Practical-1

AIM: A brief study of various types of input and output devices.

1) INITGRAPH

- Initializes the graphics system.
- Void far initgraph(int far *graphdriver)
- To start the graphic system, you must first call initgraph.
- Initgraph initializes the graphic system by loading a graphics driver fromdisk (or validating a registered driver) then putting the system into graphics mode.
- Initgraph also resets all graphics settings (color, palette, current position, viewport, etc) to their defaults then resets graph.

2) GETPIXEL, PUTPIXEL

- Getpixel gets the color of a specified pixel.
- Putpixel places a pixel at a specified point.
- Unsigned far getpixel(int x, int y)
- Void far putpixel(int x, int y, int color)
- Getpixel getsthe color ofthe pixel located at (x,y);
- Putpixel plots a point in the color defined at (x, y).
- Getpixelreturns the color ofthe given pixel.
- Putpixel does not return.

3) CLOSE GRAPH

- Shuts down the graphic system.
- Void far closegraph(void);
- Close graph deallocates all memory allocated by the graphic system.
- It then restores the screen to the mode it was in before you called initgraph.

4) ARC, CIRCLE, PIESLICE

- arc draws a circular arc.
- Circle draws a circle
- Pieslice draws and fills a circular pieslice
- Void far arc(int x, int y, int stangle, int endangle, int radius);
- Void far circle(int x, int y, int radius);
- Void far pieslice(int x, int y, int stangle, int endangle, int radius);
- Arc draws a circular arc in the current drawing color
- Circle draws a circle in the current drawing color
- Pieslice draws a pieslice in the current drawing color, then fills it using the current fill pattern and fill color.

5) ELLIPSE, FILL ELLIPSE, SECTOR

- Ellipse draws an elliptical arc.
- Fill ellipse draws and fill ellipse.
- Sector draws and fills an elliptical pie slice.
- Void far ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius)
- Void far fill ellipse(int x, int y, int xradius, int yradius)
- Void farsectoe(int x, int y, int stangle, int endangle, int xradius, int yradius)
- Ellipse draws an elliptical arc in the current drawing color.
- Fill ellipse draws an elliptical arc in the current drawing color and then fills it with fill color and fill pattern.
- Sector draws an elliptical pie slice in the current drawing color and then fills it using the pattern and color defined by setfill style or setfill pattern.

6) FLOODFILL

- Flood-fills a bounded region.
- Void far floodfill(int x, int y, int border)
- Floodfills an enclosed area on bitmap device.
- The area bounded by the color border is flooded with the current fill pattern and fill color.
- (x,y) is a “seed point”
- If the seed is within an enclosed area, the inside will be filled.
- If the seed is outside the enclosed area, the exterior will be filled.
- Use fillpoly instead of floodfill wherever possible so you can maintain code compatibility with future versions.
- Floodfill does not work with the IBM-8514 driver.
- If an error occurs while flooding a region, the graph result returns „1“.

7) GETCOLOR, SETCOLOR

- Getcolor returns the current drawing color.
- Setcolor returns the current drawing color.
- Int far getcolor(void);
- Void far setcolor(int color)
- Getcolor returns the current drawing color.
- Setcolor sets the current drawing color to color, which can range from 0 to get max color.
- To set a drawing color with setcolor ,you can pass either the color number or the equivalent color name.

8) LINE,LINEREL,LINETO

- Line draws a line between two specified pints.
- Onerel draws a line relative distance from current position (CP).
- Linrto draws a line from the current position (CP) to(x,y).
- Void far lineto(int x, int y)
- Line draws a line from (x1, y1) to (x2, y2) using the current color, line style and thickness. It does not update the current position (CP).
- Linerel draws a line from the CP to a point that is relative distance (dx, dy) from the CP, then advances the CP by (dx, dy).
- Lineto draws a line from the CP to (x, y), then moves the CP to (x,y).

9) RECTANGLE

- Draws a rectangle in graphics mode.
- Void far rectangle (int left, int top, int right, int bottom)
- It draws a rectangle in the current line style, thickness and drawing color.
- (left, top) is the upper left corner of the rectangle, and (right, bottom) is its lower right corner.

Practical 2

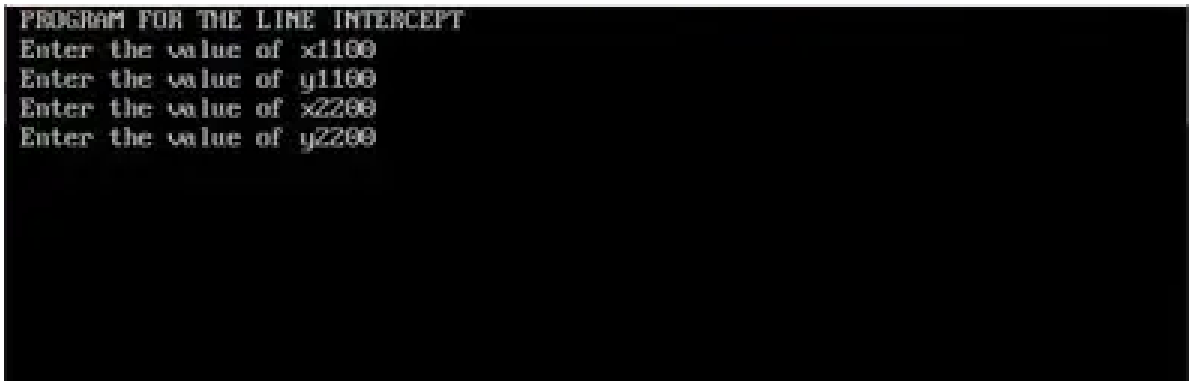
AIM: Write a program to implement a line using slope intercept formula.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
        float m,x1,y1,x2,y2;
        int x,y;
        int gdriver=DETECT,gmode,gerror;
        clrscr();
        printf(" PROGRAM FOR THE LINE INTERCEPT \n");

        printf(" Enter the value of x1");
        scanf("%f",&x1);

        printf(" Enter the value of y1");
        scanf("%f",&y1);

        printf(" Enter the value of x2");
        scanf("%f",&x2);

        printf(" Enter the value of y2");
        scanf("%f",&y2);
```

```
        initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");
        m=(y2-y1)/(x2-x1);
        for(x=1;x<=x2;x++)
        {
                y=m*(x-x1)+y1;
                putpixel(x,y,15);
                delay(50);
        }
        getch();
        closegraph();
}
```

Output:



Practical-3

AIM: Write a program to implement line using DDA algorithm.

Code:

```
#include<conio.h>
#include<iostream.h>
#include<graphics.h>
#include<math.h>
#include<dos.h>
void main()
{
        float x,y,deltax,deltay;
        int x1,y1,x2,y2,i,len;
        int gd = DETECT, gm;

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

        cout<<"Enter the Co-ordinates of starting point of line: ";
        cin>>x1>>y1;

        cout<<"Enter the Co-ordinates of end point of line ";
        cin>>x2>>y2;

        deltax = abs(x2-x1);
        deltay = abs(y2-y1);
        if(deltax>=deltay)
        {
                Len = deltax;
        }
        else
```

```
        {
                len = deltay;
        }
        deltax = deltax/len;
        deltay = deltay/len;
        x=x1;
        y=y1;
        i=1;
        while(i<=len)
        {
                putpixel(x,y,WHITE);
                x = x + deltax;
                y = y + deltay;
                i++;
                delay(20);
        }
        getch();
        closegraph();
}
```

Output:

Practical -4

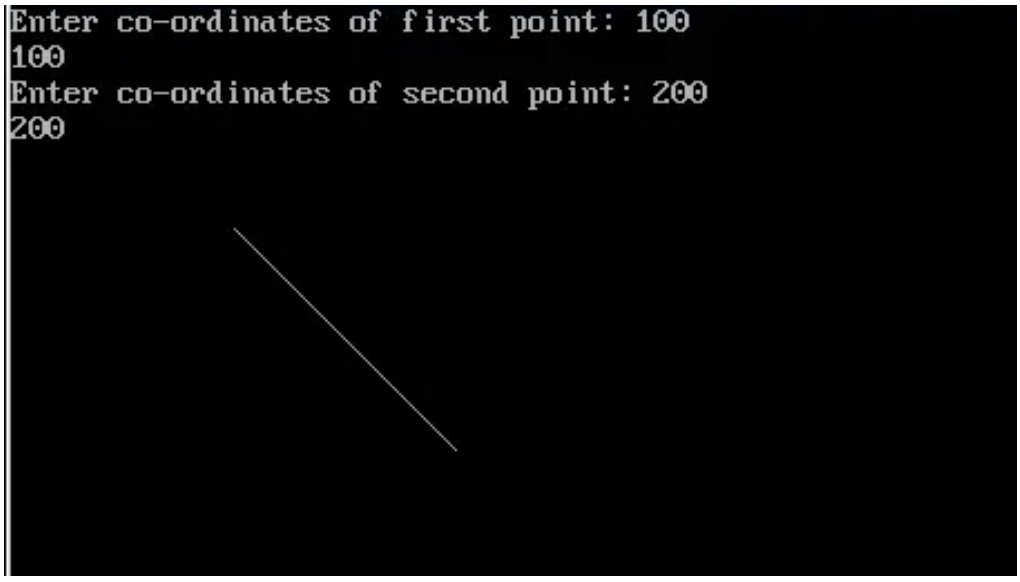
AIM: Write a program to implement line using Bresenham's algorithm.

Code:

```
#include<iostream.h>
#include<graphics.h>
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,7);
                        p=p+2*dy;
                }
                x=x+1;
```

```
        }
}
int main()
{
        int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
        initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

        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);

        return 0;
}
```

Output:



Practical -5

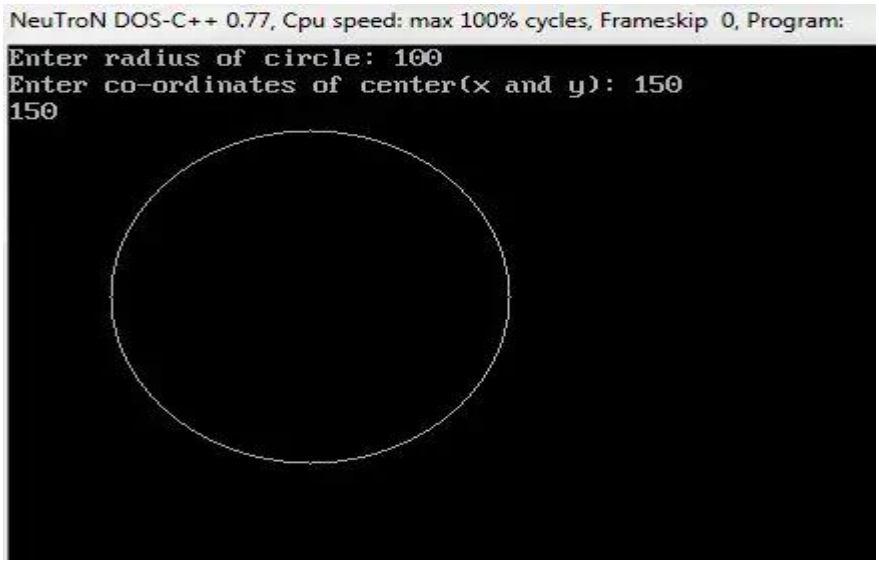
AIM: Write a program to implement circle using midpoint algorithm.

Code:

```
#include<iostream.h>
#include<graphics.h>
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()
{
    int gdriver=DETECT, gmode, error, x, y, r;
    initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    cout<<"Enter radius of circle: ";
    cin>>r;

    cout<<"Enter co-ordinates of center(x and y): ";
    cin>>x>>y;

    drawcircle(x, y, r);

    return 0;
}
```

Output:



Practical -6

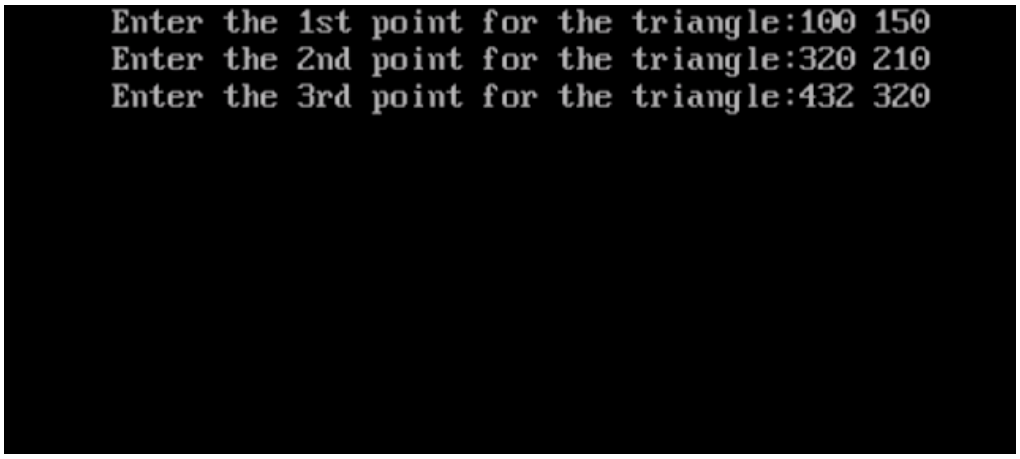
AIM: Write a program to implement translation of a line and triangle.

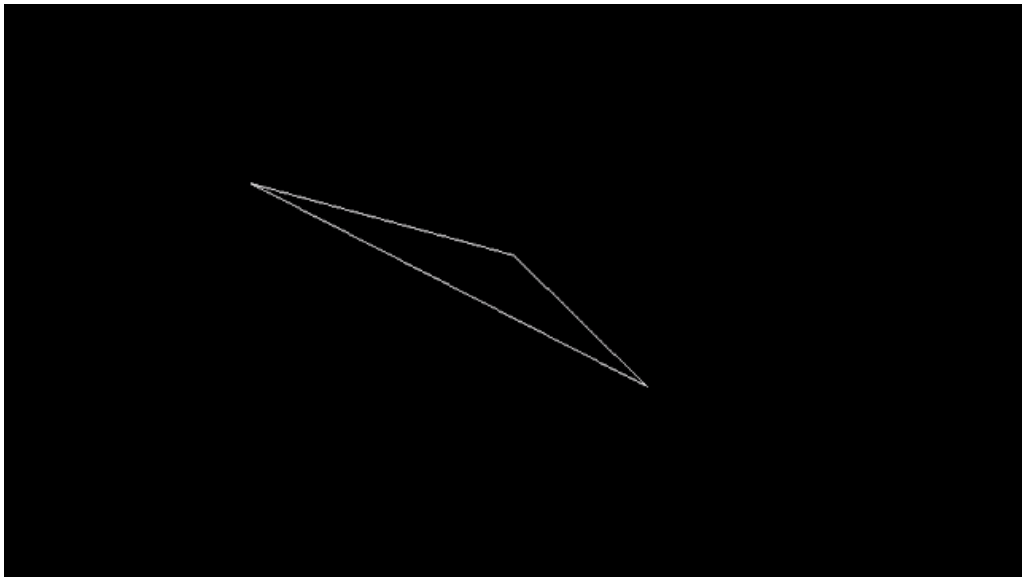
Code: (Triangle)

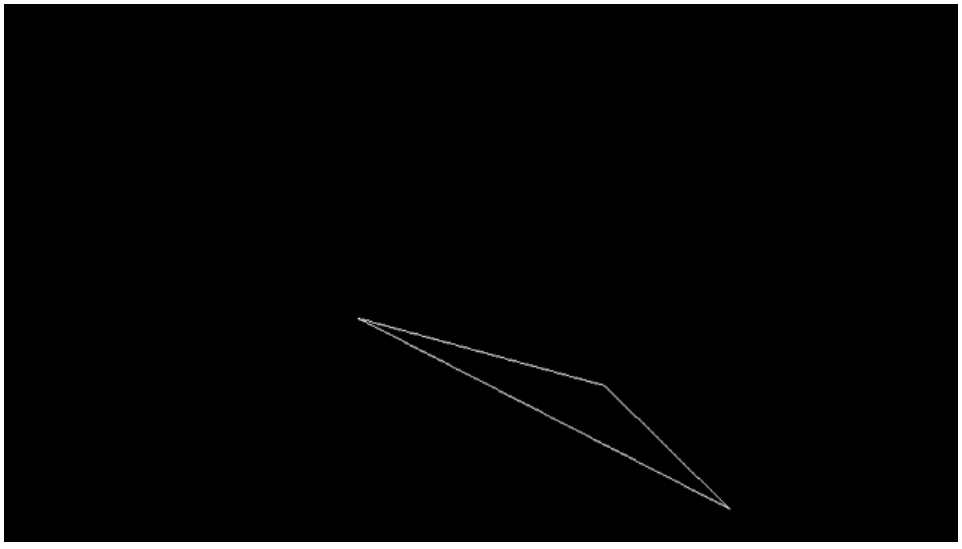
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>

int x1,y1,x2,y2,x3,y3,mx,my;
void draw();
void tri();
void main()
{
        int gd=DETECT,gm;
        int c;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

        printf("Enter the 1st point for the triangle:");
        scanf("%d%d",&x1,&y1);

        printf("Enter the 2nd point for the triangle:");
        scanf("%d%d",&x2,&y2);

        printf("Enter the 3rd point for the triangle:");
        scanf("%d%d",&x3,&y3);

        cleardevice();
        draw();
```

```
        getch();
        tri();
        getch();
}
void draw()
{
        line(x1,y1,x2,y2);
        line(x2,y2,x3,y3);
        line(x3,y3,x1,y1);
}
void tri()
{
        int x,y,a1,a2,a3,b1,b2,b3;

        printf("Enter the Transaction coordinates");
        scanf("%d%d",&x,&y);

        cleardevice();

        a1=x1+x;
        b1=y1+y;
        a2=x2+x;
        b2=y2+y;
        a3=x3+x;
        b3=y3+y;
        line(a1,b1,a2,b2);
        line(a2,b2,a3,b3);
        line(a3,b3,a1,b1);
}
```

Output:


Enter the 1st point for the triangle:100 150
Enter the 2nd point for the triangle:320 210
Enter the 3rd point for the triangle:432 320





Code: (line)

```
#include<conio.h>
#include<graphics.h>
#include<stdio.h>
void main()
{
	int gd=DETECT,gm;
	int l[2][2],v[2]={10,15},i=0,j;\

	clrscr();
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	printf("Enter the initial and final coordinates of a line ");

	while(i<2)
	{
		printf("x%d and y%d = ",i,i);
		j=0;
		scanf("%d",&l[i][j]);
		scanf("%d",&l[i][j+1]);
		i++;
	}
	line(l[0][0],l[0][1],l[1][0],l[1][1]);
	setcolor(BLUE);
	line(l[0][0]+v[0],l[0][1]+v[1],l[1][0]+v[0],l[1][1]+v[1]);
	getch();
	closegraph();
}
```

Output:


Enter the initial and final coordinates of a line x0 and y0 = 200 300
x1 and y1 = 300 200

Practical -7

AIM: Write a program to implement rotation of a line and triangle.

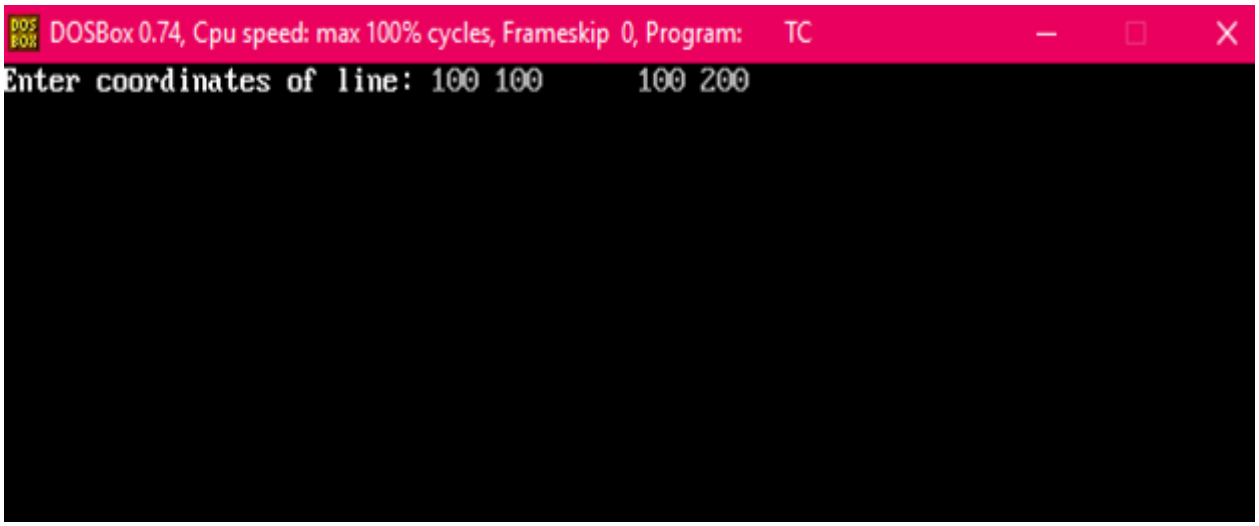
Code: (line)

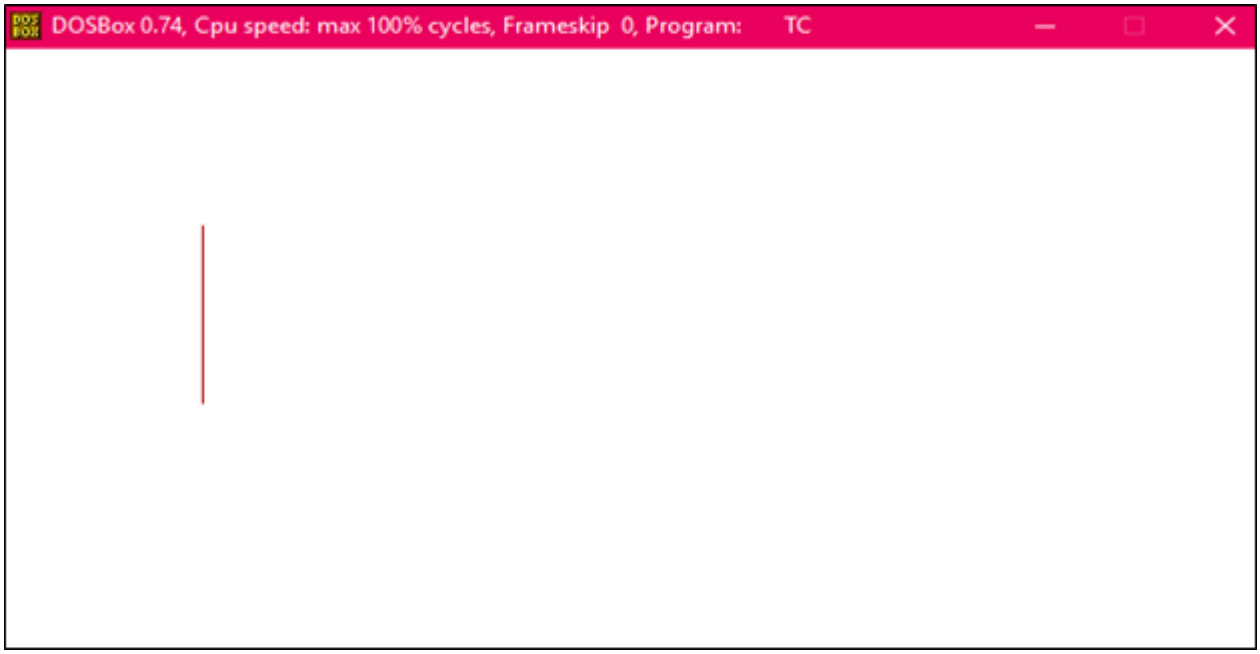
```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
int main()
{
        intgd=0,gm,x1,y1,x2,y2;
        double s,c, angle;
        initgraph(&gd, &gm,"C:\\TURBOC3\\BGI");
        setcolor(RED);

        printf("Enter coordinates of line: ");
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

        cleardevice();
        setbkcolor(WHITE);
        line(x1,y1,x2,y2);
        getch();
        setbkcolor(BLACK);

        printf("Enter rotation angle: ");
        scanf("%lf", &angle);
        setbkcolor(WHITE);

        c = cos(angle *3.14/180);
        s = sin(angle *3.14/180);
```

```
        x1 = floor(x1 * c + y1 * s);
        y1 = floor(-x1 * s + y1 * c);
        x2 = floor(x2 * c + y2 * s);
        y2 = floor(-x2 * s + y2 * c);

        cleardevice();
        line(x1, y1 ,x2, y2);
        getch();
        closegraph();
        return 0;
}
```

Output:

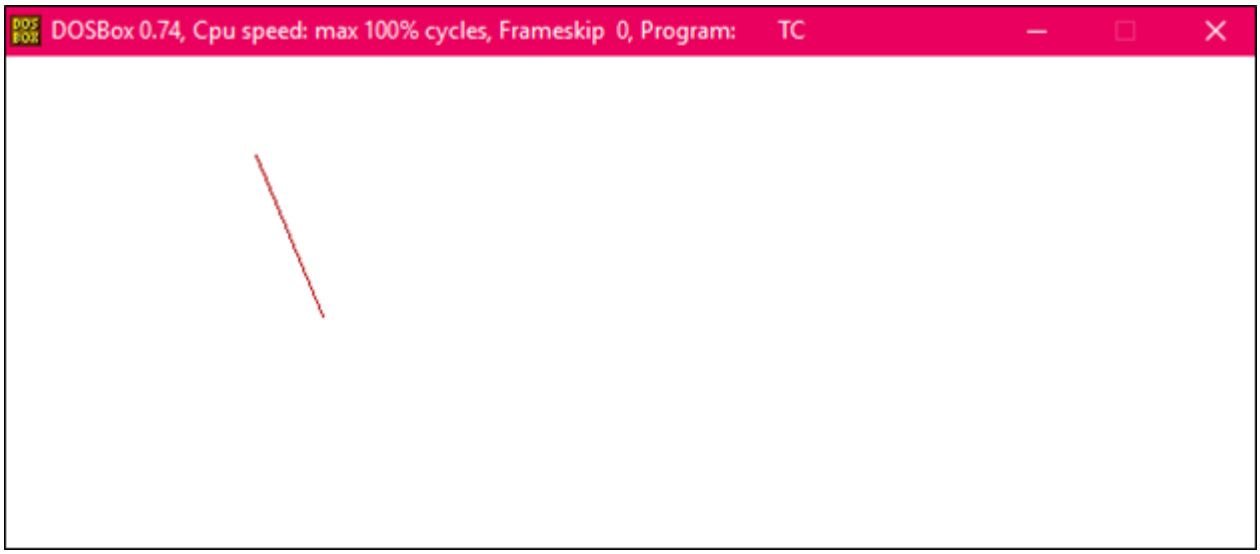
Before rotation




DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter rotation angle: 45

After rotation



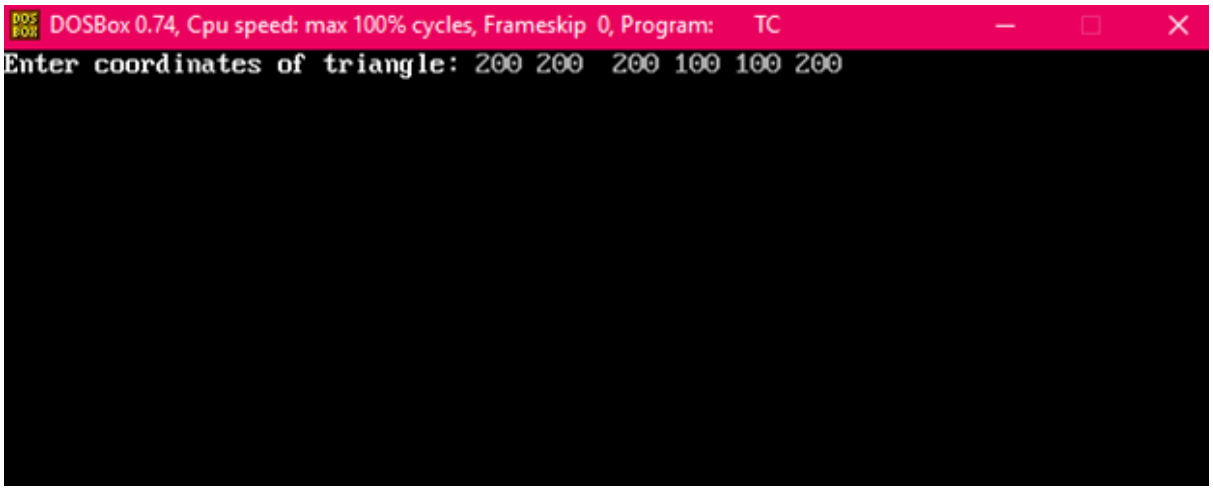
Code: (Triangle)

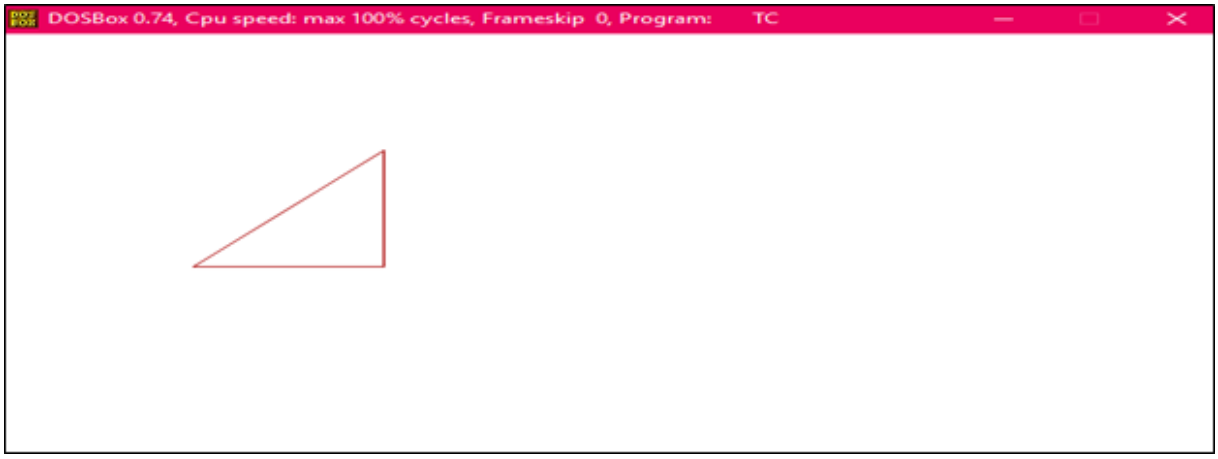
```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
        intgd=0,gm,x1,y1,x2,y2,x3,y3;
        double s,c, angle;
        initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
        setcolor(RED);

        printf("Enter coordinates of triangle: ");
        scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2, &x3, &y3);
        setbkcolor(WHITE);
        cleardevice();

        line(x1,y1,x2,y2);
        line(x2,y2, x3,y3);
        line(x3, y3, x1, y1);
        getch();
        setbkcolor(BLACK);

        printf("Enter rotation angle: ");
        scanf("%lf", &angle);

        setbkcolor(WHITE);

        c = cos(angle *M_PI/180);
        s = sin(angle *M_PI/180);
```

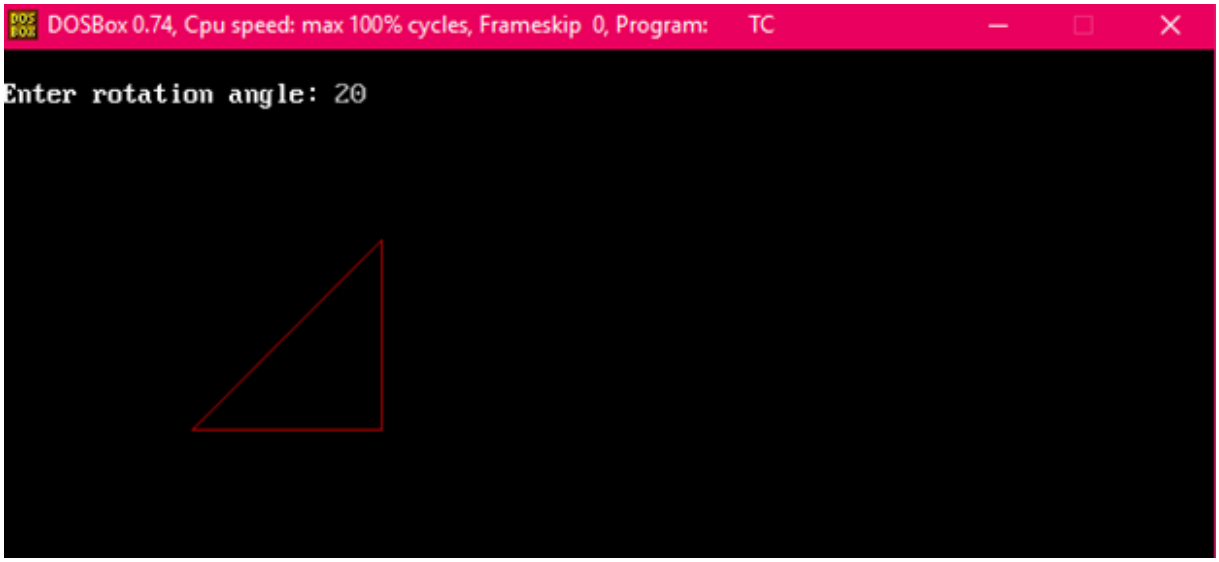
```
        x1 = floor(x1 * c + y1 * s);
        y1 = floor(-x1 * s + y1 * c);
        x2 = floor(x2 * c + y2 * s);
        y2 = floor(-x2 * s + y2 * c);
        x3 = floor(x3 * c + y3 * s);
        y3 = floor(-x3 * s + y3 * c);

        cleardevice();
        line(x1, y1 ,x2, y2);
        line(x2,y2, x3,y3);
        line(x3, y3, x1, y1);
        getch();
        closegraph();
        return 0;
}
```

Output:

Before rotation







After rotation

Practical -8

AIM: Write a program to implement scaling transformation.

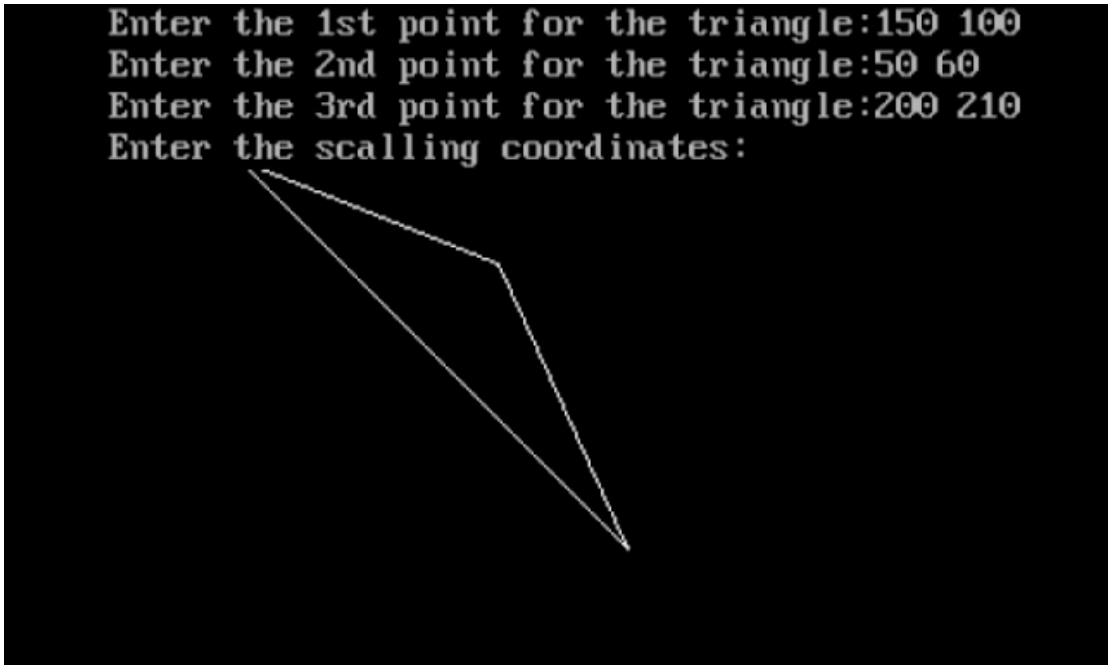
Code

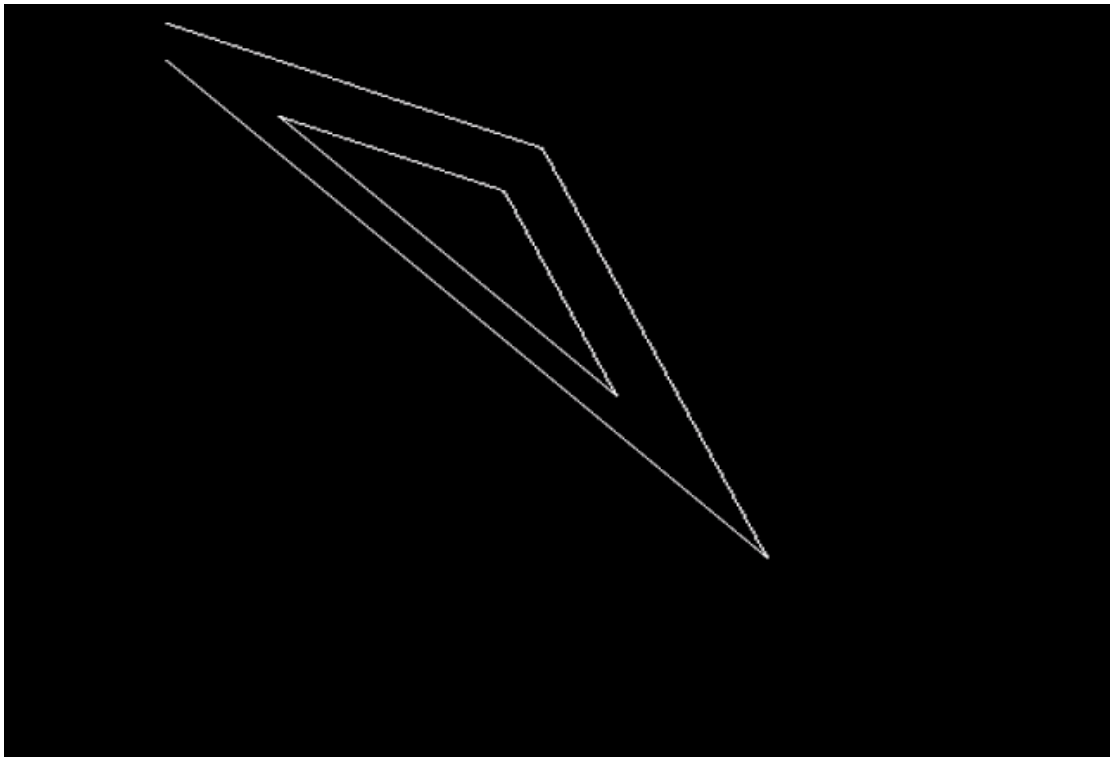
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>
int x1,y1,x2,y2,x3,y3,mx,my;
void draw();
void scale();
 void main()
{
        int gd=DETECT,gm;
        int c;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

        printf("Enter the 1st point for the triangle:");
        scanf("%d%d",&x1,&y1);

        printf("Enter the 2nd point for the triangle:");
        scanf("%d%d",&x2,&y2);

        printf("Enter the 3rd point for the triangle:");
        scanf("%d%d",&x3,&y3);

        draw();
        scale();
}
```

```
void draw()
{
        line(x1,y1,x2,y2);
        line(x2,y2,x3,y3);
        line(x3,y3,x1,y1);
}
void scale()
{
        int x,y,a1,a2,a3,b1,b2,b3;
        int mx,my;

        printf("Enter the scalling coordinates");
        scanf("%d%d",&x,&y);

        mx=(x1+x2+x3)/3;
        my=(y1+y2+y3)/3;
        cleardevice();

        a1=mx+(x1-mx)*x;
        b1=my+(y1-my)*y;

        a2=mx+(x2-mx)*x;
        b2=my+(y2-my)*y;

        a3=mx+(x3-mx)*x;
        b3=my+(y3-my)*y;

        line(a1,b1,a2,b2);
        line(a2,b2,a3,b3);
        line(a3,b3,a1,b1);
```

```
    draw();
    getch();

}
```

Output:





Practical -9

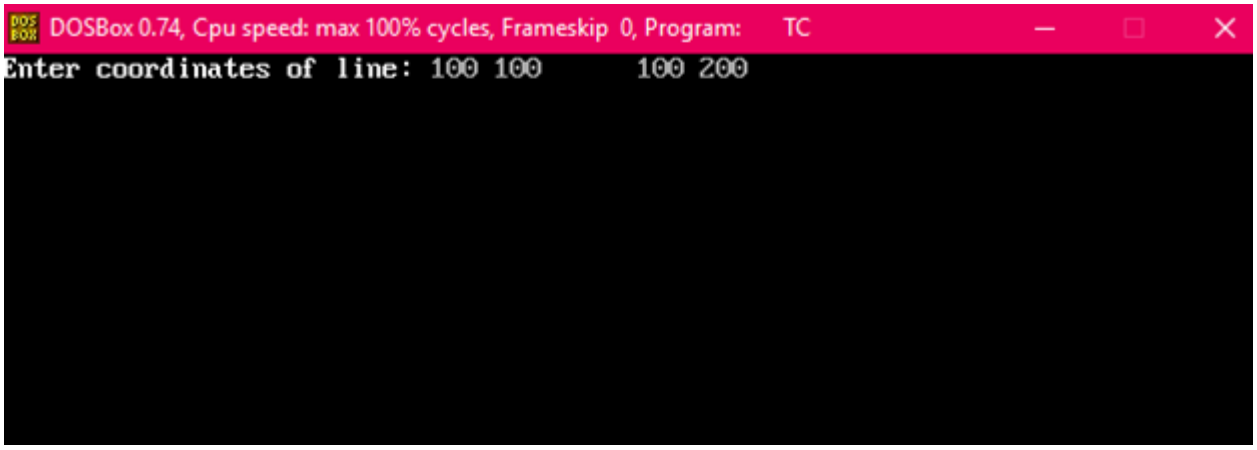
AIM: Write a program to implement 3d rotation about an arbitrary axis.

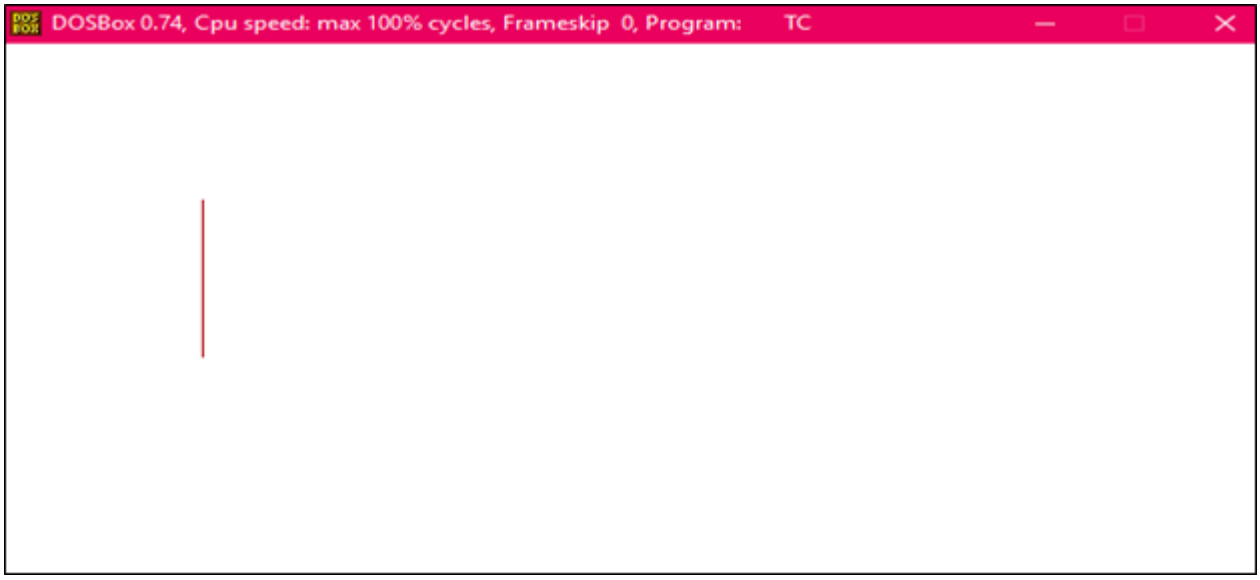
Code:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
int main()
{
        intgd=0,gm,x1,y1,x2,y2;
        double s,c, angle;
        initgraph(&gd, &gm, "C:\\TC\\BGI");
        setcolor(RED);

        printf("Enter coordinates of line: ");
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

        cleardevice();
        setbkcolor(WHITE);
        line(x1,y1,x2,y2);
        getch();

        setbkcolor(BLACK);
        printf("Enter rotation angle: ");
        scanf("%lf", &angle);

        setbkcolor(WHITE);

        c = cos(angle *3.14/180);
        s = sin(angle *3.14/180);
```

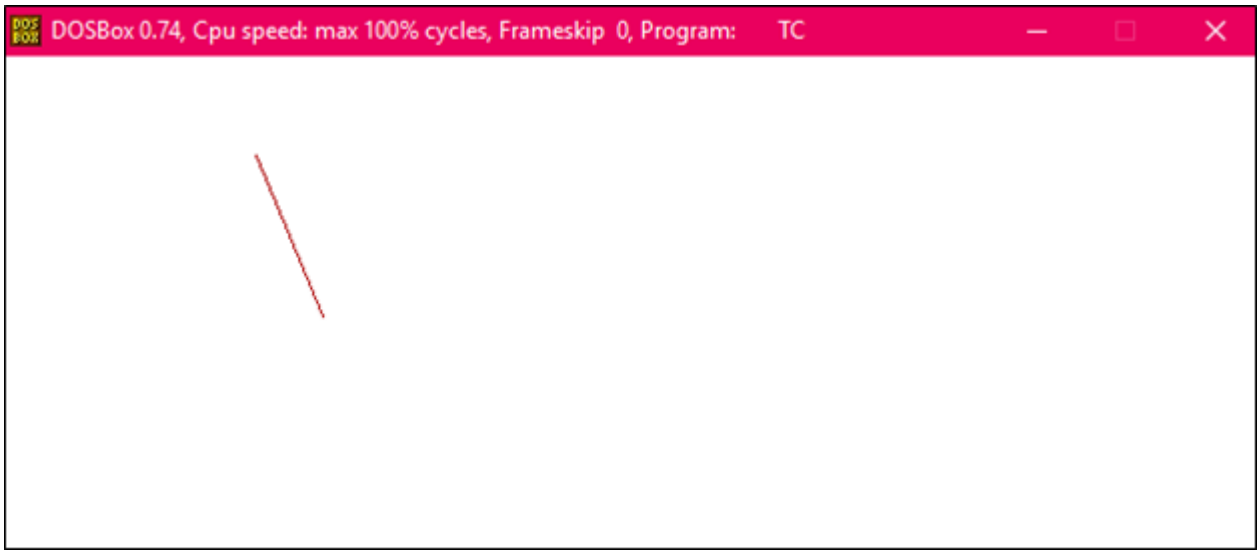
```
        x1 = floor(x1 * c + y1 * s);
        y1 = floor(-x1 * s + y1 * c);
        x2 = floor(x2 * c + y2 * s);
        y2 = floor(-x2 * s + y2 * c);

        cleardevice();
        line(x1, y1 ,x2, y2);
        getch();
        closegraph();
        return 0;
}
```

Output:






DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter rotation angle: 45


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Practical -10

AIM: Write a program to implement Cohen Sutherland Line Clipping.

Code:

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>
typedef struct coordinate
{
        int x,y;
        char code[4];
}PT;

void drawwindow();
void drawline(PT p1,PT p2);
PT setcode(PT p);
int visibility(PT p1,PT p2);
PT resetendpt(PT p1,PT p2);

void main()
{
        int gd=DETECT,v,gm;
        PT p1,p2,p3,p4,ptemp;

        cout<<"\nEnter x1 and y1\n";
        cin>>p1.x>>p1.y;
```

```
cout<<"\nEnter x2 and y2\n";
cin>>p2.x>>p2.y;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
drawwindow();
delay(500);
drawline(p1,p2);
delay(500);
cleardevice();
delay(500);

p1=setcode(p1);
p2=setcode(p2);
v=visibility(p1,p2);
delay(500);
switch(v)
{
        case 0:
                drawwindow();
                delay(500);
                drawline(p1,p2);
                break;
        case 1:
                drawwindow();
                delay(500);
                break;
        case 2:
                p3=resetendpt(p1,p2);
                p4=resetendpt(p2,p1);
                drawwindow();
```

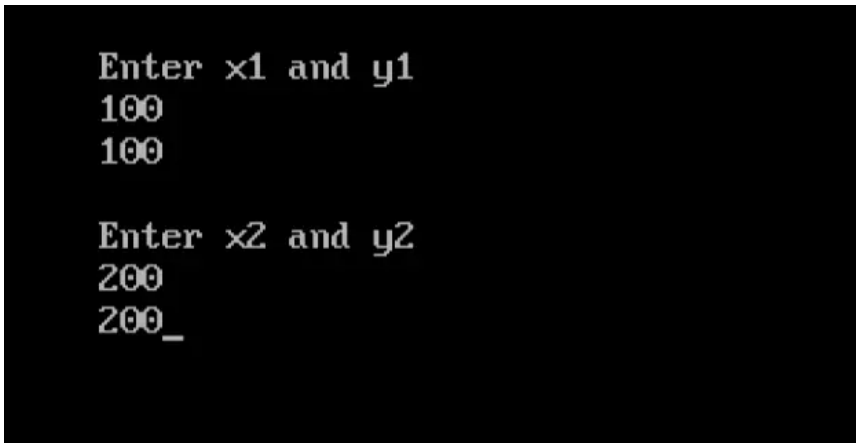
```
                delay(500);
                drawline(p3,p4);
                break;
        }
        delay(5000);
        closegraph();
}

void drawwindow()
{
        line(150,100,450,100);
        line(450,100,450,350);
        line(450,350,150,350);
        line(150,350,150,100);
}

void drawline(PT p1,PT p2)
{
        line(p1.x,p1.y,p2.x,p2.y);
}

PT setcode(PT p) //for setting the 4 bit code
{
        PT ptemp;
        if(p.y<100)
                ptemp.code[0]='1'; //Top
        else
                ptemp.code[0]='0';
        if(p.y>350)
                ptemp.code[1]='1'; //Bottom
```

```
        else
                ptemp.code[1]='0';
        if(p.x>450)
                ptemp.code[2]='1'; //Right
        else
                ptemp.code[2]='0';
        if(p.x<150)
                ptemp.code[3]='1'; //Left
        else
                ptemp.code[3]='0';
                ptemp.x=p.x;
                ptemp.y=p.y;
                return(ptemp);
}

int visibility(PT p1,PT p2)
{
        int i,flag=0;
        for(i=0;i<4;i++)
        {
                if((p1.code[i]!='0') || (p2.code[i]!='0'))
                flag=1;
        }
        if(flag==0)
                return(0);
                for(i=0;i<4;i++)
                {
                        if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))
                        flag='0';
```

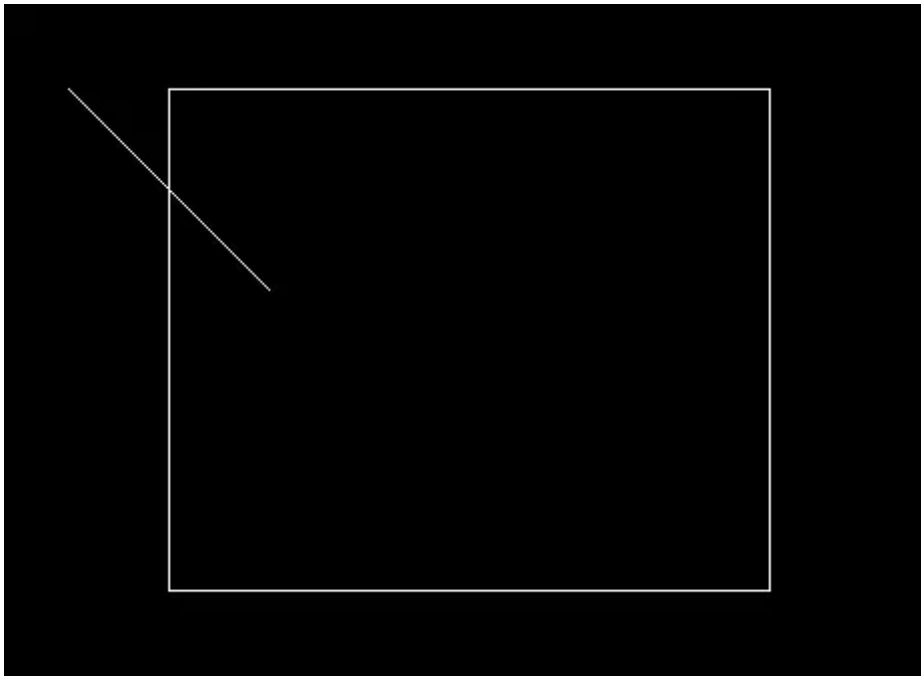
```
                }
        if(flag==0)
        return(1);
        return(2);
}

PT resetendpt(PT p1,PT p2)
{
        PT temp;
        int x,y,i;
        float m,k;
        if(p1.code[3]=='1')
                x=150;

        if(p1.code[2]=='1')
                x=450;

        if((p1.code[3]=='1') || (p1.code[2]=='1'))
        {
                m=(float)(p2.y-p1.y)/(p2.x-p1.x);
                k=(p1.y+(m*(x-p1.x)));
                temp.y=k;
                temp.x=x;
                for(i=0;i<4;i++)
                        temp.code[i]=p1.code[i];
                        if(temp.y<=350 && temp.y>=100)
                        return (temp);
        }
```

```
        if(p1.code[0]=='1')
                y=100;
        if(p1.code[1]=='1')
                y=350;
        if((p1.code[0]=='1') || (p1.code[1]=='1'))
        {
                m=(float)(p2.y-p1.y)/(p2.x-p1.x);
                k=(float)p1.x+(float)(y-p1.y)/m;
                temp.x=k;
                temp.y=y;
                for(i=0;i<4;i++)
                        temp.code[i]=p1.code[i];
                        return(temp);
        }
        else
                return(p1);
}
```

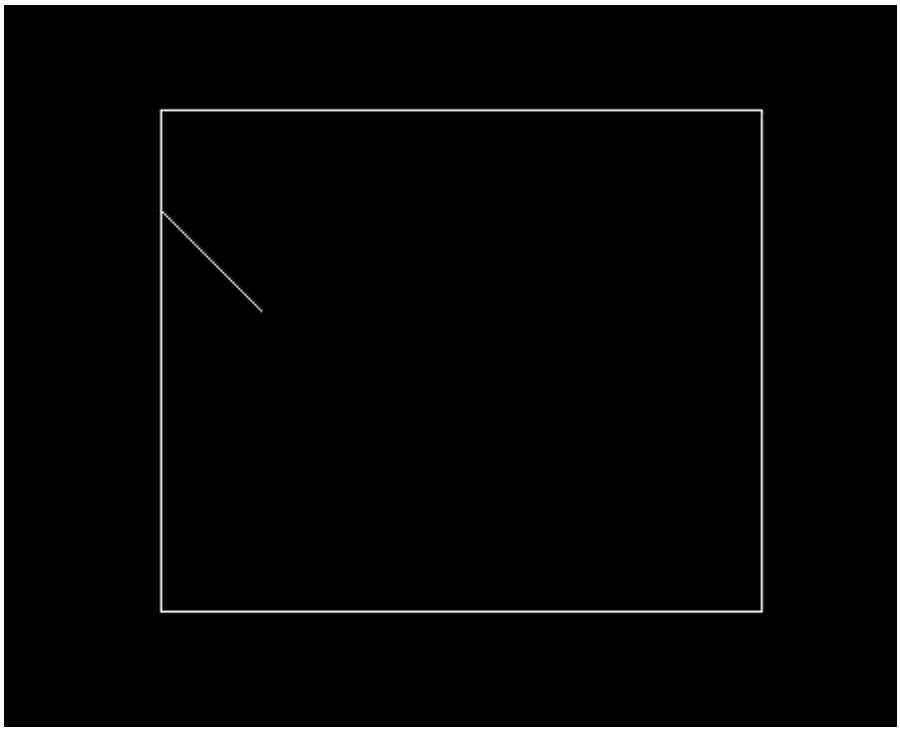
Output:



Before clipping



After clipping



Practical -11

AIM: Write a program to implement Sutherland Hodgeman Polygon Clipping.

Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#define round(a) ((int)(a+0.5))
int k;
float xmin,ymin,xmax,ymax,arr[20],m;
void clipl(float x1,float y1,float x2,float y2)
{
        if(x2-x1)
                m=(y2-y1)/(x2-x1);
         else
                m=100000;
        if(x1 >= xmin && x2 >= xmin)
        {
                arr[k]=x2;
                arr[k+1]=y2;
                k+=2;
        }
        if(x1 < xmin && x2 >= xmin)
        {
                arr[k]=xmin;
                arr[k+1]=y1+m*(xmin-x1);
                arr[k+2]=x2;
                arr[k+3]=y2;
                k+=4;
        }
```

```
        if(x1 >= xmin  && x2 < xmin)
        {
                arr[k]=xmin;
                arr[k+1]=y1+m*(xmin-x1);
                 k+=2;
        }
}
void clipt(float x1,float y1,float x2,float y2)
{
        if(y2-y1)
                m=(x2-x1)/(y2-y1);
        else
                m=100000;
        if(y1 <= ymax && y2 <= ymax)
        {
                arr[k]=x2;
                arr[k+1]=y2;
                k+=2;
        }
        if(y1 > ymax && y2 <= ymax)
        {
                arr[k]=x1+m*(ymax-y1);
                arr[k+1]=ymax;
                arr[k+2]=x2;
                arr[k+3]=y2;
                k+=4;
        }
        if(y1 <= ymax  && y2 > ymax)
        {
                arr[k]=x1+m*(ymax-y1);
```

```
                arr[k+1]=ymax;
                 k+=2;
        }
}

void clipr(float x1,float y1,float x2,float y2)
{
        if(x2-x1)
                m=(y2-y1)/(x2-x1);
        else
                m=100000;
        if(x1 <= xmax && x2 <= xmax)
        {
                arr[k]=x2;
                arr[k+1]=y2;
                k+=2;
        }
        if(x1 > xmax && x2 <= xmax)
        {
                arr[k]=xmax;
                arr[k+1]=y1+m*(xmax-x1);
                arr[k+2]=x2;
                arr[k+3]=y2;
                k+=4;
        }
        if(x1 <= xmax  && x2 > xmax)
        {
                arr[k]=xmax;
                arr[k+1]=y1+m*(xmax-x1);
                k+=2;
```

```
        }
}
void clipb(float x1,float y1,float x2,float y2)
{
        if(y2-y1)
                m=(x2-x1)/(y2-y1);
        else
                m=100000;
        if(y1 >= ymin && y2 >= ymin)
        {
                arr[k]=x2;
                arr[k+1]=y2;
                k+=2;
        }
        if(y1 < ymin && y2 >= ymin)
        {
                arr[k]=x1+m*(ymin-y1);
                arr[k+1]=ymin;
                arr[k+2]=x2;
                arr[k+3]=y2;
                 k+=4;
        }
        if(y1 >= ymin  && y2 < ymin)
        {
                arr[k]=x1+m*(ymin-y1);
                arr[k+1]=ymin;
                k+=2;
        }
}
```

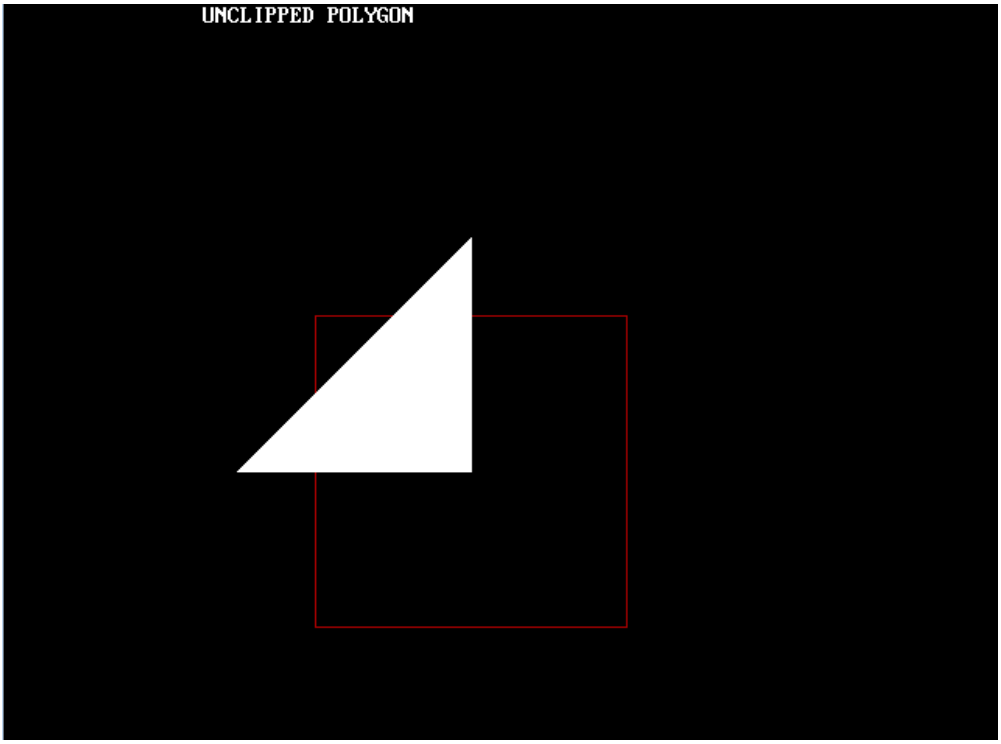
```
void main()
{
        int gdriver=DETECT,gmode,n,poly[20];
        float xi,yi,xf,yf,polyy[20];
        clrscr();

        cout<<"Coordinates of rectangular clip window :\nxmin,ymin           :";
        cin>>xmin>>ymin;

        cout<<"xmax,ymax           :";
        cin>>xmax>>ymax;

        cout<<"\n\nPolygon to be clipped :\nNumber of sides      :";
        cin>>n;

        cout<<"Enter the coordinates :";
        for(int i=0;i < 2*n;i++)
                cin>>polyy[i];
                polyy[i]=polyy[0];
                polyy[i+1]=polyy[1];

        for(i=0;i < 2*n+2;i++)
                poly[i]=round(polyy[i]);

        initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");
        setcolor(RED);
        rectangle(xmin,ymax,xmax,ymin);
        cout<<"\t\tUNCLIPPED POLYGON";
        setcolor(WHITE);
        fillpoly(n,poly);
```

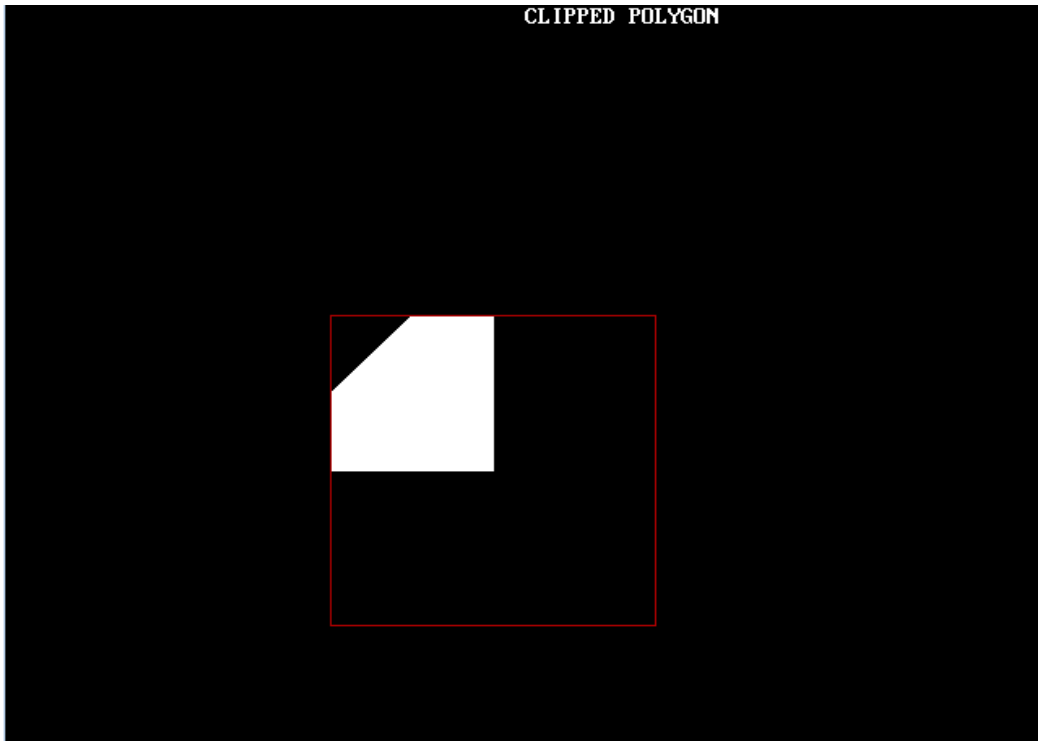
```
getch();
cleardevice();
k=0;
for(i=0;i < 2*n;i+=2)
        clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
        n=k/2;
for(i=0;i < k;i++)
        polyy[i]=arr[i];
        polyy[i]=polyy[0];
        polyy[i+1]=polyy[1];
        k=0;

for(i=0;i < 2*n;i+=2)
        clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
        n=k/2;

for(i=0;i < k;i++)
        polyy[i]=arr[i];
        polyy[i]=polyy[0];
        polyy[i+1]=polyy[1];
        k=0;

for(i=0;i < 2*n;i+=2)
        clipr(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
        n=k/2;

for(i=0;i < k;i++)
        polyy[i]=arr[i];
        polyy[i]=polyy[0];
        polyy[i+1]=polyy[1];
```

```
            k=0;

        for(i=0;i < 2*n;i+=2)
            clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

        for(i=0;i < k;i++)
            poly[i]=round(arr[i]);
            if(k)
                fillpoly(k/2,poly);
        setcolor(RED);
        rectangle(xmin,ymax,xmax,ymin);
        cout<<"\tCLIPPED POLYGON";
        getch();
        closegraph();
}
```

Output:

```
Coordinates of rectangular clip window :
xmin,ymin             :200 200
xmax,ymax             :400 400


Polygon to be clipped :
Number of sides       :3
Enter the coordinates :150 300
300 150
300 300
```


UNCLIPPED POLYGON


CLIPPED POLYGON

Practical -12

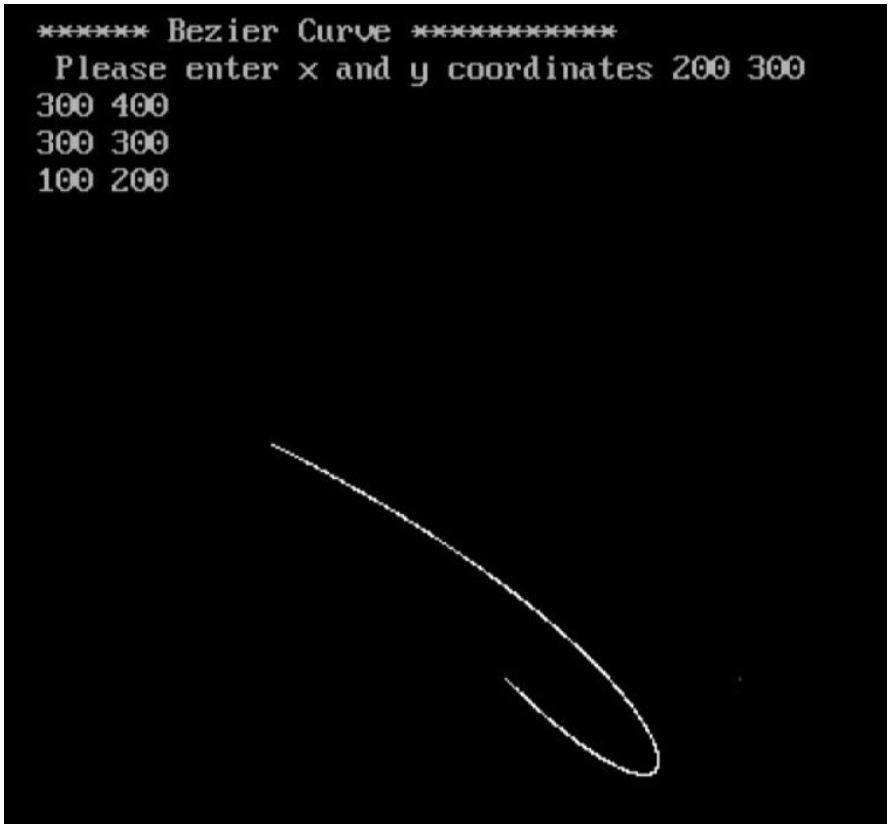
AIM: Write a program to draw Bezier curve.

Code:

```
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
void main()
{
        int x[4],y[4],i;
        double put_x,put_y,t;
        int gr=DETECT,gm;
        initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");

        printf("\n****** Bezier Curve **********");
        printf("\n Please enter x and y coordinates ");

        for(i=0;i<4;i++)
        {
                scanf("%d%d",&x[i],&y[i]);
                putpixel(x[i],y[i],3);              // Control Points
        }

        for(t=0.0;t<=1.0;t=t+0.001)            // t always lies between 0 and 1
        {
                put_x = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t*(1-t)*x[2] +
                pow(t,3)*x[3]; // Formula to draw curve
                put_y =  pow(1-t,3)*y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t*(1-t)*y[2] +
                pow(t,3)*y[3];
                putpixel(put_x,put_y, WHITE);          // putting pixel
```

```
        }
        getch();
        closegraph();
}
```

Output:



Practical- 13

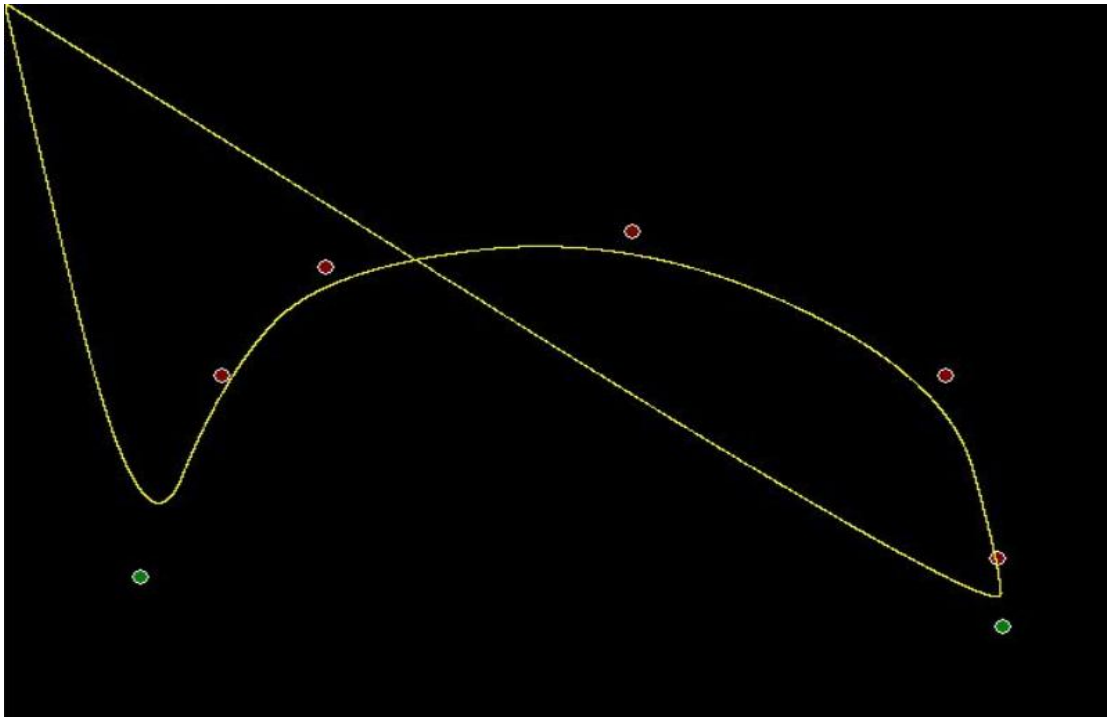
AIM: Write a program to b-spline curve.

Code:

```
void drawBSplineCurve(vector<point> poly)
{
	int n, d;

	cout << "Enter degree of curve: ";
	cin >> d;
	n = poly.size();
	vector<double> uVec;
	int i;

	for(i=0;i<n+d;i++)
	{
		uVec.push_back(((double)i)/(n+d-1));
	}
	double x, y, basis, u;
	for(u=0;u<=1;u+=0.0001)
	{
		x = 0;
		y = 0;
		for(i=0;i<poly.size();i++)
		{
			basis = blend(uVec, u, i, d);
			x += basis*poly[i].x;
			y += basis*poly[i].y;
		}
	putpixel(roundOff(x), roundOff(y), YELLOW);
```

```
        }
}
double blend(vector<double> &uVec, double u, int k, int d)
{
        if(d==1)
        {
                if(uVec[k]<=u && u<uVec[k+1])
                return 1;
                return 0;
        }
        double b;
        b = ((u-uVec[k])/(uVec[k+d-1]-uVec[k])*blend(uVec, u, k, d-1)) + ((uVec[k+d]-
u)/(uVec[k+d]-uVec[k+1])*blend(uVec, u, k+1, d-1));
        return b;
}
```

Output:



Practical -14

AIM: Write a program to make a moving colored car using inbuilt functions.

Code:

```
#include<graphics.h>
#include<conio.h>
int main()
{
        intgd=DETECT,gm, i, maxx, cy;
        initgraph(&gd, &gm,"C:\\TURBOC3\\BGI");
        setbkcolor(WHITE);
        setcolor(RED);
        maxx = getmaxx();
        cy = getmaxy()/2;

        for(i=0;i<maxx-140;i++)
        {
                cleardevice();
                line(0+i,cy-20, 0+i, cy+15);
                line(0+i, cy-20, 25+i, cy-20);
                line(25+i, cy-20, 40+i, cy-70);
                line(40+i, cy-70, 100+i, cy-70);
                line(100+i, cy-70, 115+i, cy-20);
                line(115+i, cy-20, 140+i, cy-20);
                line(0+i, cy+15, 18+i, cy+15);
                circle(28+i, cy+15, 10);
                line(38+i, cy+15, 102+i, cy+15);
                circle(112+i, cy+15,10);
                line(122+i, cy+15 ,140+i,cy+15);
                line(140+i, cy+15, 140+i, cy-20);
```

```
            rectangle(50+i, cy-62, 90+i, cy-30);
            setfillstyle(1,BLUE);
            floodfill(5+i, cy-15, RED);
            setfillstyle(1, LIGHTBLUE);
            floodfill(52+i, cy-60, RED);
            delay(10);
        }
         getch();
        closegraph();
        return 0;
}
```

Output:

Practical -15

AIM: Write a program to draw animation using increasing circles filled with different colors and patterns.

Code:

```
#include<graphics.h>
#include<conio.h>
void main()
{
        intgd=DETECT, gm, i, x, y;
        initgraph(&gd, &gm,"C:\\TURBOC3\\BGI");
        x=getmaxx()/3;
        y=getmaxx()/3;
        setbkcolor(WHITE);
        setcolor(BLUE);
        for(i=1;i<=8;i++)
       {
                setfillstyle(i,i);
                delay(20);
                circle(x, y, i*20);
                floodfill(x-2+i*20,y,BLUE);
        }
        getch();
        closegraph();
}
```

Output: