INTRODUCTION TO COMPUTER GRAPHICS.

Computer graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software. The interaction and understanding of computers and interpretation of data has been made easier because of computer graphics. Computer graphics development has made a significant impact on many types of media and has revolutionized animation, movies and the video game industry. The term computer graphics includes everything on computer almost everything that is not text or sound. Here in our lab at the program of computer graphics we think of computer graphics as drawing pictures on computer also called as *rendering*. In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called pixels. The pixel is the smallest addressable picture element. It is the smallest piece of the display screen which we can control. Computer graphics is the rapidly evolving field from the last two decades. Now computer graphics has touched not only the artists and engineers but also the common man in various ways.

NEED OF COMPUTER GRAPHICS IN COMPUTER SCIENCE

The principle reason to highlight computer in the design and development of instructional graphics is the computer's increasing range, versatility and flexibility of graphics design.

Also graphical representation will be more easier to understand than the manual graphics. Manual graphics cannot be edit easily and it is difficult to produce again and again so many copies of a same data in case of manual graphics therefore computer graphics are introduced which overcome all this. In computer graphics, one can produced as many copies as required, moreover neat and clean copies can also be obtained.

Computer graphics provide tools for producing pictures not only of concrete real world objects but also of abstract, synthetic objects such as mathematical survey in 4D and of data that have no inherent geometry. It has the ability to show moving pictures and thus it is possible to produce animations.

Also one more reason is that, Time is Money and In the 21st century people do not have much time to read huge number of pages so this problem is also solved by computer graphics.

Today we find that computer graphics routinely used in such diverse areas such as engineering, medical, business, industry, government, etc.

SOME EXAMPLES OF COMPUTER GRAPHICS

Here is a list of some the graphics tools available in the market:

1. **GIMP**: it is a very powerful image manipulation program, great for photo retouching and image creation. This program currently runs on Linux, Windows(XP and Vista) and some UNIX based systems.
2. **DARKTABLE**: It is an open source photography workflow software. With this amazing tool you can bring a level of quality to your photographs.
3. **ARTWEAVER**: This a freeware paint and drawing program for windows designed to simulate natural brush tools and effects for artists used to working on canvas. It supports most common file format.
4. **MARQUEED**: It is a free, simple tool to upload images and discuss them
5. **FONTFORGE**: It is not the easiest tool to use, but once you get the hang of it, you will be able to create fonts that perfectly meet your need.

SCREEN RESOLUTION OF COMPUTER MONITORS IN TERMS OF X-AXIS AND Y-AXIS

The display resolution of a computer monitor is the number of distinct pixels in each dimension that can be displayed. It can be ambiguous term especially as the displayed resolution is controlled by different factors in cathode ray tube, flat panel display, liquid crystal display.

X, y coordinates are respectively the horizontal and vertical addresses of any pixel or addressable point on computer display screen. The x coordinate is a given number of along the horizontal axis of display starting from the pixel on the extreme left of the screen. The y coordinate is the given number of pixel along the vertical axis of a display screen starting from the pixel at the top of the screen. On the each clickable area of an image map is specified as a pair of x and y coordinates in relative to the upper left hand corner of the image.

Resolution:

The resolution is calculated as the product of the number of pixels in the horizontal axis (rows) and the number in the vertical axis (columns) on a computer screen. Commonly available computer monitors have resolutions of 640 × 480, 800 × 600, or 1024 × 768.

In a CRT monitor, the horizontal resolution is limited by bandwidth and spot size, while the vertical resolution is limited by line spacing and spot size. In comparison, the resolution in an LCD monitor is determined by pixel pitch (the product of vertical and horizontal pixels) and ranges commonly from 1 megapixel (MP) to 5 MP. With ongoing refinements in technology, high-resolution monitors with resolutions as high as 9 MP and higher have started becoming available.

In CRT monitors, the resolution can be changed to match the frequency emitted by the video signal. This advantageous feature is referred to as 'multisync.' LCD monitors, in contrast, have a fixed resolution, which is termed the native resolution.Imagine lying down in the grass with your nose pressed deep into the thatch. Your field of vision would not be very large, and all you would see are a few big blades of grass, some grains of dirt, and maybe an ant or two. This is a 14-inch 640 x 480 monitor. Now, get up on your hands and knees, and your field of vision will improve considerably: you'll see a lot more grass. This is a 15-inch 800 x 640 monitor. For a 1280 x 1024 perspective (on a 19-inch monitor), stand up and look at the ground. Some monitors can handle higher resolutions such as 1600 x 1200 or even 1920 x 1440—somewhat akin to a view from up in a tree.

Monitors are measured in inches, diagonally from side to side (on the screen). However, there can be a big difference between that measurement and the actual viewable area. A 14-inch monitor only has a 13.2-inch viewable area, a 15-inch sees only 13.8 inches, and a 20-inch will give you 18.8 inches (viewing 85.7% more than a 15-inch screen).

A computer monitor is made of pixels (short for "picture element"). Monitor resolution is measured in pixels, width by height. 640 x 480 resolution means that the screen is 640 pixels wide by 480 tall, an aspect ratio of 4:3. With the exception of one resolution combination (1280 x 1024 uses a ratio of 5:4), all aspect ratios are the same.

Landscape *vs.* portrait:

Until recently, images in radiology were viewed in the landscape mode, a tradition set by the computer industry. This mode is clearly not optimal for use in diagnostic imaging. LCD monitors, being lightweight and thin, can be easily turned around and placed in a portrait mode, which corresponds with the image format of radiology images. A 17 × 14 image is optimally viewed in the 'portrait' rather than 'landscape' mode. The 'portrait' mode is particularly useful for chest and skeletal radiography and for mammography . By virtue of being lightweight and thin, LCD monitors can be placed in a portrait mode that corresponds with the 17×14 format of images obtained in chest and skeletal radiography.

GRAPHIC MODE AND TEXT MODE ICN C++

A graphic mode is the only way to safely communicate with the user.a graphic mode involves deciding how to allocate the memory of the video card for your program. On some platforms this means creating a virtual screen bigger than the physical resolution to do hardware scrolling or page flipping. Virtual screens can cause a lot of confusion, but they are really quite simple. Think of video memory as a rectangular piece of paper which is being viewed through a small hole (your monitor) in a bit of cardboard. Since the paper is bigger than the hole you can only see part of it at any one time, but by sliding the cardboard around you can alter which portion of the image is visible. You could just leave the hole in one position and ignore the parts of video memory that aren't visible, but you can get all sorts of useful effects by sliding the screen window around, or by drawing images in a hidden part of video memory and then flipping across to display them.

For example, you could select a 640x480 mode in which the monitor acts as a window onto a 1024x1024 virtual screen, and then move the visible screen around in this larger area (hardware scrolling). Initially, with the visible screen positioned at the top left corner of video memory.

BASIC GRAPHICS FUNCTION

1) INITGRAPH

• Initializes the graphics system.

Declaration

• Void far initgraph(int far *graphdriver)

Remarks

• To start the graphic system, you must first call initgraph.

• Initgraph initializes the graphic system by loading a graphics driver from disk (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.

Decleration

• Unsigned far getpixel(int x, int y)

• Void far putpixel(int x, int y, int color)

Remarks

• Getpixel gets the color of the pixel located at (x,y);

• Putpixel plots a point in the color defined at (x, y).

Return value

• Getpixel returns the color of the given pixel.

• Putpixel does not return.

3) CLOSE GRAPH

• Shuts down the graphic system.

Decleration

- Void far closegraph(void);

Remarks

- 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.

Return value

- None.

4) ARC, CIRCLE, PIESLICE

- arc draws a circular arc.
- Circle draws a circle
- Pieslice draws and fills a circular pieslice

Decleration

- 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);

Remarks

- Arc draws a circular arc in thecurrent 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, FILLELIPSE, SECTOR

- Ellipse draws an elliptical arc.
- Fillellipse draws and fills ellipse.
- Sector draws and fills an elliptical pie slice.

Decleration

- Void far ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius)
- Void far fillellipse(int x, int y, int xradius, int yradius)
- Void farsectoe(int x, int y, int stangle, int endangle, int xradius, int yradius)

Remarks

- Ellipse draws an elliptical arc in the current drawing color.

PRACTICAL NO.:-1

Familiarize yourself with creating and storing digital images using scanner and digital camera(compute the size of the image when store in the different formats) and convert the stored images from one format to another(BMP,GIF,JPEG,TIFF,PNG,etc) and analyse them.

There are so many image formats that it's so easy to get confused! File extensions like .jpeg, .bmp, .gif, and more can be seen after an image's file name. Most of us disregard it, thinking there is no significance regarding these image formats. These are all different and incompatible, though. These image formats have their own pros and cons. They were created for specific, yet different, purposes. The five most common image formats for the web and computer graphics are:

JPEG, GIF, BMP, TIFF and PNG.

1. **JPEG**

   JPEG is short for Joint Photographic Experts Group, and is the most popular among the image formats used on the web. JPEG files are very 'lousy', meaning so much information is lost from the original image when you save it in JPEG file. This is because JPEG discards most of the information to keep the image file size small; which means some degree of quality is also lost. If you take a closer look, the JPEG image is not as sharp as the original image. The colors are paler and the lines are less defined and the picture is noisier. If you zoom in there are JPEG artifacts like any other JPEG files.

Almost every digital camera can shoot and save in the JPEG format. JPEG is very web friendly because the file is smaller, which means it takes up less room, and requires less time to transfer to a sites. Moreover it is less grainy then GIF, the old king of the internet roost. Since 1994, JPEG has been considered the standard.

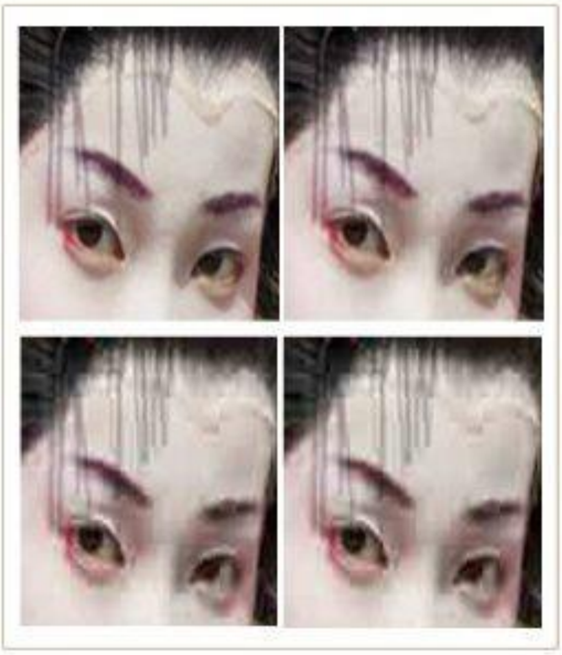
Pros of JPEG:

- 24-bit color, with up to 16 million colors
- Rich colors, great for photographs that needs fine attention to color detail
- Most used and most widely accepted image format
- Compatible in most OS (Mac, PC, Linux)

Cons of JPEG:

- They tend to discard a lot of data
- After compression, JPEG tends to create artifacts
- Cannot be animated
- Does not support transparency

SAMPLE IMAGE OF COMPRESSION OF JPEG



2. GIF

GIF, short for Graphics Interchange Format, is limited to the 8 bit palette with only 256 colors. GIF is still a popular image format on the internet because image size is relatively small compared to other image compression types.GIF compresses images in two ways: first, by reducing the number of colors in rich color images, thus reducing the number of bits per pixel. Second, GIF replaces multiple occurring patterns (large patterns) into one. So instead of storing five kinds of blue, it stores only one blue.GIF is most suitable for graphics, diagrams, cartoons and logos with relatively few colors. GIF is still the chosen format for animation effects.Compared to JPEG, it is lossless and thus more effective with compressing images with a single color, but pales in detailed or dithered pictures. In other words, GIF is lossless for images with 256 colors and below. So for a full color image, it may lose up to 99.998% of its colors.One edge of the GIF image format is the interlacing feature, giving the illusion of fast loading graphics. When it loads in a browser, the GIF first appears to be blurry and fuzzy, but as soon as more data is downloaded, the image becomes more defined until all the date has been downloaded.

Pros of GIF

- Can support transparency
- Can do small animation effects
- 'Lossless' quality–they contain the same amount of quality as the original, except of course it now only has 256 colors
- Great for images with limited colors, or with flat regions of color

Cons of GIF:

- Only supports 256 colors
- It's the oldest format in the web, having existed since 1989. It hasn't been updated since, and sometimes, the file size is larger than PNG.

3. BMP

The Windows Bitmap or BMP files are image files within the Microsoft Windows operating system. In fact, it was at one point one of the few image formats. These files are large and uncompressed, but the images are rich in color, high in quality, simple and compatible in all Windows OS and programs. BMP files are also called raster or paint images.BMP files are made of millions and millions of dots called 'pixels', with different colors and arrangements to come up with an image or pattern. It might an 8-bit, 16-bit or 24-bit image. Thus when you make a BMP image larger or smaller, you are making the individual pixels larger, and thus making the shapes look fuzzy and jagged.BMP files are not great and not very popular. Being oversized, bitmap files are not what you call 'web friendly', nor are they compatible in all platforms and they do not scale well.

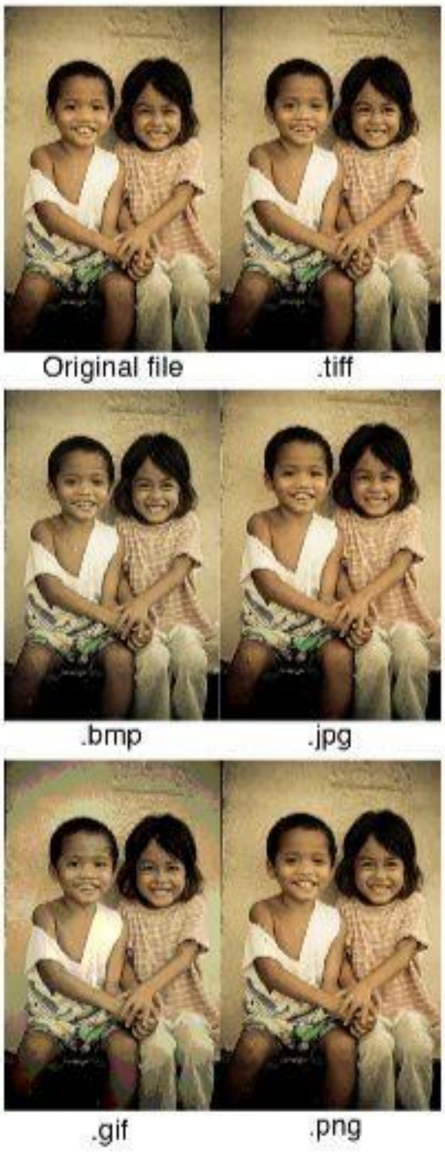
Pros of BMP:

- Works well with most Windows programs and OS, you can use it as a Windows wallpaper

Cons of BMP:

- Does not scale or compress well
- Again, very huge image files making it not web friendly
- No real advantage over other image formats

SAMPLE IMAGE OF BMP



HOW TO SCAN A DOCUMENT.

1. **Open the scanner.** Once you have your scanner and have hooked it up to your computer, you can begin scanning your documents. Open the lid of the scanner, in much the same way you'd open a copy machine, in order to reveal the glass scanning surface.
2. **Place the items into the scanning tray or glass.** Place the side of the document that you want scanned down onto the glass. This will allow the camera below to see the item. Place the document the appropriate place; the space for a standard-size paper, as well as several other common sizes, will be marked along the edges of the glass. Any part of the document which does not fit on the glass will not be scanned. Once you have placed the item, close the lid.
3. **Open your scanning software.** Now that the item is place, open your scanning

software on your computer. This can be accessed in a variety of ways. Try to install the software such that an icon is always readily accessible.

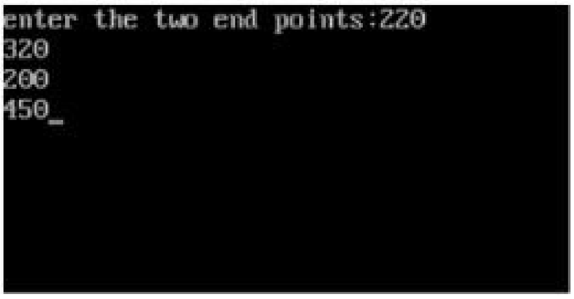
4. **Adjust the scan settings.** Adjust the settings as necessary for the document that you plan to scan. Text documents can often be scanned in black and white, while some scanning software may make allowances for scanning photos vs scanning posters. The software varies widely across brands and devices, so you will have to experiment.
    - Often the settings you will want to change will be relating to the quality of the scan. Higher quality scans will produce much larger files. Scan only to the quality you need. Images may need to be high quality where text can generally be scanned at a very low quality. Some software will have preset settings for this.
5. **Select "Scan".** Once the settings have been adjusted, you will need to tell the software or the scanner to begin scanning the document. There may be a "Scan" button in the software or you may need to press a "Scan" button on the machine itself. Read your user manual if you become too confused.
6. **Make further adjustments, if necessary.** Once the item is scanned, your software will usually give you the option to make further adjustments, such as changing the orientation of the image. Make whatever adjustments you feel are necessary and then tell the software to save the file.
    - Be aware that scanned images are usually quite large, especially if you have chosen to scan at high settings. Change the settings and rescan the image if the file size is simply too large for your purposes.
    - Pay attention to the file format that the document saves in. You may wish to choose something other than the default.

PRACTICAL:-2

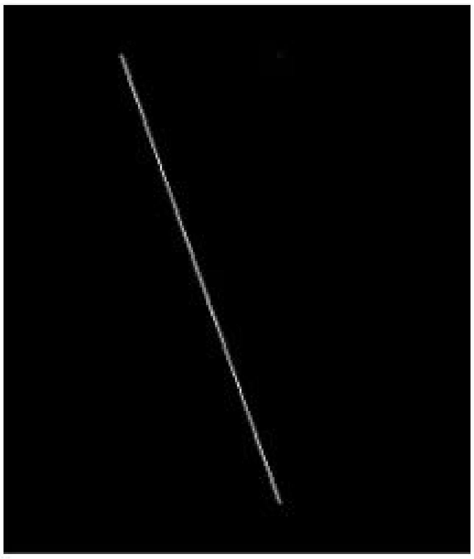
Implement bresenham's line algorithm. Also provide Provision to change attributes of graph primitives such as stippling (Dotted and Dashed pattern), colors .Implement Bresenham circle algorithm also provide to change attributes of graph primitives such as stippling(Dotted and Dashed patteren)and colors.

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int x1,x2,y1,y2;
int gd=DETECT,gm;
void linebres(int,int,int,int);
printf("enter the two end points:");
scanf("%d%d%d%d",&x1,&x2,&y1,&y2);
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
cleardevice();
linebres(x1,y1,x2,y2);
getch();
line(x1,y1,x2,y2);
getch();
closegraph();
}
void linebres(int x1,int y1,int x2,int y2)
{
int dx=abs(x1-x2),dy=abs(y1-y2);
int p,x,y,i,xend,yend;
if(dx!=0)
{
p=2*dy-dx;
if(x1>x2)
{
x=x2;
y=y2;
xend=x1;
}
else
{
x=x1;
y=y1;
xend=x2;
}
putpixel(x,y,2);
for(i=x;i<xend;i++)
{x+=1;
if(p<0)
p+=2*dy;
else
p+=2*(dy-dx);}
putpixel(x,y,2);}
else
{
p=2*dx-dy;
if(y1>y2)
{
x=x2;
y=y2;
yend=y2;
}
putpixel(x,y,2);
for(i=y;i<yend;i++)
{
y+=1;
if(p<0)
p+=2*dx;
else
{p+=1;
p+=2*(dx-dy);
}
putpixel(x,y,2);
}
}
}
```

OUTPUT:-



```
enter the two end points:220
320
200
450_
```



PRACTICAL NO:-3

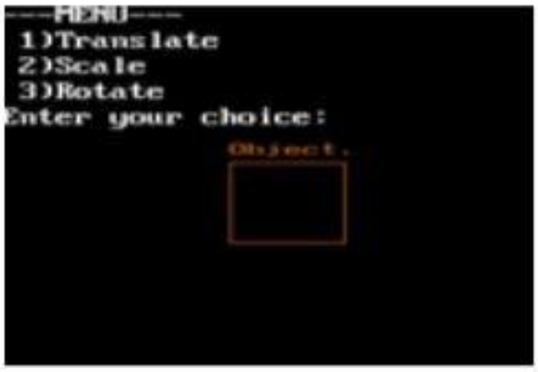
Implement 2-D transformation with translation, scaling, rotation, reflection, Shearing and scaling.

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void translate();

void scale();

void rotate();

void main()

{

intch;

intgd=DETECT,gm;

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

setcolor(6);

outtextxy (100,88,"Object.");

rectangle(100,150,150,100);

printf("---MENU---");

printf("\n 1)Translate\n 2)Scale\n 3)Rotate");
printf("\nEnter your choice: ");
scanf("%d",&ch);
cleardevice();
switch(ch)
{
case 1: translate();
break;
case 2: scale();
break;
case 3: rotate();
break;
default: printf("you have enterd wrong choice");
break;
}
getch();
closegraph();
}

void translate()
{
inttx,ty;
setcolor(2);
outtextxy(240,10,"TRANSLATION");
outtextxy(238,20,"----------");
printf("\nEntertx: ");
```

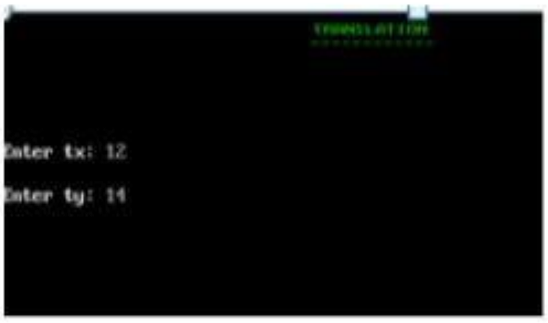
```
scanf("%d",&tx);
printf("\nEnterty: ");
scanf("%d",&ty);
cleardevice();
rectangle(100,150,150,100);
printf("\nAfter Translation");
rectangle(100+tx,150+ty,150+tx,100+ty);
}

void scale()
{
intsx,sy;
setcolor(2);
outtextxy(240,10,"SCALING");
outtextxy(238,20,"-------");
printf("\nEntersx: ");
scanf("%d",&sx);
printf("\nEntersy: ");
scanf("%d",&sy);
cleardevice();
rectangle(100,150,150,100);
printf("\nAfter Scaling");
rectangle(100*sx,150*sy,150*sx,100*sy);
}
void rotate()
```

```
{
float theta;
int x1,x2,x3,x4;
int y1,y2,y3,y4;
int ax1,ax2,ax3,ax4,ay1,ay2,ay3,ay4;
intrefx,refy;
printf("\nEnter the angle for rotation: ");
scanf("%f",&theta);
theta=theta*(3.14/180);
cleardevice();
setcolor(2);
outtextxy(240,10,"ROTATE");
outtextxy(238,20,"------");
refx=100;
refy=100;
x1=100;
y1=100;
x2=150;
y2=100;
x3=150;
y3=150;
x4=100;
y4=150;

ax1=refy+(x1-refx)*cos(theta)-(y1-refy)*sin(theta);
ay1=refy+(x1-refx)*sin(theta)+(y1-refy)*cos(theta);
ax2=refy+(x2-refx)*cos(theta)-(y2-refy)*sin(theta);
ay2=refy+(x2-refx)*sin(theta)+(y2-refy)*cos(theta);
ax3=refy+(x3-refx)*cos(theta)-(y3-refy)*sin(theta);
ay3=refy+(x3-refx)*sin(theta)+(y3-refy)*cos(theta);
ax4=refy+(x4-refx)*cos(theta)-(y4-refy)*sin(theta);
ay4=refy+(x4-refx)*sin(theta)+(y4-refy)*cos(theta);
rectangle(100,150,150,100);
line(ax1,ay1,ax2,ay2);
line(ax2,ay2,ax3,ay3);
line(ax3,ay3,ax4,ay4);
line(ax4,ay4,ax1,ay1);
}
```

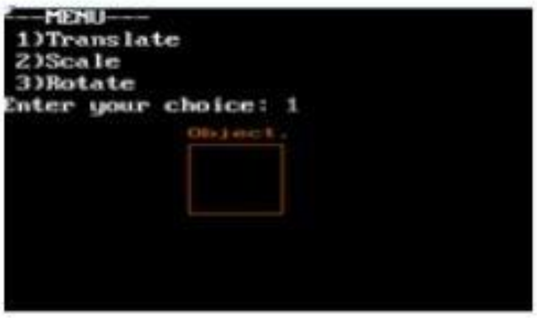
OUTPUT:-



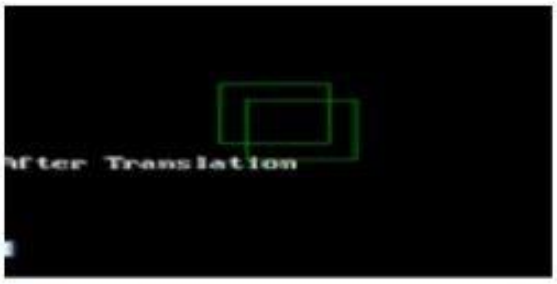
```
----MENU----
 1)Translate
 2)Scale
 3)Rotate
Enter your choice:
```



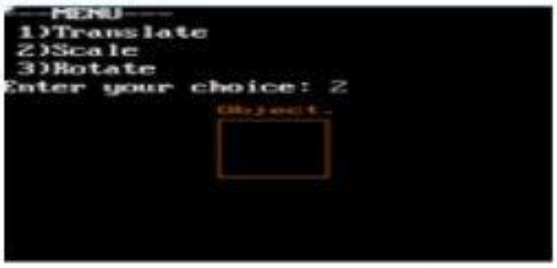
```
Enter tx: 12

Enter ty: 14
```



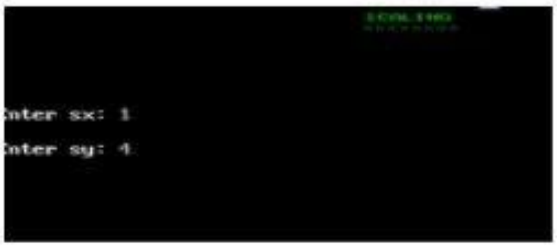
```
---MENU---
 1)Translate
 2)Scale
 3)Rotate
Enter your choice: 1
```



```
After Translation
```



```
---MENU---
 1)Translate
 2)Scale
 3)Rotate
Enter your choice: 2
```



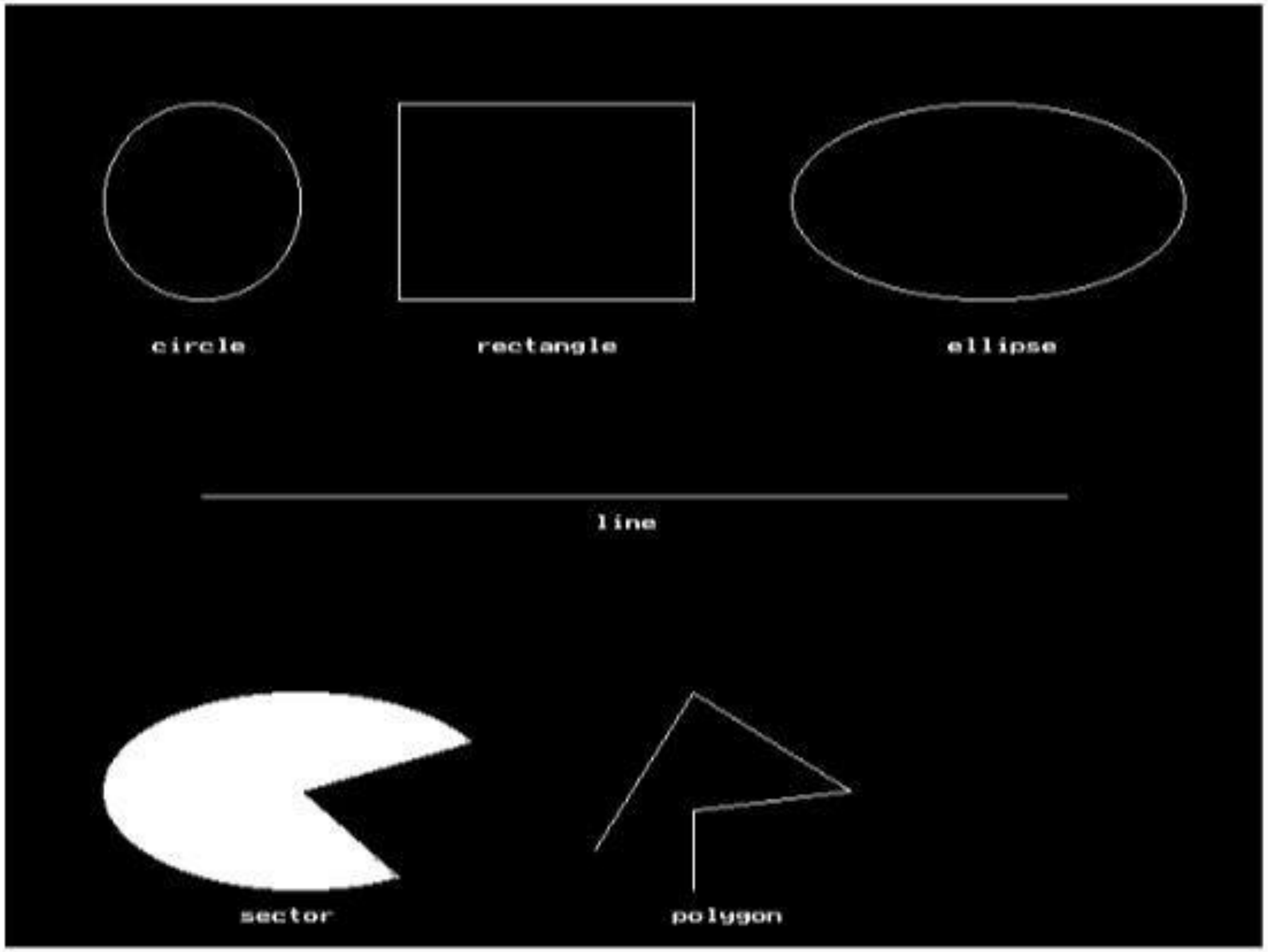
```
                                        SCALING
                                        -------


nter sx: 1

nter sy: 4
```

PRACTICAL NO:-4

Write a program to Implement tweening procedure for animation with key frames having equal different no. of edges.

```
#include<graphics.h>

#include<conio.h>

void main()

{

intgd=DETECT, gm;

int poly[12]={350,450, 350,410, 430,400, 350,350, 300,430, 350,450 };

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

 circle(100,100,50);

outtextxy(75,170, "Circle");

   rectangle(200,50,350,150);

outtextxy(240, 170, "Rectangle");

   ellipse(500, 100,0,360, 100,50);
outtextxy(480, 170, "Ellipse");
   line(100,250,540,250);
outtextxy(300,260,"Line");

   sector(150, 400, 30, 300, 100,50);
outtextxy(120, 460, "Sector");
drawpoly(6, poly);
outtextxy(340, 460, "Polygon");
getch();
closegraph();
}
```

OUTPUT:-

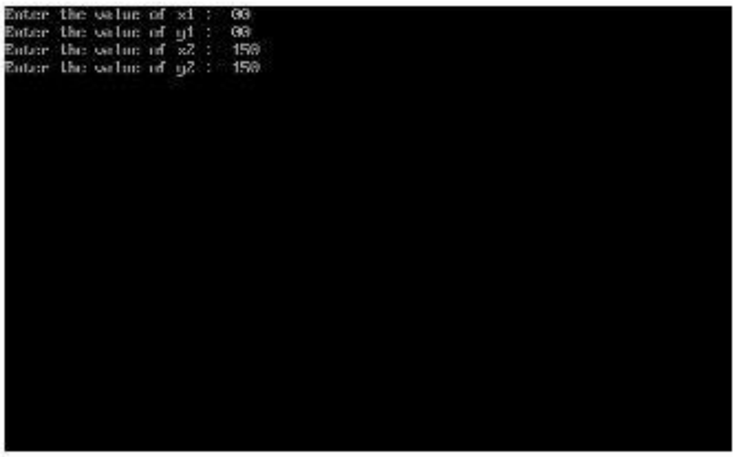

circle
rectangle
ellipse
line
sector
polygon

Practical NO:-5

Write a Program for 2 –D line drawing as Raster graphics display.

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
 main()
{
float x,y,x1,y1,x2,y2,dx,dy,length;
int i,gd,gm;
clrscr();
printf("Enter the value of x1 :\t ");
scanf("%f",&x1);
printf("Enter the value of y1 :\t ");
scanf("%f",&y1);
printf("Enter the value of x2 :\t ");
scanf("%f",&x2);
printf("Enter the value of y2 :\t ");
scanf("%f",&y2);
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
dx=abs(x2-x1);
dy=abs(y2-y1);
if(dx >= dy)
{
length = dx;
}
else
{
length = dy;
}
dx = (x2-x1)/length;
dy = (y2-y1)/length;
x = x1 + 0.5;
y = y1 + 0.5;
i = 1;


while(i <= length)
{
putpixel(x,y,15);
x = x + dx;
y = y + dy;
i = i + 1;
delay (100);
}
getch();
closegraph();
}
```

OUTPUT:-



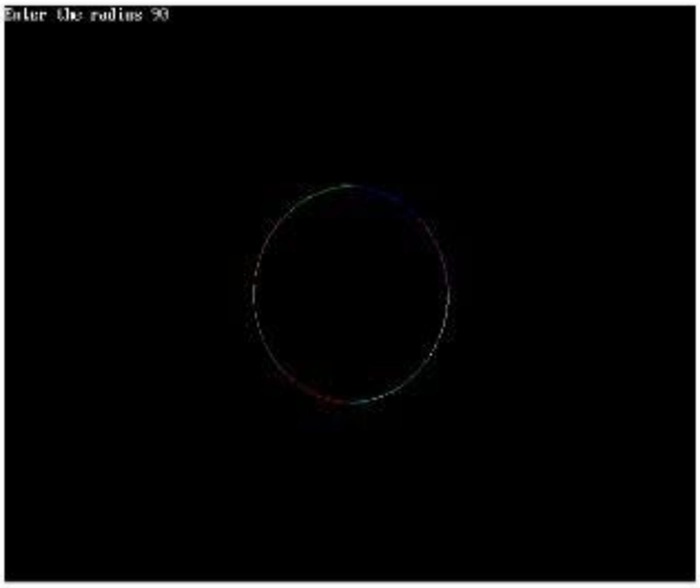
```
Enter the value of x1 :  00
Enter the value of y1 :  00
Enter the value of x2 :  150
Enter the value of y2 :  150
```

Practical NO:-6

Write a program for 2-D circle drawing as Raster graphics display.

```
# include<stdio.h>
# include<conio.h>
# include<graphics.h>
# include<math.h>
void main()
{int gd=DETECT,gm;
int r,x,y,p,xc=320,yc=240;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
cleardevice();
printf("Enter the radius ");
scanf("%d",&r);
x=0;
y=r;
putpixel(xc+x,yc-y,1);
p=3-(2*r);
for(x=0;x<=y;x++)
{ if (p<0)
{
y=y;
p=(p+(4*x)+6);
}
else
{y=y-1;
p=p+((4*(x-y)+10));
}
putpixel(xc+x,yc-y,1);
putpixel(xc-x,yc-y,2);
putpixel(xc+x,yc+y,3);
putpixel(xc-x,yc+y,4);
putpixel(xc+y,yc-x,5);
putpixel(xc-y,yc-x,6);
putpixel(xc+y,yc+x,7);
putpixel(xc-y,yc+x,8);
}
getch();
closegraph();
}
```

OUTPUT:-



```
Enter the radius 90
```

Practical No:-7

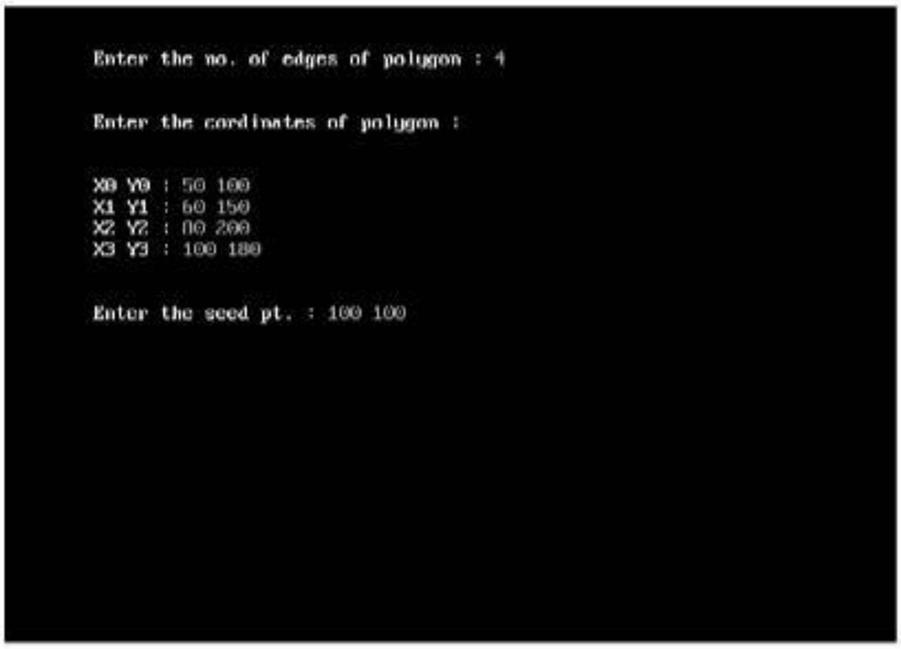
Write a program for 2-D polygon filling as Raster graphics display.

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

void fill_right(x,y)
int x , y ;
{
if(getpixel(x,y) == 0)
{
putpixel(x,y,RED);
fill_right(++x,y);
x = x - 1 ;
fill_right(x,y-1);
fill_right(x,y+1);

}
}

void fill_left(x,y)
int x , y ;
{
if(getpixel(x,y) == 0)
{
putpixel(x,y,RED);

fill_left(–x,y);
x = x + 1 ;
fill_left(x,y-1);
fill_left(x,y+1);

}
}
```

```
void main()
{
int x , y ,a[10][10];
int gd, gm ,n,i;

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

printf("\n\n\tEnter the no. of edges of polygon : ");
scanf("%d",&n);
printf("\n\n\tEnter the cordinates of polygon :\n\n\n ");

for(i=0;i<n;i++)
{
printf("\tX%d Y%d : ",i,i);
scanf("%d %d",&a[i][0],&a[i][1]);
}

a[n][0]=a[0][0];
a[n][1]=a[0][1];

printf("\n\n\tEnter the seed pt. : ");
scanf("%d%d",&x,&y);


cleardevice();
setcolor(WHITE);

for(i=0;i<n;i++) /*- draw poly -*/
{
line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
}

fill_right(x,y);
fill_left(x-1,y);

getch();
}
/*Enter the number of edges of polygon 4
X0 Y0 = 50 50
X1 Y1 = 200 50
```

```
X2 Y2 = 200 300
X3 Y3 = 50 300
Enter the seed point 100 100*/
```

OUTPUT:-



```
Enter the no. of edges of polygon : 4


Enter the cordinates of polygon :


X0 Y0 : 50 100
X1 Y1 : 60 150
X2 Y2 : 80 200
X3 Y3 : 100 180


Enter the seed pt. : 100 100
```

Practical NO:-8

Write a program for line clipping

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>
#include<process.h>
int pixels[2][4];
float xn1,xn2,yn1,yn2,x3,y3,m;
int xmin,ymin,xmax,ymax,x1,y1,x2,y2;
intchoice,ed[20],num;
voidsu_co(int x1,int y1,int x2,int y2,int xmin,intymin,intxmax,intymax)
{
inti,j,fl;
for(i=0;i<2;i++)
for(j=0;j<4;j++)
pixels[i][j]=0;
if(y1>ymax)
pixels[0][0]=1;
if(y1<ymin)
pixels[0][1]=1;
if(x1>xmax)
pixels[0][2]=1;
if(x1<xmin)
```

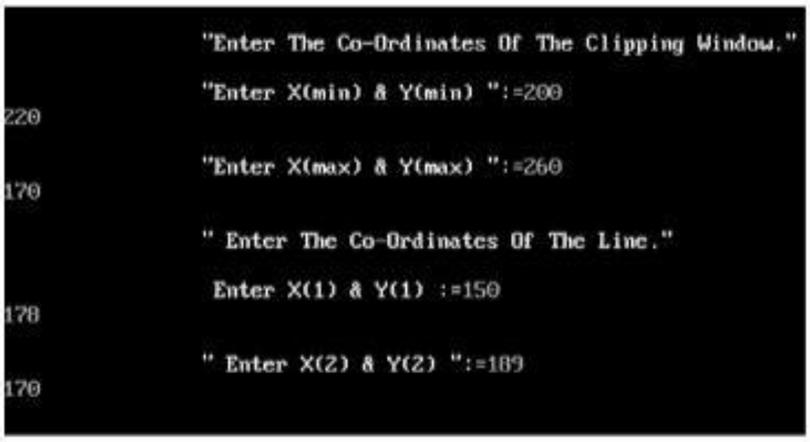
```
pixels[0][3]=1;
if(y2>ymax)
pixels[1][0]=1;
if(y2<ymin)
pixels[1][1]=1;
if(x2>xmax)
pixels[1][2]=1;
if(x2<xmin)
pixels[1][3]=1;
for(j=0;j<4;j++)
{
if(pixels[0][j]==0&&pixels[1][j]==0)
continue;
if(pixels[0][j]==1&&pixels[1][j]==1)
{
fl=3;
break;
}
fl=2;
}
switch(fl)
{
case 1:
line(320+x1,240-y1,320+x2,240-y2);
break;
```

```
case 3:
printf("\n\n\a\"Line Is Not Visible....");
break;
case 2:
m=(y2-y1)/(x2-x1);
xn1=x1;
yn1=y1;
xn2=x2;
yn2=y2;
if(pixels[0][0]==1)
{
xn1=x1+(ymax-y1)/m;
yn1=ymax;
}
if(pixels[0][1]==1)
{
xn1=x1+(ymin-y1)/m;
yn1=ymin;
}
if(pixels[0][2]==1)
{
yn1=y1+(xmax-x1)*m;
xn1=xmax;
}
if(pixels[0][3]==1)
```

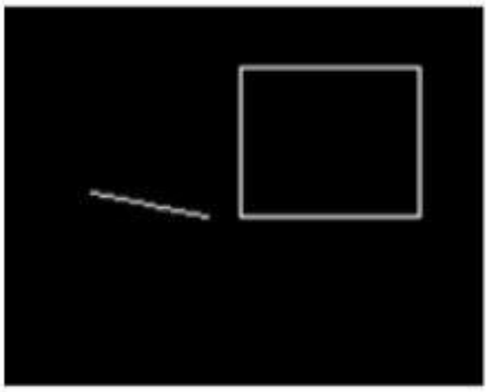
```
{
yn1=y1+(xmin-x1)*m;
xn1=xmin;
}
if(pixels[1][0]==1)
{
xn2=x2+(ymax-y2)/m;
yn2=ymax;
}
if(pixels[1][1]==1)
{
xn2=x2+(ymin-y2)/m;
yn2=ymin;
}
if(pixels[1][2]==1)
{ yn2=y2+(xmax-x2)*m;
xn2=xmax;
}
if(pixels[1][3]==1)
{
yn2=y2+(xmin-x2)*m;
xn2=xmin;
}
line(320+xn1,240-yn1,320+xn2,240-yn2);
break;
```

```
scanf("%d %d",&xmin,&ymin);
printf("\n\t\t\"Enter X(max) & Y(max) \":=");
scanf("%d %d",&xmax,&ymax);
printf("\n\t\t\" Enter The Co-Ordinates Of The Line.\"");
printf("\n\n\t\t Enter X(1) & Y(1) \:=");
scanf("%d %d",&x1,&y1);
printf("\n\t\t\" Enter X(2) & Y(2) \":=");
scanf("%d %d",&x2,&y2);
clrscr();
cohen();
}
```

OUTPUT:-



```
                "Enter The Co-Ordinates Of The Clipping Window."
                "Enter X(min) & Y(min) ":=200
220
                "Enter X(max) & Y(max) ":=260
170
                " Enter The Co-Ordinates Of The Line."
                 Enter X(1) & Y(1) :=150
178
                " Enter X(2) & Y(2) ":=189
170
```



Practical NO:-9

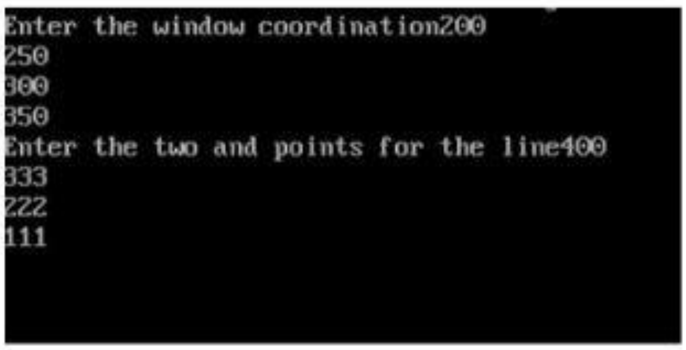
Write a program for Polygon clipping.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
#define Round(val)((int)(val+.5))
intmaxx,maxy,miny,minx;
void main()
{
intgd=DETECT,gm;
void clipping(intxa,intya,intxb,int y);
intxa,xb,ya,yb;
printf("Enter the window coordination");
scanf("%d%d%d%d",&minx,&maxy,&maxx,&miny);
printf("Enter the two and points for the line");
scanf("%d%d%d%d",&xa,&ya,&xb,&yb);
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
rectangle(minx,miny,maxx,maxy);
line(xa,ya,xb,yb);
getch();
closegraph();
}
void clipping(intxa,intya,intxb,intyb)
{
```

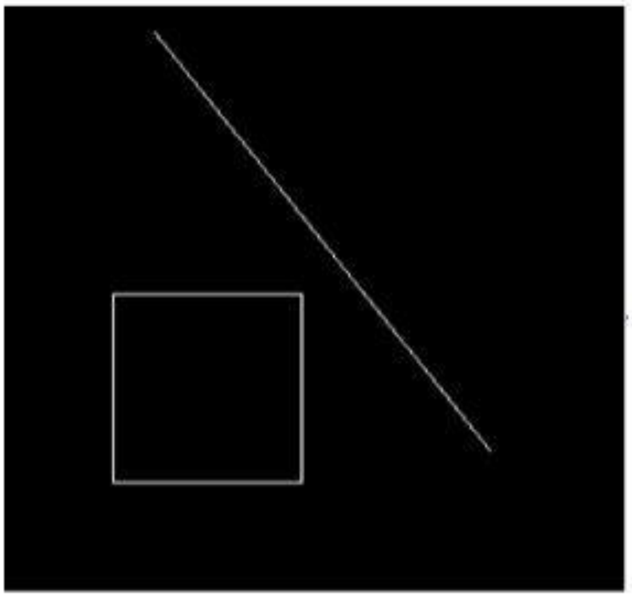
```
intDx=xb-xa,Dy=yb-ya,steps,k;
int visible1=0,visible2=0;
floatxin,yin,x=xa,y=ya;
if(abs(Dx)>abs(Dy))
steps=abs(Dx);
else
steps=abs(Dy);
xin=Dx/(float)steps;
yin=Dy/(float)steps;
putpixel(Round(x),Round(y),2);
for(k=0;k<steps;k++)
  {
     x+=xin;
     y+=yin;
if((y>miny&& y<maxx))
     {
        visible1=1;
putpixel(Round(x),Round(y),2);
      }
else
     visible2=1;
  }
if(visible1==0)
outtextxy(20,200,"complextely visible");
if(visible1==1 && visible2==1)
```

```
outtextxy(20,20,"partialy visible");
if(visible1==1&&visible2==0)
outtextxy(20,20,"completly visible");
}
```

OUTPUT:-



```
Enter the window coordination200
250
300
350
Enter the two and points for the line400
333
222
111
```

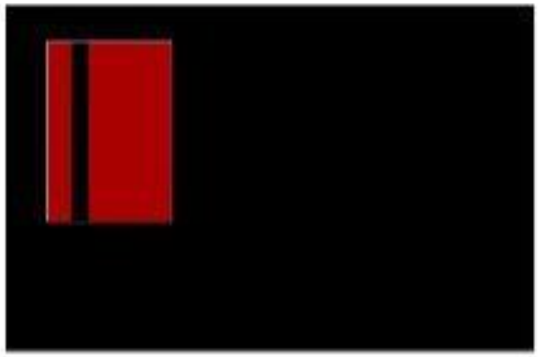


Practical NO:-10

Implement Flood Fill method to fill interior and exterior of a polygon.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void fill_right(x,y)
int x , y ;
{
if((getpixel(x,y) != WHITE)&&(getpixel(x,y) != RED))
{
putpixel(x,y,RED);
fill_right(++x,y);
x = x - 1 ;
fill_right(x,y-1);
fill_right(x,y+1);
}
delay(1);
}
void fill_left(x,y)
int x , y ;
{
if((getpixel(x,y) != WHITE)&&(getpixel(x,y) != RED))
{
putpixel(x,y,RED);
fill_left(–x,y);
x = x + 1 ;
fill_left(x,y-1);
fill_left(x,y+1);
}
delay(1);
}
void main()
{
int x,y,n,i;
int gd=DETECT,gm;
clrscr();

initgraph(&gd,&gm,"c:\\turboc3\\bgi");
/*- draw object -*/
line (50,50,200,50);
line (200,50,200,300);
line (200,300,50,300);
line (50,300,50,50);
/*- set seed point -*/
x = 100; y = 100;
fill_right(x,y);
fill_left(x-1,y);

getch();
}
```

OUTPUT:-



Practical NO:-11

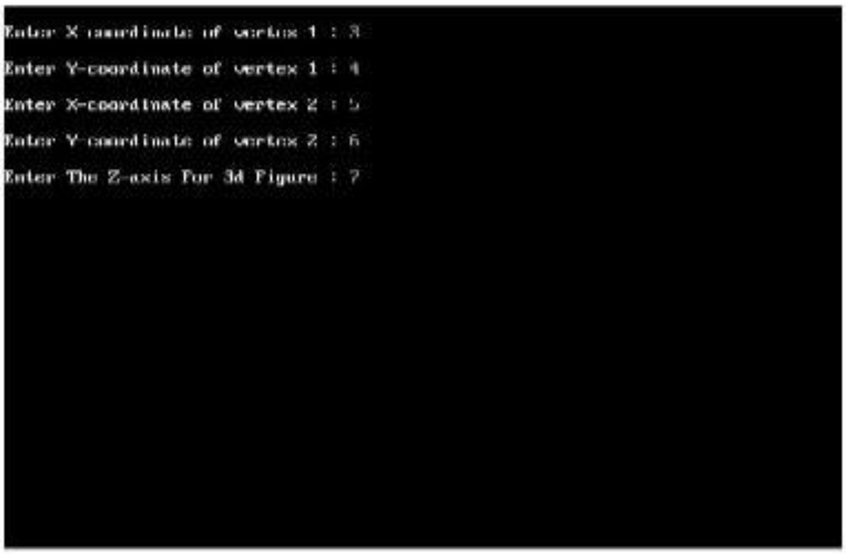
Write a program for displaying 3-D objects as 2-D display using perspective

transformation.

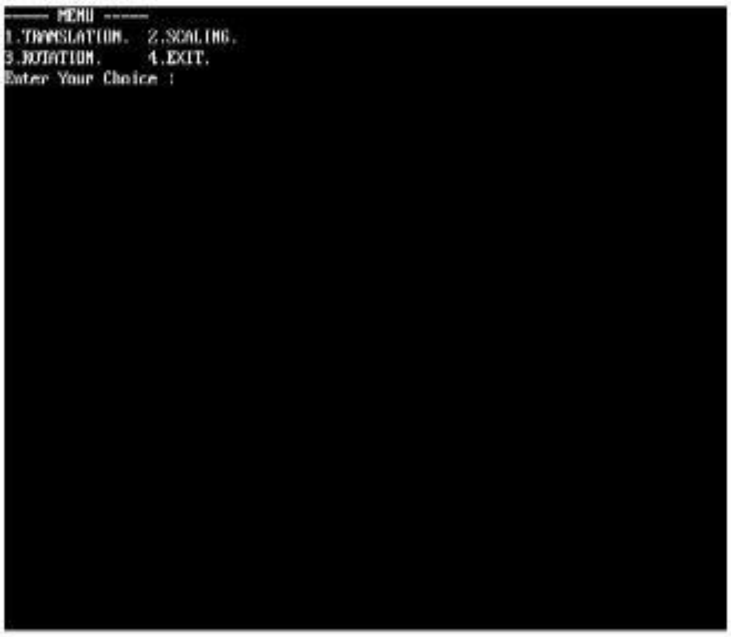
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
#include<stdlib.h>

int xp[2],yp[2],z;
void display();
void translate();
void scaling();
void rotation();
void matrixmul(int [4][4]);

void main()
{
int gd=DETECT,gm;
int ch,i;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
for(i=0;i<2;i++)
{
printf("\nEnter X-coordinate of vertex %d : ",i+1);
scanf("%d",&xp[i]);
printf("\nEnter Y-coordinate of vertex %d : ",i+1);
scanf("%d",&yp[i]);
}
printf("\nEnter The Z-axis For 3d Figure : ");
scanf("%d",&z);
clrscr();
cleardevice();
display(xp,yp);
getche();
do
{
printf("---- MENU ----");
printf("\n1.TRANSLATION.");
printf("\t2.SCALING.");
printf("\n3.ROTATION.");
printf("\t4.EXIT.");
printf("\nEnter Your Choice : ");
scanf("%d",&ch);
clrscr();
cleardevice();
display(xp,yp);
switch(ch)
{
case 1 : translate();
break;

case 2 : scaling();
break;

case 3 : rotation();
break;

case 4 : exit(0);

default:
outtextxy(1,66,"-PLEASE SELECT THE RIGHT OPTION-");
}
}
while(ch!=4);
getch();
closegraph();
}

void translate()
{
int p[4][4];
int tx,ty,tz,i,j;
for(i=0;i<4;i++)
for(j=0;j<4;j++)
p[i][j]=(i==j);
printf("\nEnter The TranslatingFactor tx : ");
scanf("%d",&tx);
printf("\nEnter The Translating Factor ty : ");
scanf("%d",&ty);
printf("\nEnter The Translating Factor tz : ");
scanf("%d",&tz);
clrscr();
cleardevice();
display();
p[0][3]=tx;
p[1][3]=ty;
p[2][3]=tz;
matrixmul(p);
}
```

```
line(getmaxx()/2+xp[1],getmaxy()/2-yp[0],getmaxx()/2+x4,getmaxy()/2-y3);
}
}

void matrixmul(int a[4][4])
{
float res[4][1],b[4][1];
int i,j,k,l;
for(i=0;i<2;i++)
{
b[0][0]=xp[i];
b[1][0]=yp[i];
b[2][0]=z;
b[3][0]=1;
for(j=0;j<4;j++)
{

for(k=0;k<1;k++)
{
res[j][k]=0;
for(l=0;l<4;l++)
{
res[j][k]=res[j][k]+(a[j][l]*b[l][k]);
}
}
}
xp[i]=res[0][0];
yp[i]=res[1][0];
}
z=res[2][0];
display(xp,yp);
}
```

OUTPUT:-



```
Enter X-coordinate of vertex 1 : 3

Enter Y-coordinate of vertex 1 : 4

Enter X-coordinate of vertex 2 : 5

Enter Y-coordinate of vertex 2 : 6

Enter The Z-axis For 3d Figure : 7
```



```
----- MENU -----
1.TRANSLATION.  2.SCALING.
3.ROTATION.     4.EXIT.
Enter Your Choice :
```

Practical NO:-13

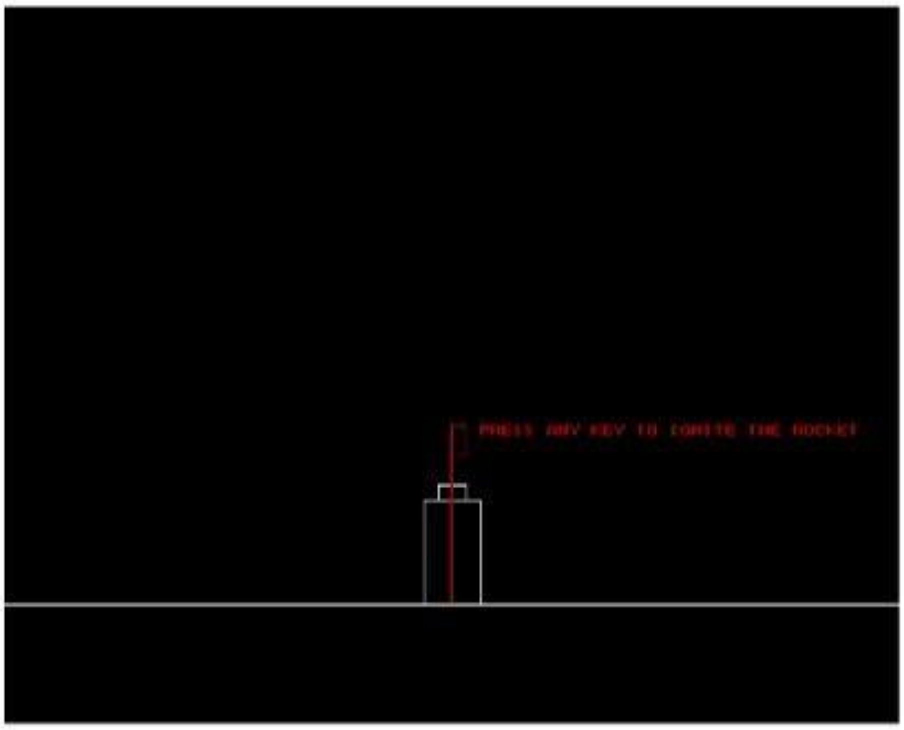
Write a program to draw different shape and fill them with various patteren.

```
#include<conio.h>
#include<graphics.h>
#include<stdio.h>
#include<math.h>
void main()
{
int gd,gm;
int x,y;
int i,j,kk;

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

setcolor(WHITE);
line(0,400,640,400);
rectangle(300,330,340,400);
rectangle(310,320,330,330);
setcolor(4);
line(319,280,319,398);
line(320,280,320,398);
rectangle(320,280,330,300);
outtextxy(340,280,"PRESS ANY KEY TO IGNITE THE ROCKET");
getch();
for(j=400;j<640;j++)
{
cleardevice();
setcolor(WHITE);
line(0,j,640,j);
rectangle(300,j-70,340,j);
rectangle(310,j-80,330,j-70);

setcolor(RED);
line(319,280,319,400);
line(320,280,320,400);
rectangle(320,280,330,300);

setcolor(YELLOW);
circle(325,300,2);
```

```
delay(5);
}

for(i=400;i>340;i-)
{
cleardevice();

setcolor(RED);
line(319,i,319,i-120);
line(320,i,320,i-120);
rectangle(320,i-120,330,i-100);

setcolor(YELLOW);
circle(325,i-100,2);
delay(25);
}

cleardevice();
kk=0;
for(j=100;j<350;j++)
{
if(j%20==0)
{
setcolor(kk);
kk=kk+3;
delay(50);
}
ellipse(320,30,0,360,j+100,j+0);
}
for(j=100;j<350;j++)
{
if(j%20==0)
{
setcolor(BLACK);
delay(2);
}
ellipse(320,30,0,360,j+100,j+0);
}
cleardevice();
for(i=0;i<70;i++)
{
setcolor(i);
settextstyle(GOTHIC_FONT,HORIZ_DIR,6);
outtextxy(110,150,"HAPPY NEWYEAR");
```

```
delay(90);
}
getch();
}
```

OUTPUT:-



```
PRESS ANY KEY TO IGNITE THE ROCKET
```