INDEX

PRACTICAL – 1

Objective: To study introduction of Computer Graphics

1. Introduction

Computer is information processing machine. User needs to communicate with computer and the computer graphics is one of the most effective and commonly used ways of communication with the user. It displays the information in the form of graphical objects such as pictures, charts, diagrams and graphs. Graphical objects convey more information in less time and easily understandable formats. In computer graphics, pictures or graphics objects are presented as a collection of discrete pixels.

2. Definition

Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images.

3. Applications

- Graphical User Interface: Multiple windows, icons, menus allow a computer setup to be utilized more efficiently.
- Computer aided drafting and design: It uses graphics to design components and system such as automobile bodies structures of building etc.
- Simulation and Animation: Use of graphics in simulation makes mathematical models and mechanical systems more realistic and easier to study.
- Process Control: Nowadays automation is used which is graphically displayed on the screen.
- Image Processing: It is used to process image by changing property of the image.
- Education and Training: Computer graphics can be used to generate models of physical, financial and economic systems. These models can be used as educational aids.

4. Advantages

- Computer graphics is one of the most effective and commonly used ways of communication with computer.
- It provides tool for producing pictures of ‘real-world’ as well as synthetic objects.
- It has ability to show moving pictures thus possible to produce animations with computer graphics.
- It provides tools called motion dynamics in which user can move objects as well as observes as per requirement.

- It provides tools call update dynamics. With this, we can change the shape color and other properties of object.
- Now in recent development of digital signal processing and audio synthesis chip the interactive graphics can now provide audio feedback along with graphical feedbacks.

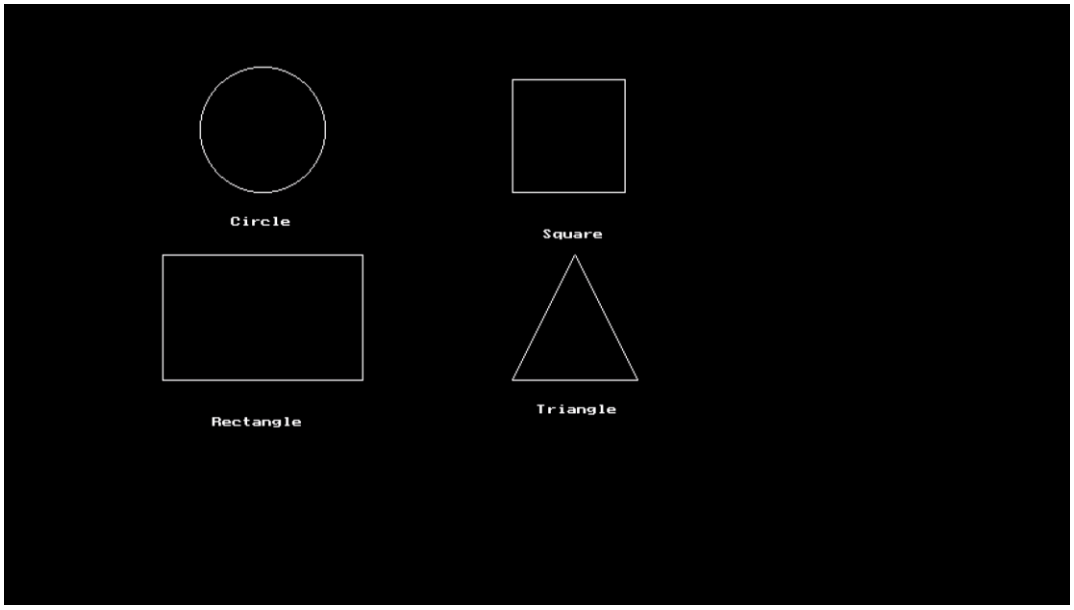
PRACTICAL – 2

Objective: Write a program to draw various shapes using graphics.h.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
int main()
{
	int gd = DETECT,gm;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	// rectangle
	rectangle(20,300,180,200);
	outtextxy(60,330,"Rectangle");

	// circle
	circle(100,100,50);
	outtextxy(75,170,"Circle");

	// square
	moveto(300,60);
	lineto(390,60);
	lineto(390,150);
	lineto(300,150);
	lineto(300,60);
	outtextxy(325,180,"Square");

	// triangle
	moveto(350,200);
	lineto(300,300);
	lineto(400,300);
	lineto(350,200);
	outtextxy(320,320,"Triangle");

	getch();
	closegraph();
	return 0;
}
```

Output:



PRACTICAL – 3

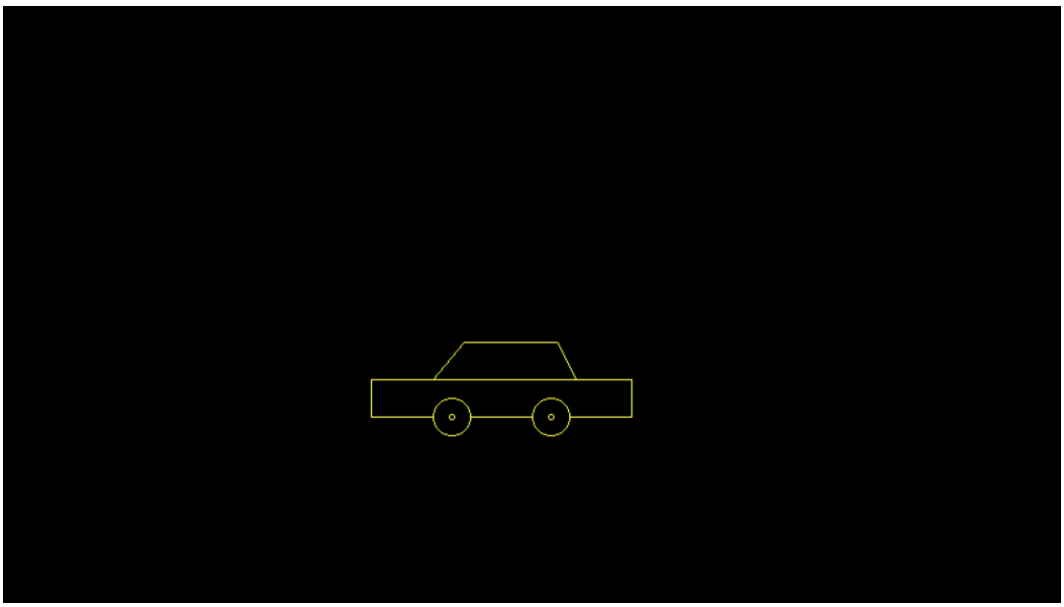
Objective: Write a program to draw moving car.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>

void draw_moving_car(void)
{
	int i,gd = DETECT, gm;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

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

		// These lines for bonnet and
		// body of car
		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);

		// For left wheel of car
		circle(65 + i, 330, 15);
		circle(65 + i, 330, 2);

		// For right wheel of car
		circle(145 + i, 330, 15);
		circle(145 + i, 330, 2);

		// Line left of left wheel
		line(0 + i, 330, 50 + i, 330);

		// Line middle of both wheel
		line(80 + i, 330, 130 + i, 330);

		// Line right of right wheel
		line(210 + i, 330, 160 + i, 330);

		delay(100);

		setcolor(BLACK);

		// Lines for bonnet and body of car
		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);

            // For left wheel of car
            circle(65 + i, 330, 15);
            circle(65 + i, 330, 2);

            // For right wheel of car
            circle(145 + i, 330, 15);
            circle(145 + i, 330, 2);

            // Line left of left wheel
            line(0 + i, 330, 50 + i, 330);

            // Line middle of both wheel
            line(80 + i, 330, 130 + i, 330);

            // Line right of right wheel
            line(210 + i, 330, 160 + i, 330);
      }
      getch();
      closegraph();
}

int main()
{
      draw_moving_car();
      return 0;
}
```

Output:



PRACTICAL – 4

Objective: Write a program to draw bouncing ball.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>

int main()
{
    int gd = DETECT,gm;
    int x, y = 0, j, t = 400, c = 1;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    setcolor(RED);
    setfillstyle(SOLID_FILL, RED);
    for (x = 40; x < 602; x++)
      {
      cleardevice();
      circle(x, y, 30);
      floodfill(x, y, RED);
      delay(50);
      if (y >= 400)
            {
          c = 0;
          t -= 20;
      }
      if (y <= (400 - t))
          c = 1;
      y = y + (c ? 15 : -15);
    }
    getch();
    closegraph();
    return 0;
}
```

Output:

PRACTICAL – 5

Objective: Write a program to draw a human body moving in the rain with the umbrella.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>

#define ScreenWidth getmaxx()
#define ScreenHeight getmaxy()
#define GroundY ScreenHeight*0.75

int ldisp=0;

void DrawManAndUmbrella(int x,int ldisp)
{
	//head
	circle(x,GroundY-90,10);
	line(x,GroundY-80,x,GroundY-30);

	//hand
	line(x,GroundY-70,x+10,GroundY-60);
	line(x,GroundY-65,x+10,GroundY-55);
	line(x+10,GroundY-60,x+20,GroundY-70);
	line(x+10,GroundY-55,x+20,GroundY-70);

	//legs
	line(x,GroundY-30,x+ldisp,GroundY);
	line(x,GroundY-30,x-ldisp,GroundY);

	//umbrella
	pieslice(x+20,GroundY-120,0,180,40);
	line(x+20,GroundY-120,x+20,GroundY-70);
}

void Rain(int x)
{
	int i,rx,ry;
	for(i=0;i<400;i++)
	{
		rx=rand() % ScreenWidth;
		ry=rand() % ScreenHeight;
	if(ry<GroundY-4)
	{
	if(ry<GroundY-120 || (ry>GroundY-120 && (rx<x-20 || rx>x+60)))
	line(rx,ry,rx+0.5,ry+4);
	}
```

```
      }
}

int main()
{
  int gd=DETECT,gm,x=0;
  initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

  while(!kbhit())
  {
   line(0,GroundY,ScreenWidth,GroundY);
   Rain(x);
   ldisp=(ldisp+2)%20;
   DrawManAndUmbrella(x,ldisp);
   delay(20);
   cleardevice();
   x=(x+2)%ScreenWidth;
   }
  getch();
  closegraph();
  return 0;
}
```

Output:

PRACTICAL – 6

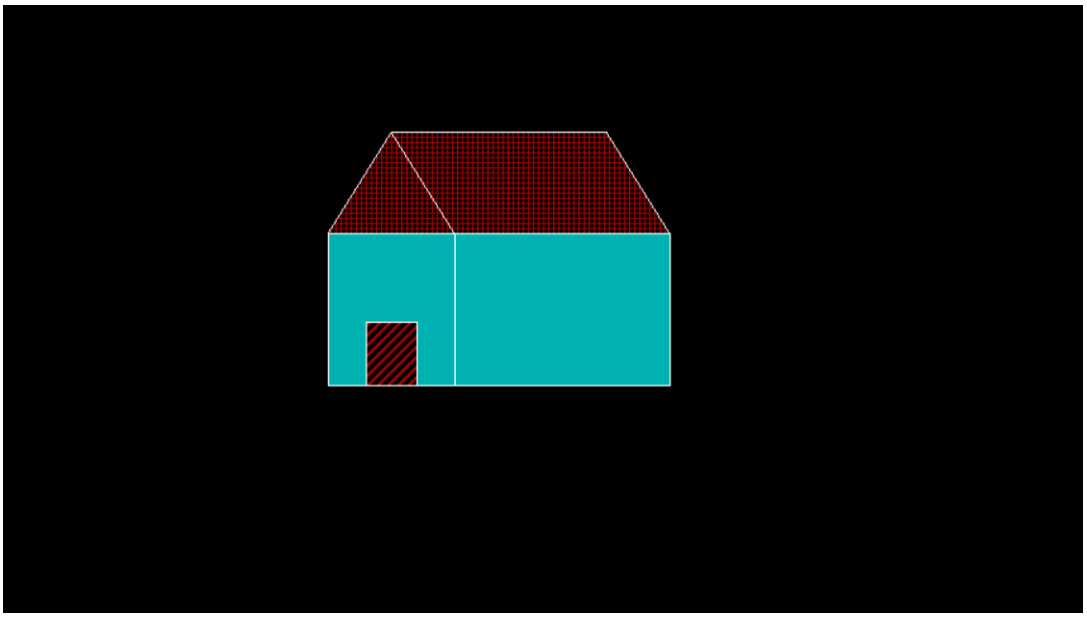
Objective: Write a program to draw a house using DDA algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<dos.h>
#include<graphics.h>
void ddaLine(int x1,int y1,int x2,int y2)
{
      int  dx, dy, length, i;
    float x, y, xinc, yinc;
      dx = x2 - x1;
    dy = y2 - y1;
    if (abs(dx) > abs(dy))
        length = abs(dx);
    else
        length = abs(dy);
    xinc = dx / (float)length;
    yinc = dy / (float)length;
    x = x1;
    y = y1;
    putpixel(x, y, 15);
    for (i = 0; i < length; i++)
    {
        putpixel(x, y,15);
        x = x + xinc;
        y = y + yinc;
        delay(10);
    }
}
int main()
{
      int gd=DETECT,gm;
      initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
      setcolor(WHITE);
      ddaLine(150,180,250,180);
      ddaLine(250,180,250,300);
      ddaLine(250,300,150,300);
      ddaLine(150,300,150,180);

      ddaLine(250,180,420,180);
      ddaLine(420,180,420,300);
      ddaLine(420,300,250,300);
      ddaLine(250,300,250,180);
```

```
        ddaLine(180,250,220,250);
        ddaLine(220,250,220,300);
        ddaLine(220,300,180,300);
        ddaLine(180,300,180,250);

        ddaLine(200,100,150,180);
       ddaLine(200,100,250,180);
       ddaLine(200,100,370,100);
       ddaLine(370,100,420,180);

       setfillstyle(SOLID_FILL, CYAN);
       floodfill(152, 182, WHITE);
       floodfill(252, 182, WHITE);
       setfillstyle(SLASH_FILL, RED);
       floodfill(182, 252, WHITE);
       setfillstyle(HATCH_FILL, RED);
       floodfill(200, 105, WHITE);
       floodfill(210, 105, WHITE);

       getch();
       closegraph();

       return 0;
}
```

Output:



PRACTICAL – 7

Objective: Write a program to draw a line using Bresenham's algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.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 gd=DETECT, gm, x0, y0, x1, y1;
      initgraph(&gd, &gm,"C:\\TURBOC3\\BGI");

      printf("Enter co-ordinates of first point: ");
      scanf("%d%d", &x0, &y0);

      printf("Enter co-ordinates of second point: ");
      scanf("%d%d", &x1, &y1);
      drawline(x0, y0, x1, y1);
      delay(10);
      getch();
```

```
        closegraph();
        return 0;
}
```

Output:

PRACTICAL – 8

Objective: Write a program to draw a circle using midpoint algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
void pixel(int xc,int yc,int x,int y)
{
	putpixel(xc+x,yc+y,WHITE);
	putpixel(xc+x,yc-y,WHITE);
	putpixel(xc-x,yc+y,WHITE);
	putpixel(xc-x,yc-y,WHITE);
	putpixel(xc+y,yc+x,WHITE);
	putpixel(xc+y,yc-x,WHITE);
	putpixel(xc-y,yc+x,WHITE);
	putpixel(xc-y,yc-x,WHITE);
}
int main()
{
	int gd=DETECT,gm,xc,yc,r,x,y,p;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

	printf("Enter center of circle :");
	scanf("%d%d",&xc,&yc);
	printf("Enter radius of circle :");
	scanf("%d",&r);

	x=0;
	y=r;
	p=1-r;
	pixel(xc,yc,x,y);

	while(x<y)
	{
		if(p<0)
		{
			x++;
			p=p+2*x+1;
		}
		else
		{
			x++;
			y--;
			p=p+2*(x-y)+1;
		}
		pixel(xc,yc,x,y);
```

```
        }
        delay(1000);
        closegraph();
        return 0;
}
```

Output:

PRACTICAL – 9

Objective: Write a program to draw a circle using Bresenham's algorithm.

Program:

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<graphics.h>

void drawCircle(int xc, int yc, int x, int y)
{
    putpixel(xc+x, yc+y, GREEN);
    putpixel(xc-x, yc+y, GREEN);
    putpixel(xc+x, yc-y, GREEN);
    putpixel(xc-x, yc-y, GREEN);
    putpixel(xc+y, yc+x, GREEN);
    putpixel(xc-y, yc+x, GREEN);
    putpixel(xc+y, yc-x, GREEN);
    putpixel(xc-y, yc-x, GREEN);
}

void circleBres(int xc, int yc, int r)
{
    int x = 0, y = r;
    int d = 3 - 2 * r;
    drawCircle(xc, yc, x, y);
    while (y >= x)
    {
        x++;

        if (d > 0)
        {
            y--;
            d = d + 4 * (x - y) + 10;
        }
        else
            d = d + 4 * x + 6;
        drawCircle(xc, yc, x, y);
        delay(40);
    }
}

int main()
{
    int xc,yc,r,gd=DETECT,gm;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    printf("Enter center of circle : ");
    scanf("%d%d",&xc,&yc);
    printf("\nEnter radius : ");
```

```
    scanf("%d",&r);

    circleBres(xc, yc, r);
    delay(1000);
    closegraph();
    getch();
    return 0;
}
```

Output:

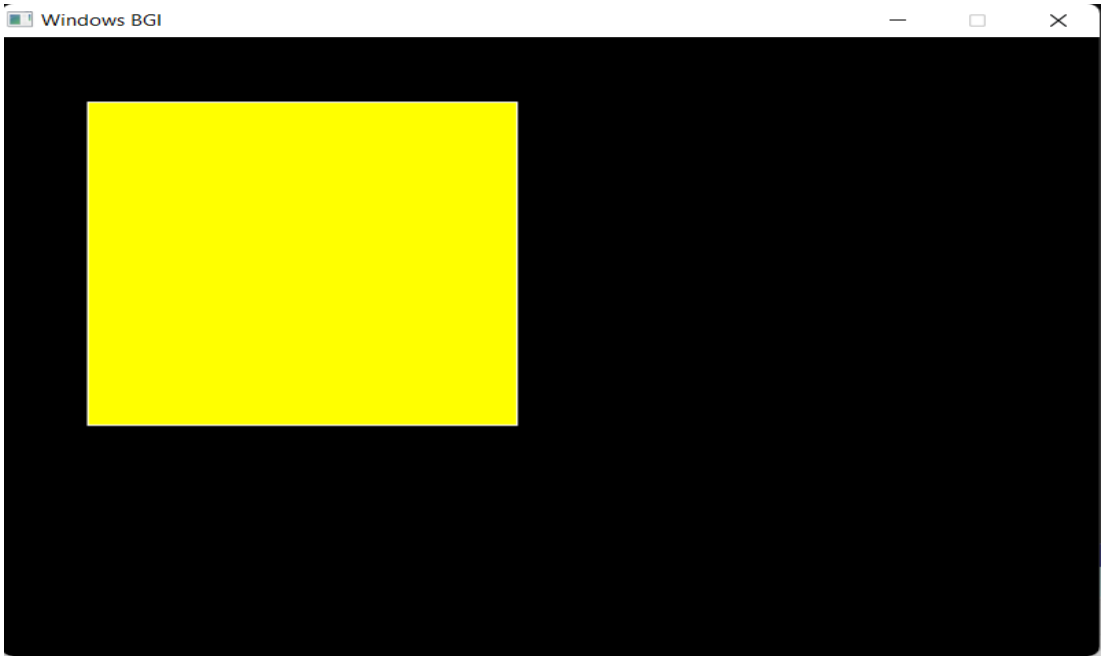
PRACTICAL – 10

Objective: Write a program to implement flood fill algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>

void floodFill(int x,int y,int fillcol,int oldcol)
{
	if(getpixel(x,y) == oldcol)
	{
	putpixel(x,y,fillcol);
	floodFill(x+1,y,fillcol,oldcol);
	floodFill(x-1,y,fillcol,oldcol);
	floodFill(x,y+1,fillcol,oldcol);
	floodFill(x,y-1,fillcol,oldcol);
	}
}

int main()
{
	int gm,gd=DETECT;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

	rectangle(50,50,300,300);
	floodFill(51,51,14,0);
	getch();
	closegraph();
	return 0;
}
```

Output:



PRACTICAL – 11

Objective: Write a program to implement boundary fill algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>

void boundaryfill(int x,int y,int fill,int boundary)
{
	int current;
	current = getpixel(x,y);
	if((current!=fill)&&(current!=boundary))
	{
		setcolor(fill);
		putpixel(x,y,fill);
		boundaryfill(x+1,y,fill,boundary);
		boundaryfill(x-1,y,fill,boundary);
		boundaryfill(x,y+1,fill,boundary);
		boundaryfill(x,y-1,fill,boundary);
	}
}

int main()
{
    int gd=DETECT,gm, x = 250, y = 200, radius = 80;
    initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
    circle(x, y, radius);
    boundaryfill(x, y, 4, 15);
    delay(1000);
    closegraph();
    return 0;
}
```

Output:

PRACTICAL – 12

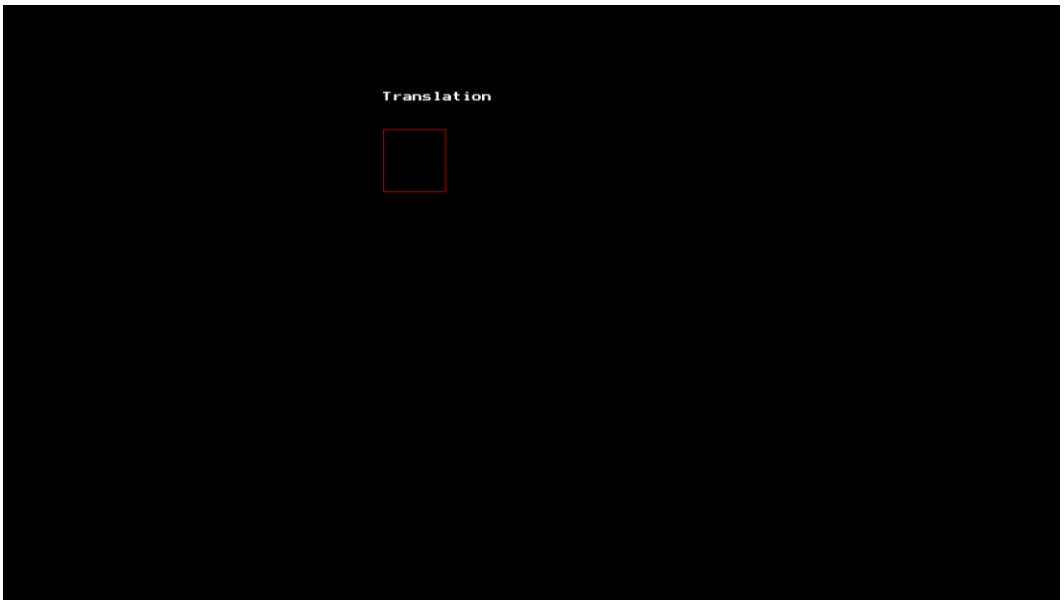
Objective: Write a program to implement 3 basic transformations.

Program:

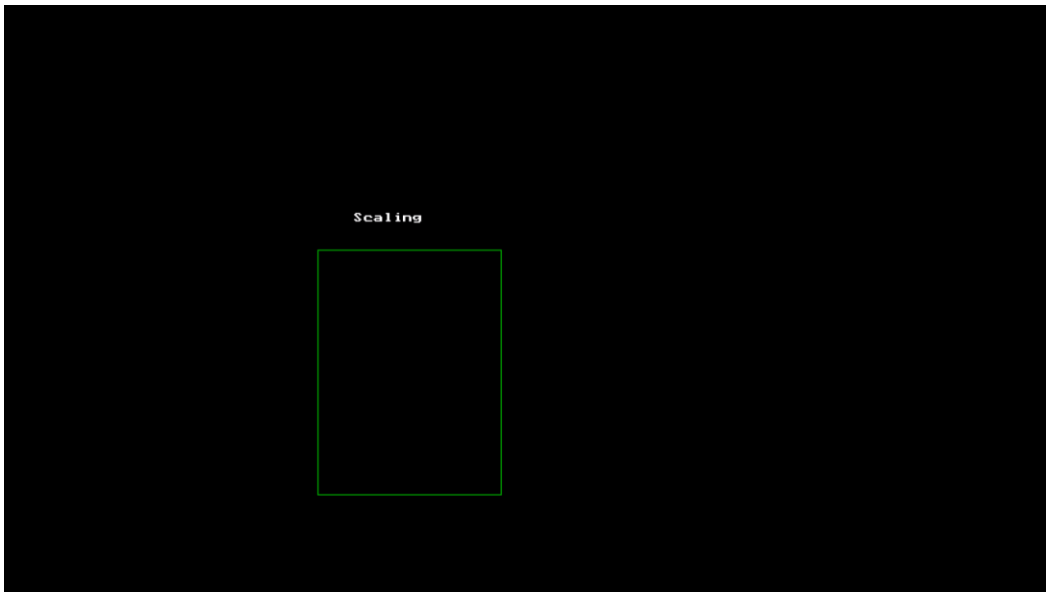
```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>
int main()
{
	int gd=DETECT,gm,tx,ty,sx,sy,x1,y1,x2,y2;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	printf("BASIC TRANSFORMATIONS : \n");
	setcolor(WHITE);
	rectangle(50,50,100,100);
	delay(1500);

	cleardevice();
	// Translation
	tx=150,ty=50;
	outtextxy(200,70,"Translation");
	setcolor(RED);
	rectangle(50+tx,50+ty,100+tx,100+ty);
	delay(1500);

	cleardevice();
	// Rotation
	double angle = (30*3.14)/180;
	setcolor(WHITE);
	outtextxy(50,20,"Rotation");
	setcolor(YELLOW);
	x1 = (int)(50+((50-50)*cos(angle)-(100-50)*sin(angle)));
	y1 = (int)(50+((50-50)*sin(angle)+(100-50)*cos(angle)));
	x2 = (int)(100+((100-100)*cos(angle)-(100-50)*sin(angle)));
	y2 = (int)(100+((100-100)*sin(angle)+(100-50)*cos(angle)));
	line(50,50,x1,y1);
	line(x1,y1,x2,y2);
	line(x2,y2,100,50);
	line(100,50,50,50);
	delay(1500);

	cleardevice();
	// Scaling
	sx=3,sy=4;
	setcolor(WHITE);
	outtextxy(180,170,"Scaling");
	setcolor(GREEN);
	rectangle(50*sx,50*sy,100*sx,100*sy);
	delay(1500);
```

```
        getch();
        closegraph();
        return 0;
}
```

Output:




Rotation


Scaling

PRACTICAL – 13

Objective: Write a program to implement Cohen Sutherland line clipping algorithm.

Program:

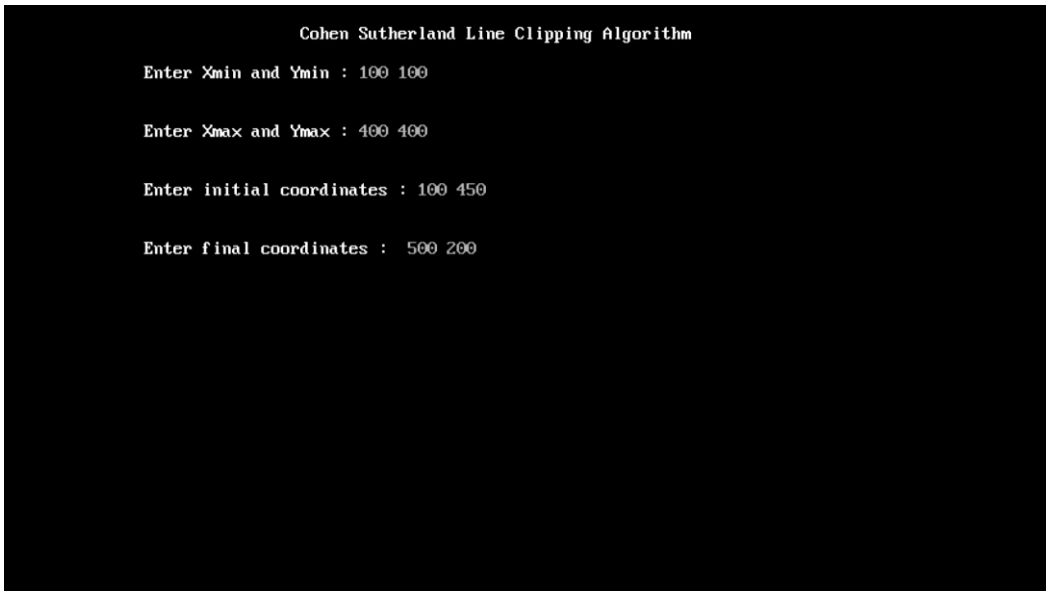
```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
int main()
{
	int rcode_begin[4]={0,0,0,0},rcode_end[4]={0,0,0,0},region_code[4];
	int W_xmax,W_ymax,W_xmin,W_ymin,flag=0;
	float slope;
	int x,y,x1,y1,i, xc,yc;
	int gd=DETECT,gm;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

	printf("\n\t\t Cohen Sutherland Line Clipping Algorithm");
	printf("\n\n Enter Xmin and Ymin : ");
	scanf("%d%d",&W_xmin,&W_ymin);
	printf("\n\n Enter Xmax and Ymax : ");
	scanf("%d%d",&W_xmax,&W_ymax);
	printf("\n\n Enter initial coordinates : ");
	scanf("%d%d",&x,&y);
	printf("\n\n Enter final coordinates :  ");
	scanf("%d%d",&x1,&y1);
	cleardevice();
	rectangle(W_xmin,W_ymin,W_xmax,W_ymax);
	line(x,y,x1,y1);

	if(y>W_ymax)  {
	rcode_begin[0]=1;	// Top
	flag=1 ;
	}

	if(y<W_ymin) {
	rcode_begin[1]=1;	// Bottom
	flag=1;
	}

	if(x>W_xmax)  {
	rcode_begin[2]=1;	// Right
	flag=1;
	}

	if(x<W_xmin)   {
```

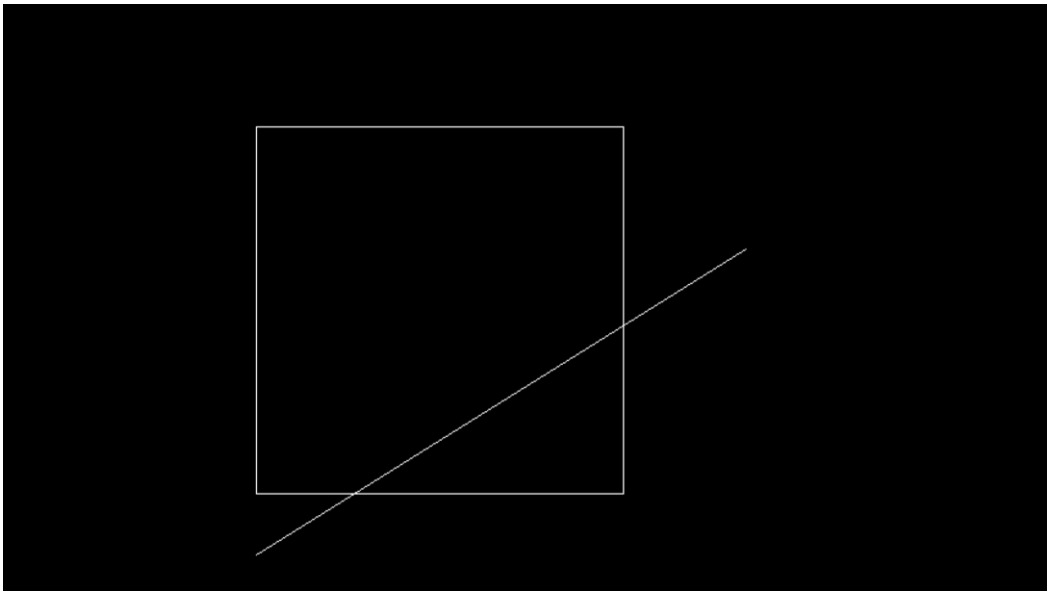
```
rcode_begin[3]=1;      //Left
flag=1;
}

if(y1>W_ymax){
rcode_end[0]=1;       // Top
flag=1;
}

if(y1<W_ymin) {
rcode_end[1]=1;       // Bottom
flag=1;
}

if(x1>W_xmax){
rcode_end[2]=1;       // Right
flag=1;
}

if(x1<W_xmin){
rcode_end[3]=1;       //Left
flag=1;
}

if(flag==0)
{
      printf("No need of clipping as it is already in window");
}

flag=1;
for(i=0;i<4;i++)
{
region_code[i]= rcode_begin[i] && rcode_end[i];
if(region_code[i]==1)
flag=0;
}

if(flag==0)
{
printf("\n Line is completely outside the window");
}

else
{
slope=(float)(y1-y)/(x1-x);

if(rcode_begin[2]==0 && rcode_begin[3]==1)   //left
{
y=y+(float) (W_xmin-x)*slope ;
```

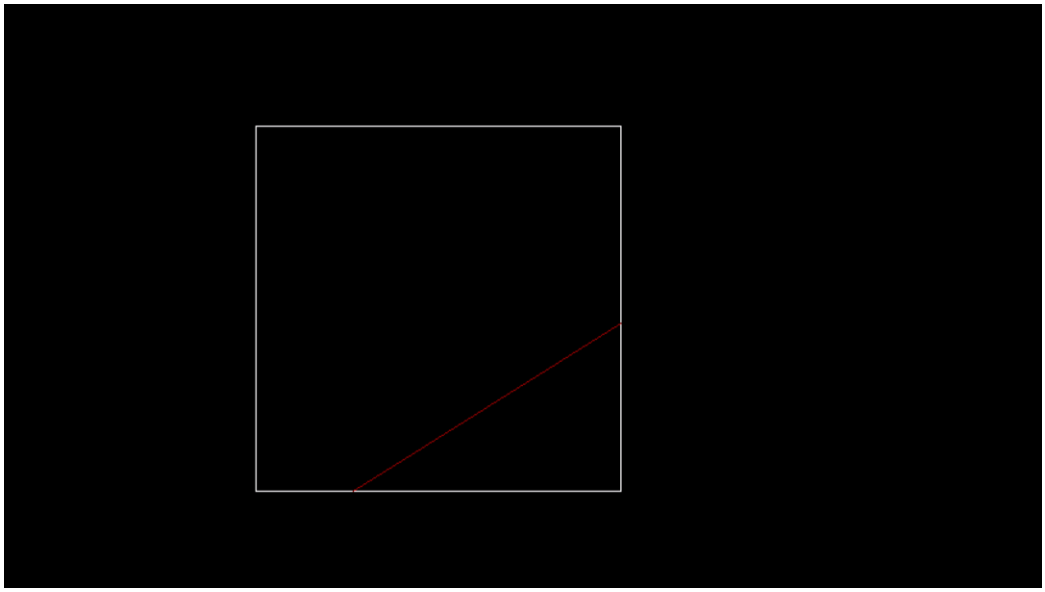
```
        x=W_xmin;
        }

        if(rcode_begin[2]==1 && rcode_begin[3]==0)   // right
        {
        y=y+(float) (W_xmax-x)*slope ;
        x=W_xmax;
        }

        if(rcode_begin[0]==1 && rcode_begin[1]==0)   // top
        {
        x=x+(float) (W_ymax-y)/slope ;
        y=W_ymax;
        }

        if(rcode_begin[0]==0 && rcode_begin[1]==1)   // bottom
        {
        x=x+(float) (W_ymin-y)/slope ;
        y=W_ymin;
        }

        // end points
        if(rcode_end[2]==0 && rcode_end[3]==1)      //left
        {
        y1=y1+(float) (W_xmin-x1)*slope ;
        x1=W_xmin;
        }

        if(rcode_end[2]==1 && rcode_end[3]==0)      // right
        {
        y1=y1+(float) (W_xmax-x1)*slope ;
        x1=W_xmax;
        }

        if(rcode_end[0]==1 && rcode_end[1]==0)      // top
        {
        x1=x1+(float) (W_ymax-y1)/slope ;
        y1=W_ymax;
        }

        if(rcode_end[0]==0 && rcode_end[1]==1)      // bottom
        {
        x1=x1+(float) (W_ymin-y1)/slope ;
        y1=W_ymin;
        }
}
        delay(1000);
        clearviewport();
        rectangle(W_xmin,W_ymin,W_xmax,W_ymax);
```

```
        setcolor(RED);
        line(x,y,x1,y1);
        getch();
        closegraph();
        return 0;
}
```

Output:







PRACTICAL – 14

Objective: Write a program to implement Sutherland Hodgman polygon clipping algorithm.

Program:

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>

#define round(a)((int)(a+0.5))

int k;
float xmin,ymin,xmax,ymax,arr[20],m;

void clipleft(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 cliptop(float x1, float y1, float x2, float y2)
{
      if(y2-y1)
      m=(x2-x1)/(y2-y1);
      else
```

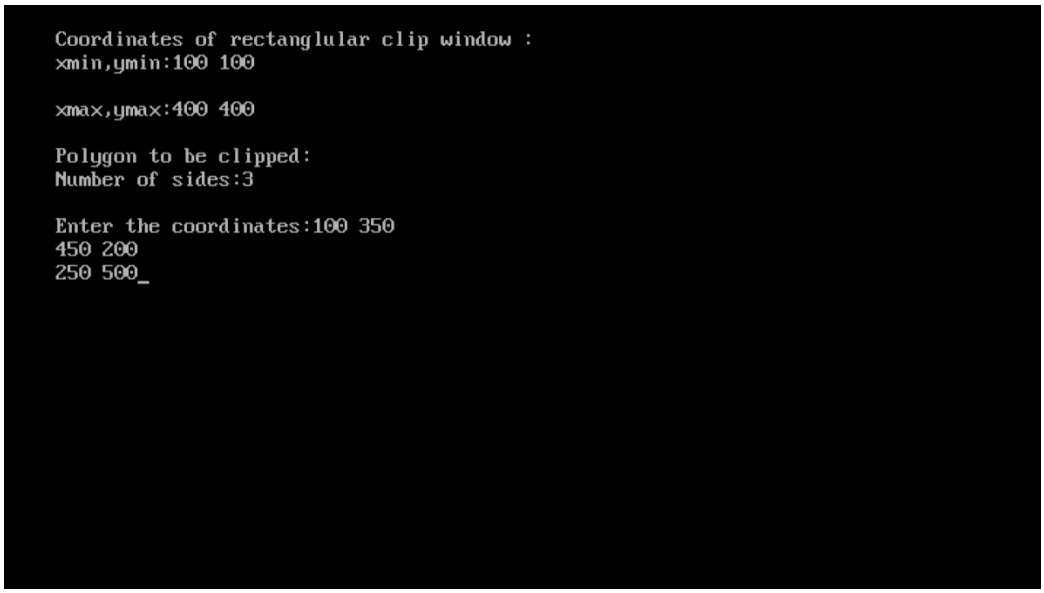
```
	m=100000;
	if(y1<=ymax && y2<=ymax)
	{
	arr[k]=x2;
	arr[k+1]=y2;
	k+=2;
	}

	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 clipright(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;
```

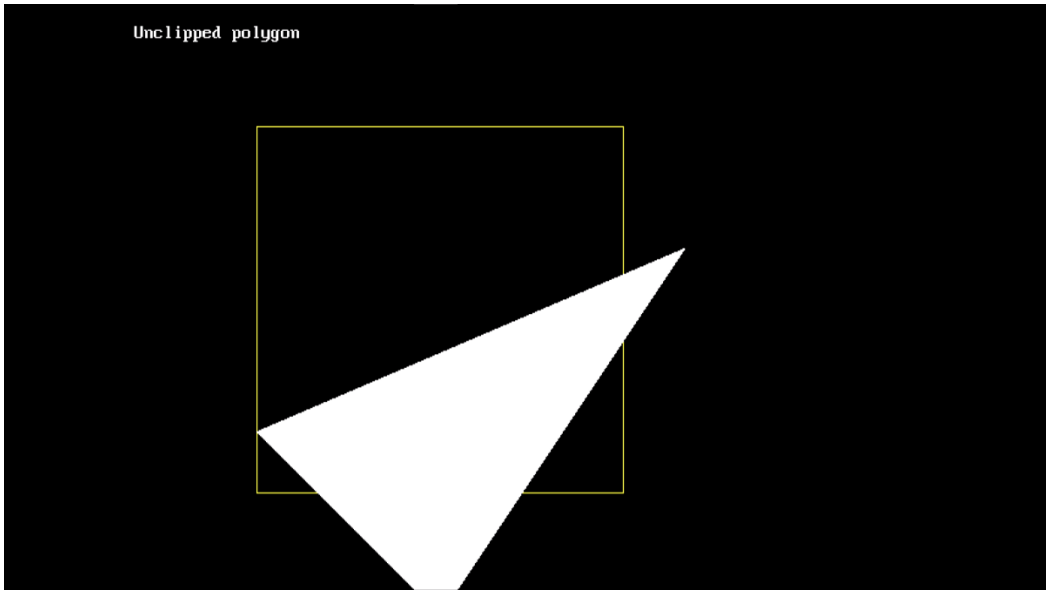
```
	}
}

void clipbottom(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 gd=DETECT,gm,n,poly[20];
	float xi,yi,xf,yf,polyy[20];
	clrscr();
	cout<<"\nCoordinates of rectangular clip window :\nxmin,ymin:";
	cin>>xmin>>ymin;
	cout<<"\nxmax,ymax:";
	cin>>xmax>>ymax;
	cout<<"\nPolygon to be clipped:\nNumber of sides:";
	cin>>n;
	cout<<"\nEnter 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]);
```

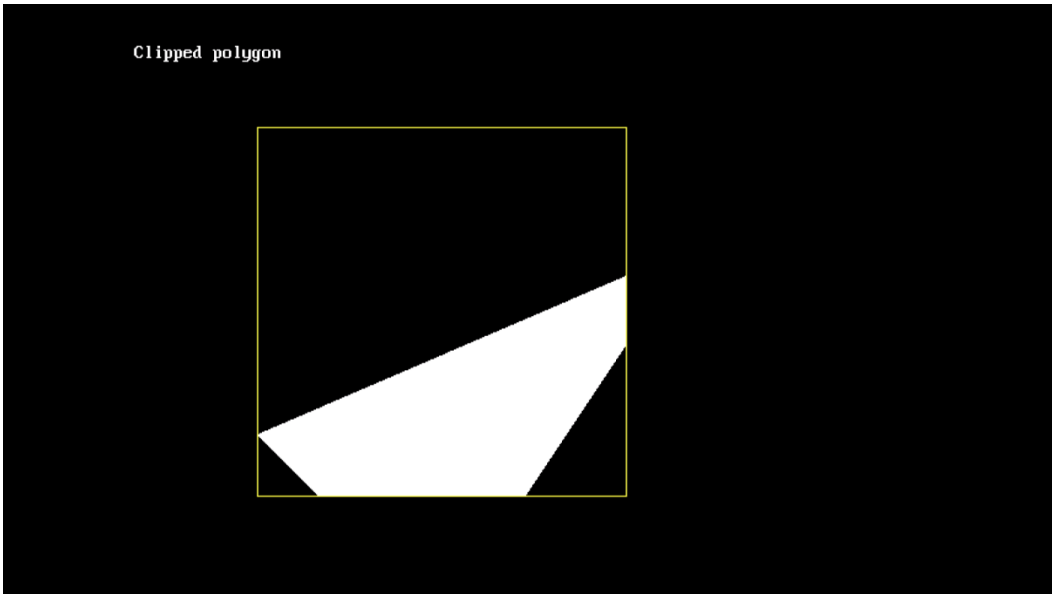
```
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
setcolor(YELLOW);
rectangle(xmin,ymax,ymax,ymin);
cout<<"\nUnclipped polygon";
setcolor(WHITE);
fillpoly(n,poly);
getch();
cleardevice();
k=0;

for(i=0;i<2*n;i+=2)
clipleft(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)
cliptop(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)
      clipright(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)
      clipbottom(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(YELLOW);

rectangle(xmin,ymax,xmax,ymin);
```

```
        cout<<"\nClipped polygon";
        getch();
        closegraph();

}
```

Output:






Clipped polygon