t_matrix[1][1]=cos(angle_x);
```

```
t_matrix[1][2]=0;
```

```
t_matrix[1][3]=0;
```

```
t_matrix[2][0]=sin(angle_y);
```

```
t_matrix[2][1]=(-1)*(cos(angle_y)*sin(angle_x));
```

```
t_matrix[2][2]=0;
```

```
t_matrix[2][3]=0;
```

```
t_matrix[3][0]=0;
```

```
t_matrix[3][1]=0;
```

```
t_matrix[3][2]=0;
```

```
t_matrix[3][3]=1;
```

```
    multiplication();
```

```
    setcolor(YELLOW);
```

```
    display_cube();
```

```
    delay(20);
```

```
    getch();
```

```
}
```

```
void projections::cavalier(double angle)
```

```
{
```

```
    angle=((pi/180)*angle);
```

```
    cleardevice();
```



```
clearviewport();  
cout<<"\tCAVALIER PROJECTION";  
  
copyback();  
  
t_matrix[0][0]=1;  
t_matrix[0][1]=0;  
t_matrix[0][2]=0;  
t_matrix[0][3]=0;  
  
t_matrix[1][0]=0;  
t_matrix[1][1]=1;  
t_matrix[1][2]=0;  
t_matrix[1][3]=0;  
  
t_matrix[2][0]=(-1)*1*cos(angle);  
t_matrix[2][1]=(-1)*1*sin(angle);  
t_matrix[2][2]=0;  
t_matrix[2][3]=0;  
  
t_matrix[3][0]=0;  
t_matrix[3][1]=0;  
t_matrix[3][2]=0;  
t_matrix[3][3]=1;  
  
multiplication();  
  
setcolor(YELLOW);  
display_cube();  
delay(20);
```

```

    getch();
}

void projections::cabinet(double angle)
{
    angle=((pi/180)*angle);

    cleardevice();
    clearviewport();
    cout<<"\tCABINET PROJECTION";

    copyback();

    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=(-1)*0.5*cos(angle);
    t_matrix[2][1]=(-1)*0.5*sin(angle);
    t_matrix[2][2]=0;
    t_matrix[2][3]=0;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;

```

```

t_matrix[3][2]=0;
t_matrix[3][3]=1;

multiplication();

setcolor(YELLOW);
display_cube();
delay(20);
getch();
}

void projections::single_point(double r)
{
    double l=10,m=10,n=10;

    cleardevice();
    clearviewport();
    cout<<"\tSINGLE POINT PRESPECTIVE PROJECTION";

    r=(-1/r);
    copyback();

    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;

```

```

t_matrix[1][3]=0;

t_matrix[2][0]=0;
t_matrix[2][1]=0;
t_matrix[2][2]=0;
t_matrix[2][3]=r;

t_matrix[3][0]=1;
t_matrix[3][1]=m;
t_matrix[3][2]=0;
t_matrix[3][3]=r*n+1;

multiplication();

setcolor(YELLOW);
display_cube();
delay(20);
getch();
}

void projections::two_point(double r)
{
    double angle=45;
    angle=(pi/180)*angle;

    cleardevice();
    clearviewport();
    cout<<"\tTWO POINT PRESPECTIVE PROJECTION";

    copyback();

```

```

r=(-1/r);

t_matrix[0][0]=cos(angle);
t_matrix[0][1]=0;
t_matrix[0][2]=(-1*sin(angle));
t_matrix[0][3]=sin(angle)/r;

t_matrix[1][0]=0;
t_matrix[1][1]=1;
t_matrix[1][2]=0;
t_matrix[1][3]=0;

t_matrix[2][0]=sin(angle);
t_matrix[2][1]=0;
t_matrix[2][2]=cos(angle);
t_matrix[2][3]=(-1*cos(angle))/r;

t_matrix[3][0]=0;
t_matrix[3][1]=0;
t_matrix[3][2]=0;
t_matrix[3][3]=1;

multiplication();

setcolor(YELLOW);
display_cube();
delay(20);
getch();
}

```

```

void projections::three_point(double r)
{
    double angle_y=45,angle_x=45;
    angle_y=(pi/180)*angle_y;
    angle_x=(pi/180)*angle_x;

    cleardevice();
    clearviewport();
    cout<<"\tTHREE POINT PRESPECTIVE PROJECTION";

    copyback();

    r=(-1/r);

    t_matrix[0][0]=cos(angle_y);
    t_matrix[0][1]=sin(angle_y)*sin(angle_x);
    t_matrix[0][2]=0;
    t_matrix[0][3]=(sin(angle_y)*cos(angle_x))/r;

    t_matrix[1][0]=0;
    t_matrix[1][1]=cos(angle_x);
    t_matrix[1][2]=0;
    t_matrix[1][3]=(-1*sin(angle_x))/r;

    t_matrix[2][0]=(sin(angle_y));
    t_matrix[2][1]=(-1*cos(angle_y)*sin(angle_x));
    t_matrix[2][2]=0;
    t_matrix[2][3]=(-1*cos(angle_y)*cos(angle_x))/r;

```

```

t_matrix[3][0]=0;
t_matrix[3][1]=0;
t_matrix[3][2]=0;
t_matrix[3][3]=1;

multiplication();

setcolor(YELLOW);
display_cube();
delay(20);
getch();
}

int main()
{
    system("cls");
    int gd=DETECT,gm,choice;
    projections t1;
    char ch1,ch2,axis,axis1,axis2;
    double angle_x,angle_y,angle,ratio,ratio1,ratio2,ratio3;

    do
    {
        cout<<"\n\n\t.....PROJECTIONS OF 3D OBJECTS.....\n";
        cout<<"\nEnter the details of a cube(i.e. 3D object)";
        t1.get_vertices();

        do
        {
            initgraph(&gd,&gm,(char*)"");

```

```

cout<<"\n\n.....MENU.....";
cout<<"\n1.Orthographic.";
cout<<"\n2.Axonometric.";
cout<<"\n3.Cavalier (Oblique type 1).";
cout<<"\n4.Cabinet (Oblique type 2)";
cout<<"\n5.Single-Point presepective.";
cout<<"\n6.Two-Point presepective.";
cout<<"\n7.Three-Point presepective.";
cout<<"\n.....";

cout<<"\n\nEnter your choice :: ";
cin>>choice;

switch(choice)
{
    case 1:t1.orthographic();
        break;

    case 2:cout<<"\n\nFOR AXONOMETRIC PROJECTION";
        cout<<"\n\nEnter the angle of rotation about :";
        cout<<"\nx-axis : ";
        cin>>angle_x;
        cout<<"y-axis : ";
        cin>>angle_y;

        t1.axonometric(angle_x,angle_y);
        break;

    case 3:cout<<"\n\nFOR CAVALIER PROJECTION";
        cout<<"\n\nEnter the angle of inclination : ";

```



```

        cin>>angle;

        t1.cavalier(angle);
        break;

    case 4:cout<<"\n\nFOR CABINET PROJECTION";
        cout<<"\nEnter the angle of inclination : ";
        cin>>angle;

        t1.cabinet(angle);
        break;

    case 5:cout<<"\n\nFOR SINGLE POINT PRESPECTIVE PROJECTION";
        cout<<"\nAssuming that the VP lies on the z-axis, Enter the prespective
ratio: ";

        cin>>ratio;

        t1.single_point(ratio);

        break;

    case 6:cout<<"\n\nFOR TWO POINT PRESPECTIVE PROJECTION";
        cout<<"\nAssuming that the VP lies on the z-axis, Enter the prespective
ratio: ";

        cin>>ratio;

        t1.two_point(ratio);

        break;

```

```

        case 7:cout<<"\n\nFOR THREE POINT PRESPECTIVE PROJECTION";
                cout<<"\nAssuming that the VP lies on the z-axis, Enter the prespective
ratio: ";

                cin>>ratio;

                t1.three_point(ratio);

                break;

        default:cout<<"\n\n\t!!!INVALID CHOICE!!!";
                getch();
    }
    closegraph();

    cout<<"\nDo you want to try another projection(Y/N)?";
    cin>>ch2;

    } while(ch2=='y' || ch2=='Y');

    cout<<"\nDo you want to try a cube with different dimensions(Y/N)?";
    cin>>ch1;

    } while(ch1=='y' || ch1=='Y');
    getch();
    return 0;
}

```

Output:

```
.....PROJECTIONS OF 3D OBJECTS.....
```

```
Enter the details of a cube(i.e. 3D object)
```

```
Enter vertex 1:
```

```
x : 50
```

```
y : 50
```

```
z : 100
```

```
Enter vertex 2:
```

```
x : 50
```

```
y : 100
```

```
z : 100
```

```
Enter vertex 3:
```

```
x : 100
```

```
y : 100
```

```
z : 100
```

```
Enter vertex 4:
```

```
x : 100
```

```
y : 50
```

```
z : 100_
```

```
Enter vertex 4:
```

```
x : 100
```

```
y : 50
```

```
z : 100
```

```
Enter vertex 5:
```

```
x : 70
```

```
y : 70
```

```
z : 100
```

```
Enter vertex 6:
```

```
x : 70
```

```
y : 120
```

```
z : 100
```

```
Enter vertex 7:
```

```
x : 120
```

```
y : 120
```

```
z : 100
```

```
Enter vertex 8:
```

```
x : 120
```

```
y : 70
```

```
z : 100_
```



ORTHOGRAPHIC PROJECTION

```

.....MENU.....
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
.....

Enter your choice :: 2

FOR AXONOMETRIC PROJECTION
Enter the angle of rotation about :
x-axis : 45
y-axis : 120

```



AXONOMETRIC PROJECTION

```

.....MENU.....
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
.....

Enter your choice :: 3

FOR CAVALIER PROJECTION
Enter the angle of inclination : 200

```



CAVALIER PROJECTION

```
.....MENU.....  
1.Orthographic.  
2.Axonometric.  
3.Cavalier (Oblique type 1).  
4.Cabinet (Oblique type 2)  
5.Single-Point presepective.  
6.Two-Point presepective.  
7.Three-Point presepective.  
.....
```

Enter your choice :: 4

FOR CABINET PROJECTION

Enter the angle of inclination : 50



CABINET PROJECTION

```

.....MENU.....
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
.....

```

Enter your choice :: 5

FOR SINGLE POINT PRESPECTIVE PROJECTION

Assuming that the VP lies on the z-axis, Enter the prespective ratio: 0.9



SINGLE POINT PRESPECTIVE PROJECTION

```

.....MENU.....
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
.....

```

Enter your choice :: 6

FOR TWO POINT PRESPECTIVE PROJECTION

Assuming that the VP lies on the z-axis, Enter the prespective ratio: 0.5



TWO POINT PRESPECTIVE PROJECTION

```
.....MENU.....  
1.Orthographic.  
2.Axonometric.  
3.Cavalier (Oblique type 1).  
4.Cabinet (Oblique type 2)  
5.Single-Point presepective.  
6.Two-Point presepective.  
7.Three-Point presepective.  
.....
```

Enter your choice :: 7

FOR THREE POINT PRESPECTIVE PROJECTION

Assuming that the UP lies on the z-axis, Enter the prespective ratio: 0.3



THREE POINT PRESPECTIVE PROJECTION

Question 7: Write a program to draw Hermite /Bezier curve.

Hermite

Code:

```
#include <conio.h>
#include <graphics.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

struct point
{
    int x, y;
};

void hermite(point p1, point p4, double r1, double r4)
{
    float x, y, t;
    for (t = 0.0; t <= 1.0; t += 0.00005)
    {
        x = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * p1.x +
            (-2 * pow(t, 3) + 3 * pow(t, 2)) * p4.x +
            (pow(t, 3) - 2 * pow(t, 2) + t) * r1 +
            (pow(t, 3) - pow(t, 2)) * r4;
        y = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * p1.y +
            (-2 * pow(t, 3) + 3 * pow(t, 2)) * p4.y +
            (pow(t, 3) - 2 * pow(t, 2) + 1) * r1 +
```



```

        (pow(t, 3) - pow(t, 2)) * r4;
    putpixel(x, y, WHITE);
}

circle(p1.x, p1.y, 3);
circle(p4.x, p4.y, 3);
}

int main()
{
    point p1, p4;
    double r1, r4;

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "..\\BGI");

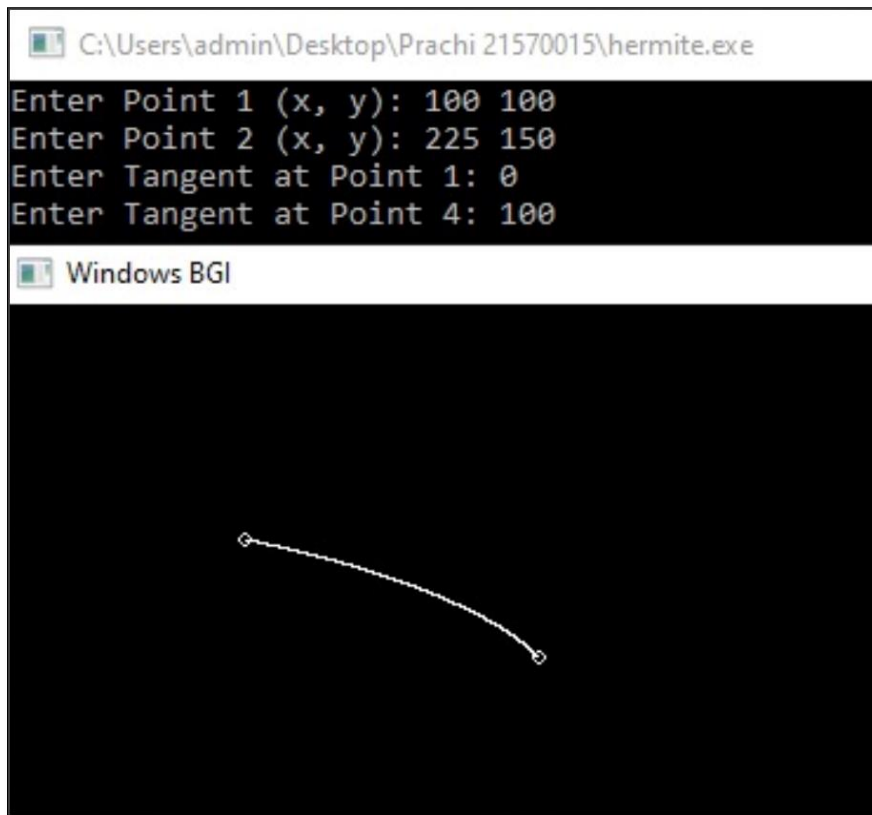
    cout << "Enter Point 1 (x, y): ";
    cin >> p1.x >> p1.y;
    cout << "Enter Point 2 (x, y): ";
    cin >> p4.x >> p4.y;
    cout << "Enter Tangent at Point 1: ";
    cin >> r1;
    cout << "Enter Tangent at Point 4: ";
    cin >> r4;

    hermite(p1, p4, r1, r4);

    getch();
    closegraph();
}

```

Output:



- Bezier

Code:

```
#include <conio.h>
#include <graphics.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

void bezier(int x[4], int y[4])
{
    for (double t = 0.0; t < 1.0; t += 0.00005)
```

```

    {
        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(xt, yt, WHITE);
    }

    for (int i = 0; i < 4; i++)
    {
        circle(x[i], y[i], 3);
    }

    getch();
    closegraph();
    return;
}

int main()
{
    int i;
    int x[4], y[4];
    int gd = DETECT, gm, errorcode;

    initgraph(&gd, &gm, "..\\bgi");

    for (i = 0; i < 4; i++)
    {
        cout << "Enter Point " << i + 1 << " (x, y): ";
        cin >> x[i] >> y[i];
    }

```

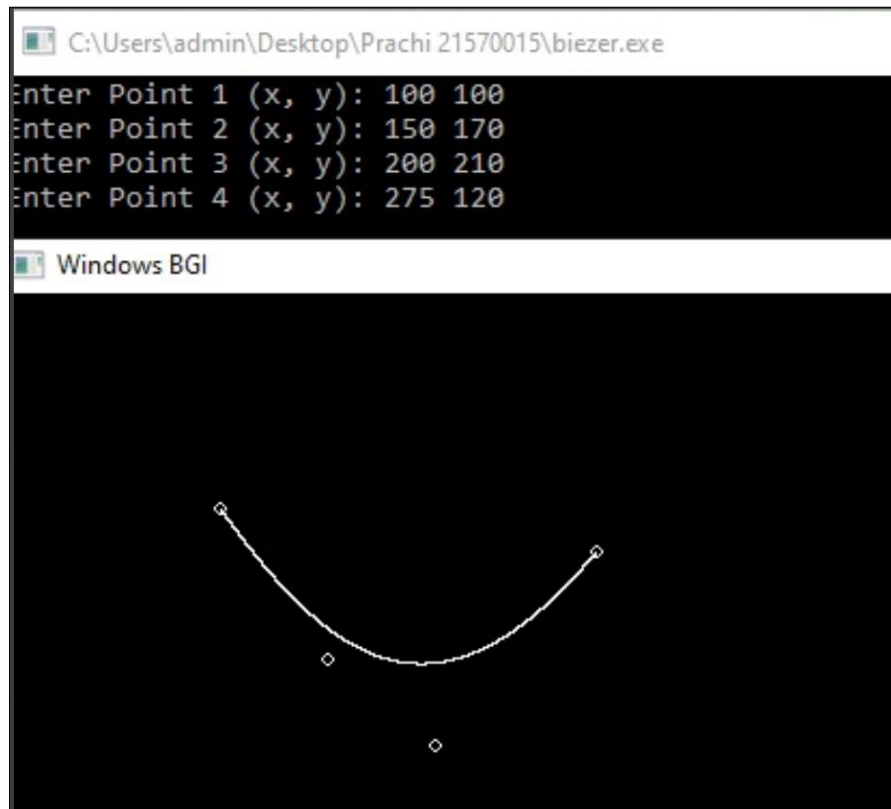
```
}
```

```
bezier(x, y);
```

```
return 0;
```

```
}
```

Output:



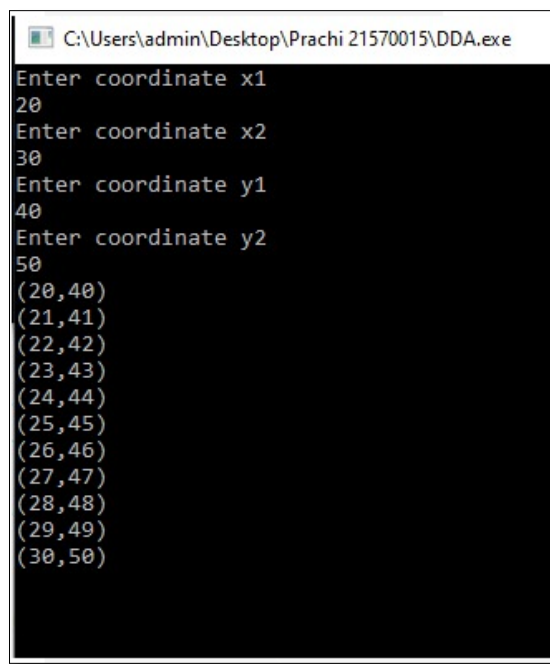
Question 8:Dda line algorithm

Code:

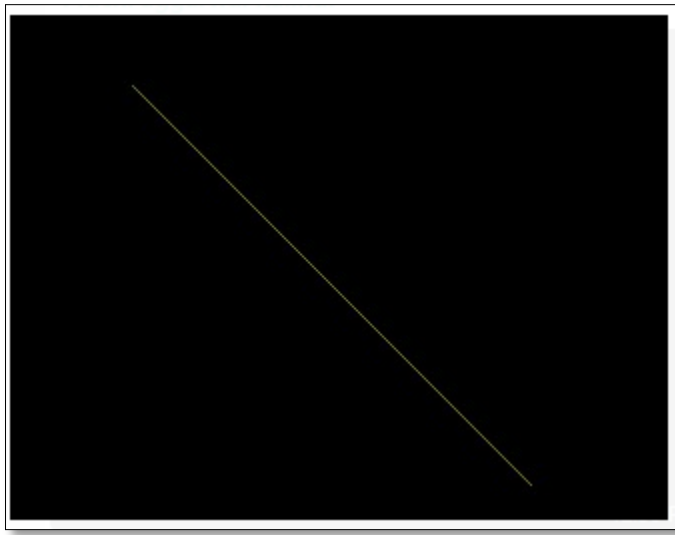
```
#include<iostream>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
using namespace std;
int DDA(int xa,int xb, int ya, int yb){
    int dx,dy,steps;
    dx=xb-xa;
    dy=yb-ya;
    float xInc, yInc;
    int x,y;
    x=xa,y=ya;
    if(abs(dx)>abs(dy)){
        steps = abs(dx);
    }
    else{
        steps = abs(dy);
    }
    xInc = dx/(float)steps;
    yInc = dy/(float)steps;
    for(int v =0 ; v<steps; v++){
        x = x+xInc;
        y = y+yInc;
        putpixel(round(x),round(y),6);
    }
}
int main(){
```

```
system("cls");  
int gd = DETECT, gm;  
initgraph(&gd, &gm, (char*)"");  
int x1,x2,y1,y2;  
cout<<"Enter coordinate x1"<<endl;  
cin>>x1;  
cout<<"Enter coordinate x2"<<endl;  
cin>>x2;  
cout<<"Enter coordinate y1"<<endl;  
cin>>y1;  
cout<<"Enter coordinate y2"<<endl;  
cin>>y2;  
DDA(x1,x2,y1,y2);  
getch();  
return 0;  
}
```

Output:



```
C:\Users\admin\Desktop\Prachi 21570015\DDA.exe  
Enter coordinate x1  
20  
Enter coordinate x2  
30  
Enter coordinate y1  
40  
Enter coordinate y2  
50  
(20,40)  
(21,41)  
(22,42)  
(23,43)  
(24,44)  
(25,45)  
(26,46)  
(27,47)  
(28,48)  
(29,49)  
(30,50)
```



Question 9:Mid-point line algorithm

Code:

```
#include<iostream>

#include<graphics.h>

#include<stdlib.h>

using namespace std;

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

    int dx,dy;

    dx=x2-x1;

    dy=y2-y1;

    if(dy<=dx){

        int d = dy - (dx/2);

        int x = x1,y=y1;

        cout<<"("<<x<<','<<y<<")"<<"\n";

        while(x<x2){

            putpixel(x,y,6);

            x++;

            if(d<0)

                d=d+dy;

            else{

                d += dy-dx;

                y++;

            }

            cout<<"("<<x<<','<<y<<")"<<"\n";

        }

    }

    else if(dx<dy){

        int d= dx - (dy/2);

        int x = x1,y=y1;

        cout<<"("<<x<<','<<y<<")"<<"\n";
```



```

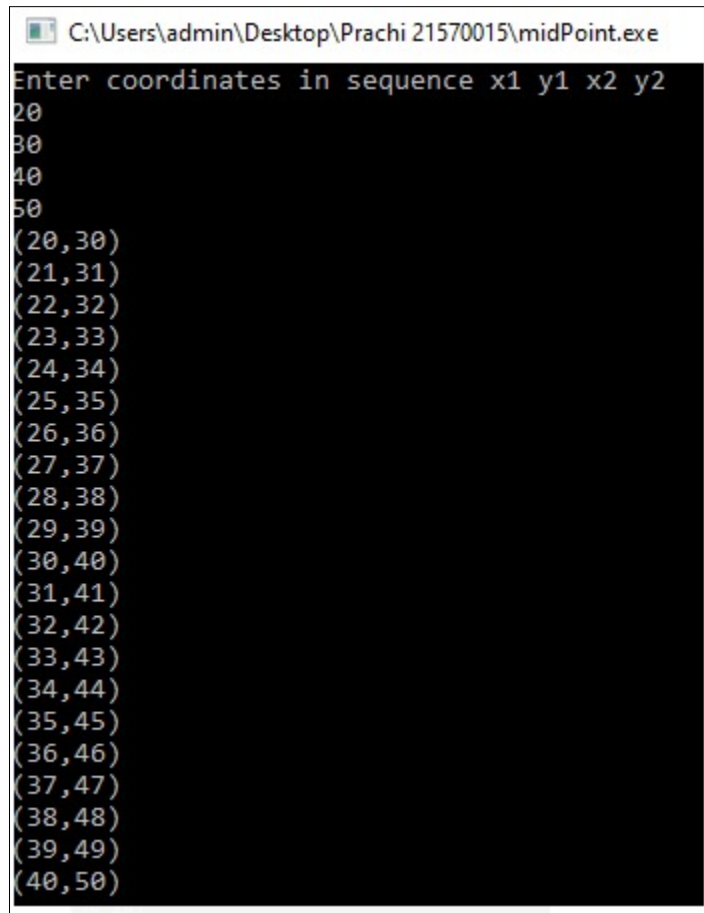
        while(y<y2){
            putpixel(x,y,6);
            y++;
            if(d<0){
                d=d+dx;
            }
            else{
                d += dx-dy;
                x++;
            }
            cout<<"("<<x<<', '<<y<<")"<<"\n";
        }
    }

}

int main(){
    system("cls");
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");
    int x1,x2,y1,y2;
    cout<<"Enter coordinates in sequence x1 y1 x2 y2"<<endl;
    cin>>x1>>y1>>x2>>y2;
    midPoint(x1,y1,x2,y2);
    getch();
    return 0;
}

```

Output:



```
C:\Users\admin\Desktop\Prachi 21570015\midPoint.exe
Enter coordinates in sequence x1 y1 x2 y2
20
30
40
50
(20,30)
(21,31)
(22,32)
(23,33)
(24,34)
(25,35)
(26,36)
(27,37)
(28,38)
(29,39)
(30,40)
(31,41)
(32,42)
(33,43)
(34,44)
(35,45)
(36,46)
(37,47)
(38,48)
(39,49)
(40,50)
```

