



LAB ASSIGNMENT

Course Code: CSE-4202

Course Title: Computer Graphics and Animation Lab

Submitted by,	Submitted to,
Saima Siddique Tashfia ID: CE-16022 4 th YEAR 2 nd SEMESTER Dept. of CSE MBSTU	Lubna Yasmin Pinky Assistant Professor Dept. of CSE MBSTU

Experiment No.: 01

Experiment Name: Study of basic graphics functions defined in “**graphics.h**”.

i) Program to create a progress bar using graphics.h

Theory:

To create a process bar with **graphics.h** we need to initialize a graph with two parameters and a path to the "bgi" folder in our system.

For initialize the graphics we use: **initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");**

initgraph() function is a function of graphics.h header file.

After that, we will call the function called **settextstyle()** with 3 parameters(font-family, direction, stroked size) then we call **outtextxy()** with 3 parameters(x coordinate, y-coordinate, string), then a for loop to produces process bar from x1 to x2 coordinates with height h.

Code:

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

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

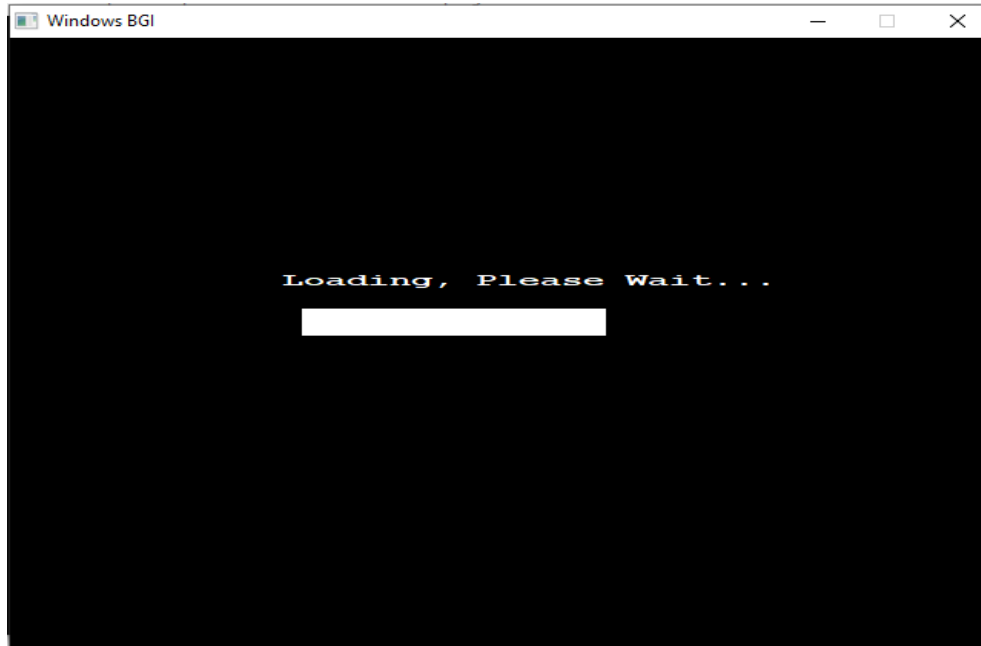
    settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
    outtextxy(177,180,"Loading, Please Wait...");

    for(int i=190;i<600;i++)
    {
        delay(30);
        line(i,210,i,230);
    }

    closegraph();
    _getch();

    return 0;
}
```

Output:



Experiment Name: ii) Program to create a smiley face using graphics.h

Theory:

The First thing is to initiate a graph. This will allow us to make graphics. Second thing is to set the color to yellow. As we want our smiley to be in yellow.

After that, with the help of **circle()** function create a circle by passing the parameters for a center with a radius. This will allow us to make a circle at our desired location.

Third thing is to fill the circle with yellow color with the help of **setfillstyle()** and **floodfill()**.

Fourth thing is to make four ellipses with the color black by calling the function **fillellipse()**.

Code:

```
// C program to create a smiley face
#include<graphics.h>
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
using namespace std;
```

```

int main()
{

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

    settextstyle(9,HORIZ_DIR,1);
    outtextxy(50, 200, "Program to create smiley face using graphics.h");
    // Set color of smiley to yellow
    setcolor(YELLOW);

    // creating circle and fill it with
    // yellow color using floodfill.
    circle(300, 100, 50);
    setfillstyle(SOLID_FILL, YELLOW);
    floodfill(300, 100, YELLOW);

    // Set color of background to black
    setcolor(BLACK);
    setfillstyle(SOLID_FILL, BLACK);

    // Use fill ellipse for creating eyes
    fillellipse(310, 85, 2, 6);
    fillellipse(290, 85, 2, 6);

    // Use ellipse for creating mouth
    ellipse(300, 100, 205, 335, 20, 9);
    ellipse(300, 100, 205, 335, 20, 10);
    ellipse(300, 100, 205, 335, 20, 11);

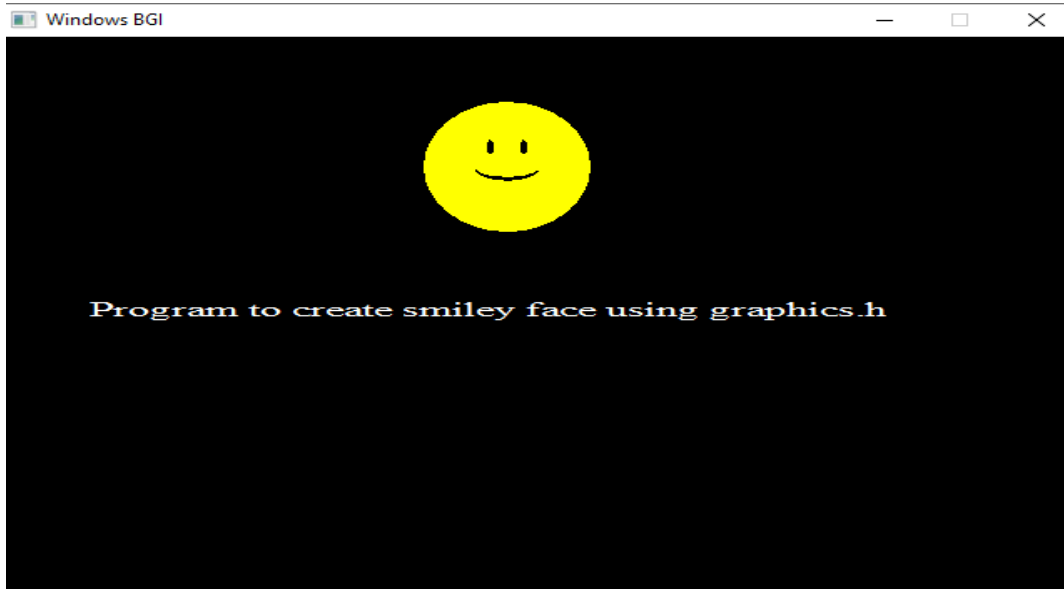
    getch();

    // closegraph function closes the
    // graphics mode and deallocates
    // all memory allocated by
    // graphics system
    closegraph();

    return 0;
}

```

Output:



Experiment Name: iii) Program to create polygons of various shapes using graphics.h

Theory:

Polygon function accepts an array of numbers, which should be even as these numbers are the coordinates of different vertices of the shape.

The First and the last co-ordinate means the first two and the last two numbers should be repeated to complete the shape, as every polygon is close.

drawpoly() function can be used to create any polygon as it asks for two parameters to be passed.

First is the number of coordinates, which as we have already talked about defines the location of the vertices.

Seconds is the array that contains all these numbers, defines the pairs of points on the x and y-axis.

These functions are from the header file graphics.h and should be included.

Code:

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

int main()
{
    //initilizing graphic driver and
```

```

//graphic mode variable
int graphicdriver=DETECT,graphicmode;

//calling initgraph with parameters
initgraph(&graphicdriver,&graphicmode,"c:\\turbo3\\bgi");

//Printing message for user
outtextxy(10, 10 + 10, "Program to draw polygon of different shapes in C
graphics");

//points of ploygon 1
int p1[]={520,250,230,470,290,240,520,250};

//points of ploygon
int p2[]={120,150,20,200,150,267,450,50,120,150};

setcolor(YELLOW);
//drawing polygon 1
drawpoly(4, p1);

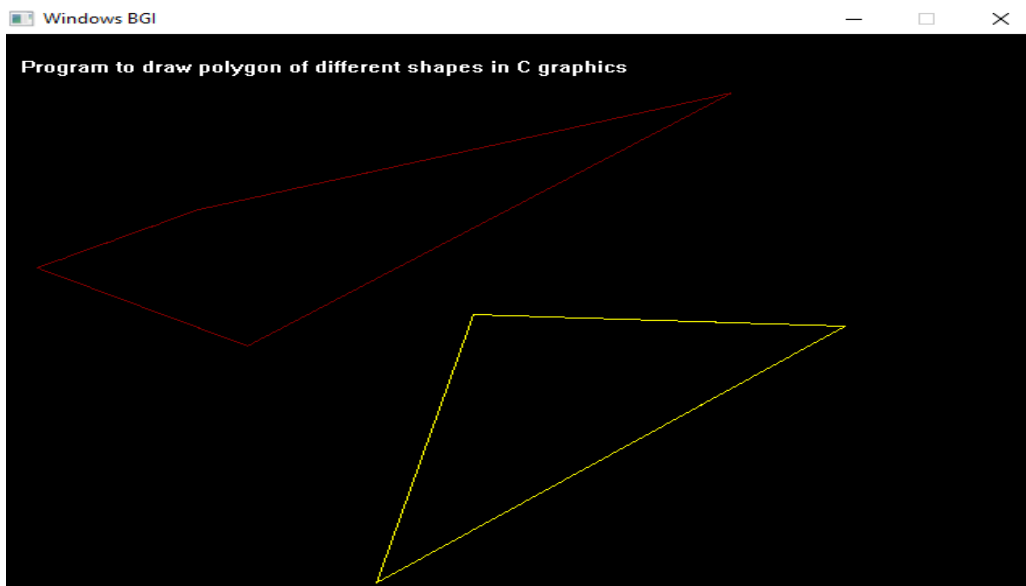
setcolor(RED);
//drawing polygon 2
drawpoly(5, p2);

getch();

return 0;
}

```

Output:



Experiment Name: iv) Print text in different fonts using graphics.h

Theory:

The Fonts, Size, and directions are the crucial part of the styling of the texts.

There is a very useful function called **settextstyle()** which takes three parameters that define the **font, direction, charsize** of the text. These are all numbers therefore in the example below, you can see the for loop which is providing the input for the function.

We can also pass the exact number for the font or charsize or direction of the text. This is a fairly useful concept because all you need is a number to be passed to the function before outputting the text on the screen.

Code:

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

int main()
{
    //initilizing graphic driver and
    //graphic mode variable
    int graphicdriver=DETECT,graphicmode;

    //calling initgraph with parameters
    initgraph(&graphicdriver,&graphicmode,"c:\\turbo3\\bgi");

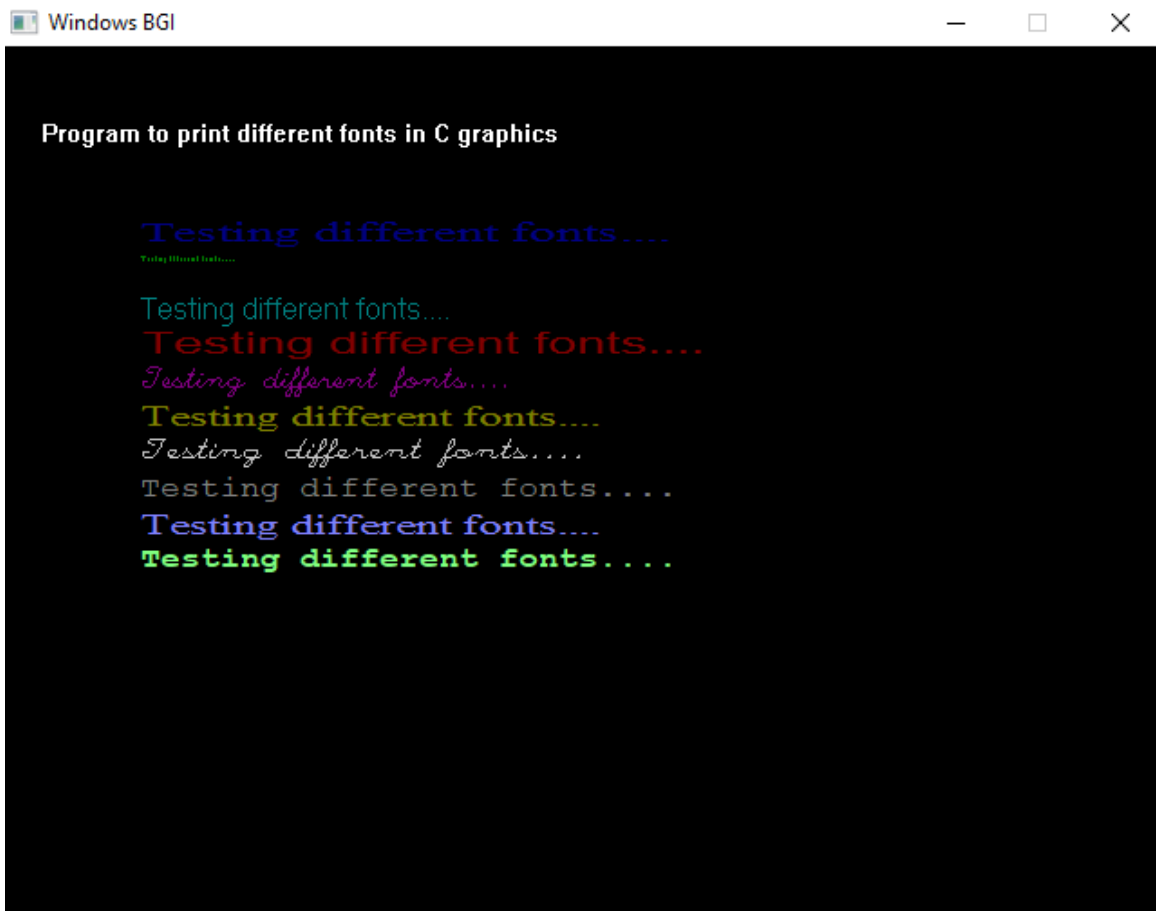
    //Printing message for user
    outtextxy(20, 20 + 20, "Program to print different fonts in C graphics\n");

    //initilizing variables
    int x = 75, y = 75, f = 0;

    //for loop to print different fonts
    for (f = 0; f <= 10; f++)
    {
        settextstyle(f, HORIZ_DIR, 1);
        setcolor(f);
        outtextxy(x, y, "Testing different fonts....");
        y = y + 20;
    }

    getch();
    return 0;
}
```

Output:



Experiment Name: v) Program to design traffic signal using graphics.h functions

Theory:

To create a rectangle and circle in our graph all we need to do is use **rectangle()** and **circle()** functions. For a rectangle, we will call the function called rectangle with four numbers as the four coordinates of the rectangle. With these parameters, we will have a rectangle on our screen.

For circle, we will call the function called circle() with three numbers as the coordinates of the center and radius. So this function will create a circle with a center with the given radius.

setcolor() function is used to set the color of the shape which we will draw after that statement.

Now, the first thing is we need to create a rectangle that will include all three circles with red, yellow, and green. Then, make the circles in a vertical direction.

Code:

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

int main()
{
    //initilizing graphic driver and
    //graphic mode variable
    int graphicdriver=DETECT,graphicmode;

    //calling initgraph with parameters
    initgraph(&graphicdriver,&graphicmode,"c:\\turbo3\\bgi");

    //Printing message for user
    outtextxy(50, 50 + 50, "Program to create traffic signal in C graphics");

    //initilizing variables
    int middlex, middley;

    //getting middle x and y
    middlex = getmaxx()/2;
    middley = getmaxy()/2;

    //setting color as white for the outline
    setcolor(WHITE);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    rectangle(middlex-30,middley-80,middlex+30,middley+80);
    circle(middlex, middley-50, 22);

    //filling red color to signify stop sign
    setfillstyle(SOLID_FILL,RED);
    floodfill(middlex, middley-50,WHITE);
    setcolor(BLUE);
    outtextxy(middlex-15,middley-50,"STOP");

    //setting color as white for the outline
    setcolor(WHITE);
    rectangle(middlex-30,middley-80,middlex+30,middley+80);
    circle(middlex, middley, 20);

    //filling yellow color to signify ready sign
    setfillstyle(SOLID_FILL,YELLOW);
    floodfill(middlex, middley,WHITE);
    setcolor(BLUE);
    outtextxy(middlex-18,middley-3,"READY");
```

```

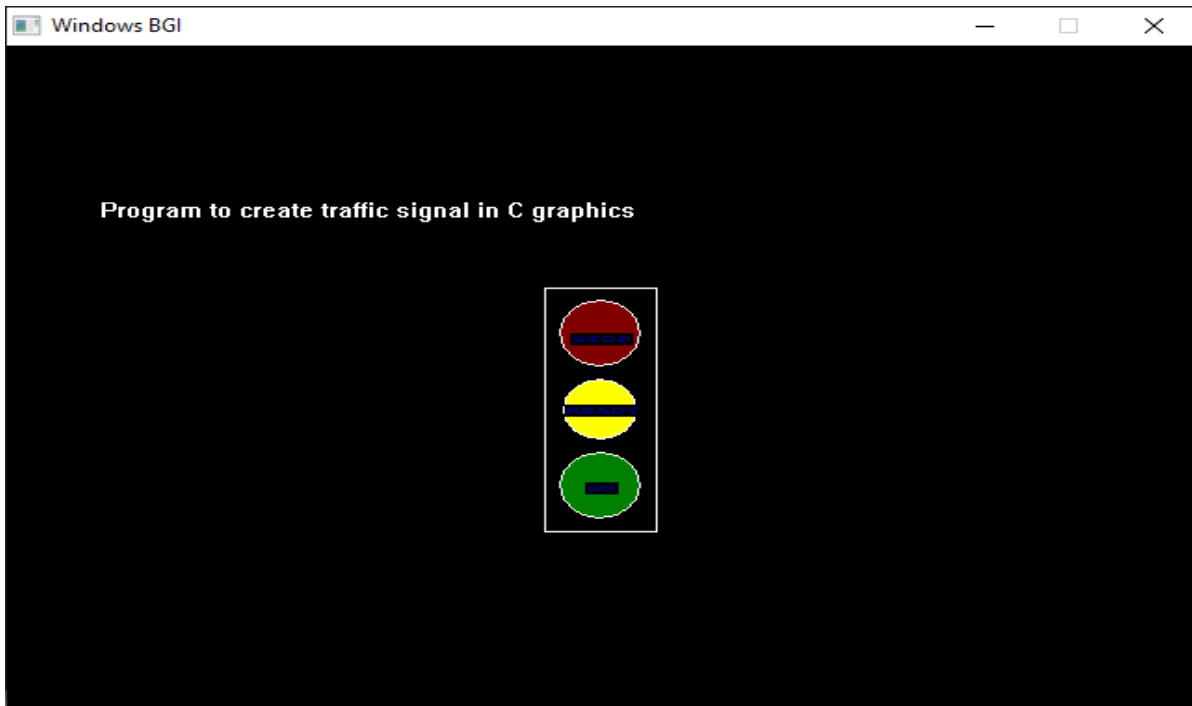
//setting white color for outline
setcolor(WHITE);
rectangle(middlex-30,middley-80,middlex+30,middley+80);
circle(middlex, middley+50, 22);

//filling green color to signify go sign
setfillstyle(SOLID_FILL, GREEN);
floodfill(middlex, middley+50, WHITE);
setcolor(BLUE);
outtextxy(middlex-7, middley+48, "GO");
setcolor(RED);
settextstyle(SCRIPT_FONT, HORIZ_DIR, 4);

getch();
return 0;
}

```

Output:



Experiment Name: vi) Fill color using putpixel() function of graphics.h

Theory:

To put a pixel on the screen at a particular position, calling the **putpixel()** function is a good way. This function takes three parameters as the position of the pixel and also the color of the pixel. In the example below, we implemented three for loops which will make three squares of different colors, because we have decided the boundaries of the squares as lower and upper limits of the loop. Every loop has a different upper limit and lower limit, with different pixel colors. So, it will look like three different squares of different colors, even though these are just pixels. If we change the limits of loops the squares will also change and we can also change the color of the pixel and color of the square will change.

Code:

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

int main()
{
    //initilizing graphic driver and
    //graphic mode variable
    int graphicdriver=DETECT,graphicmode;

    //calling initgraph
    initgraph(&graphicdriver,&graphicmode,"c:\\turbo3\\bgi");

    //Printing message for user
    outtextxy(20, 20 + 20, "Program to fill color in different areas using putpixel in C
    graphics");

    //initilizing the type of variables

    int i,j;

    //loop for first area
    for(i=60;i<=120;i++)
    {
        for(j=60;j<=120;j++)
        {
            putpixel(j, i, YELLOW);
        }
    }

    //loop of second area
    for(i=121;i<=180;i++)
    {
```

```

        for(j=121;j<=180;j++)
        {
            putpixel(j, i, GREEN);
        }
    }

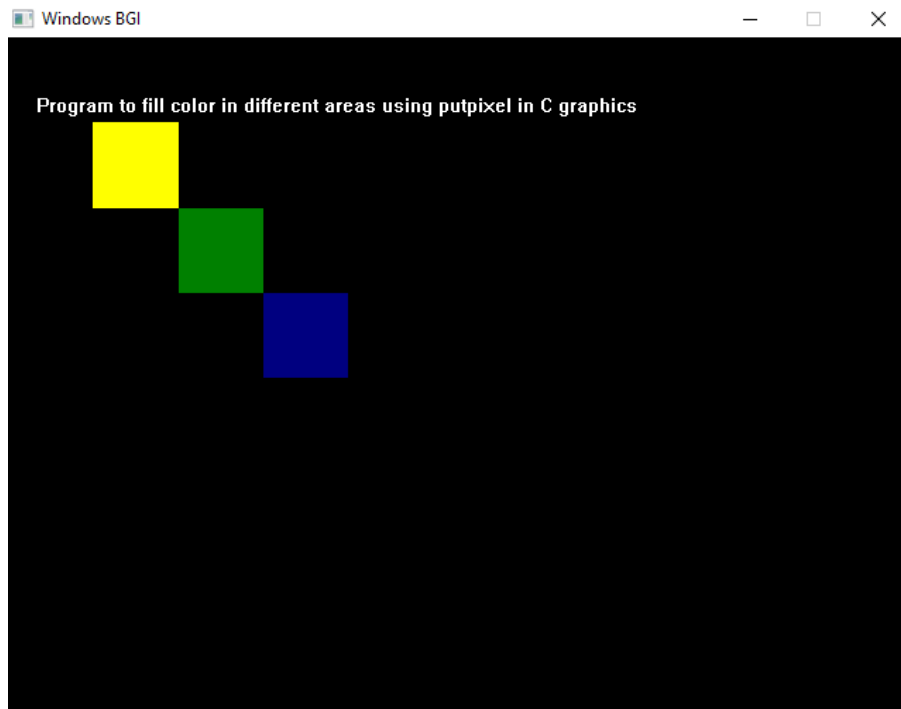
    //loop of third area
    for(i=181;i<=240;i++)
    {
        for(j=181;j<=240;j++)
        {
            putpixel(j, i, BLUE);
        }
    }

    getch();

    return 0;
}

```

Output:



Experiment Name: vii) bar3d() function in graphics.h

Theory:

This is a very useful function as it allows us to make 3d shapes in C. To make this 3d bar, the first thing is to initiate a graph. After initiating the graph we have to call the **bar3d()** function.

This function takes six parameters, which should be numbers. The Parameters defines the **Left co-ordinate, Top co-ordinate, Right co-ordinate, Bottom co-ordinate, depth co-ordinate, and Top flag**. The first four parameters define the coordinate of the **front face**, while the fifth parameter defines the **depth or the width of the bar**.

To make it look bigger and more realistic, keep the depth relative to the width and length.

Code:

```
// C Implementation for bar3d() function
#include <graphics.h>

// driver code
int main()
{
    // gm is Graphics mode which is
    // a computer display mode that
    // generates image using pixels.
    // DETECT is a macro defined in
    // "graphics.h" header file
    int gd = DETECT, gm;

    // initgraph initializes the
    // graphics system by loading a
    // graphics driver from disk
    initgraph(&gd, &gm, "");

    // location of sides
    int left, top, right, bottom;

    // depth of the bar
    int depth;

    // top flag denotes top line.
    int topflag;
    settextstyle(BOLD_FONT, HORIZ_DIR, 2);
    outtextxy(375, 200, "3D BAR GRAPH");
    // left, top, right, bottom,
    // depth, topflag location's
    setfillstyle(1, YELLOW);

    bar3d(left = 150, top = 250,
```

```

    right = 190, bottom = 350,
    depth = 20, topflag = 1);

    bar3d(left = 220, top = 150,
    right = 260, bottom = 350,
    depth = 20, topflag = 2);

    bar3d(left = 290, top = 200,
    right = 330, bottom = 350,
    depth = 20, topflag = 1);

setcolor(WHITE);
    // y axis line
    line(100, 50, 100, 350);

    // x axis line
    line(100, 350, 400, 350);
    outtextxy(65,60,"Y");
    outtextxy(390,370,"X");

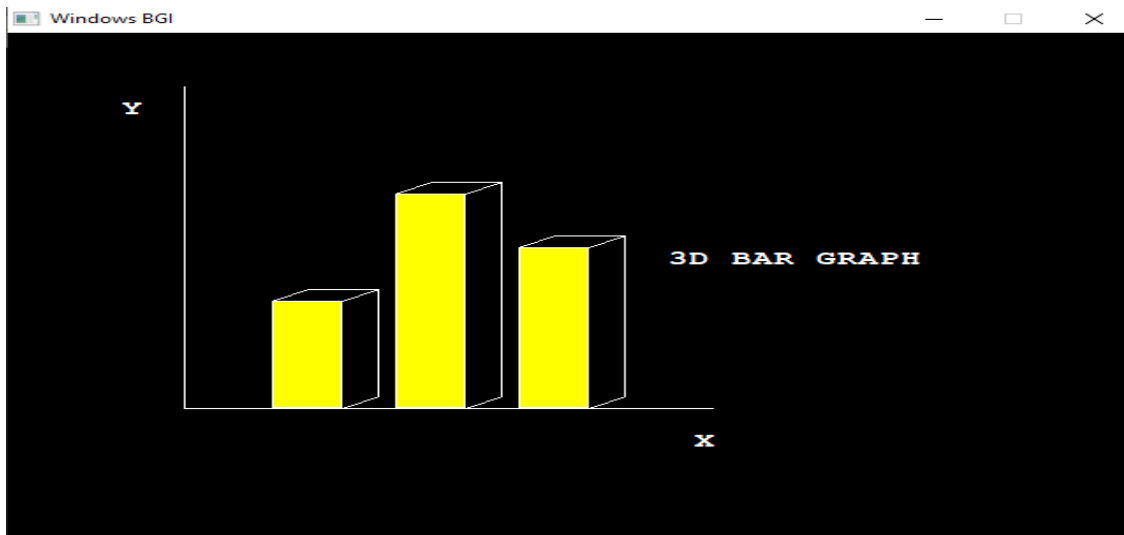
    getch();

    // closegraph function closes the
    // graphics mode and deallocates
    // all memory allocated by
    // graphics system .
    closegraph();

    return 0;
}

```

Output:



Experiment No.: 02

Experiment Name: i) Implement Bresenham's mid-point line algorithm for the line coordinates (2, 1) and (7, 4).

Theory:

In the Mid-Point algorithm, for any given/calculated previous pixel $P(X_p, Y_p)$, there are two candidates for the next pixel closest to the line, $E(X_p+1, Y_p)$ and $NE(X_p+1, Y_p+1)$ (E stands for East and NE stands for North-East).

In the Mid-Point algorithm, we do the following:

1. Find the middle of two possible next points. Middle of $E(X_p+1, Y_p)$ and $NE(X_p+1, Y_p+1)$ is $M(X_p+1, Y_p+1/2)$.
2. If M is above the line, then choose E as the next point.
3. If M is below the line, then choose NE as the next point.

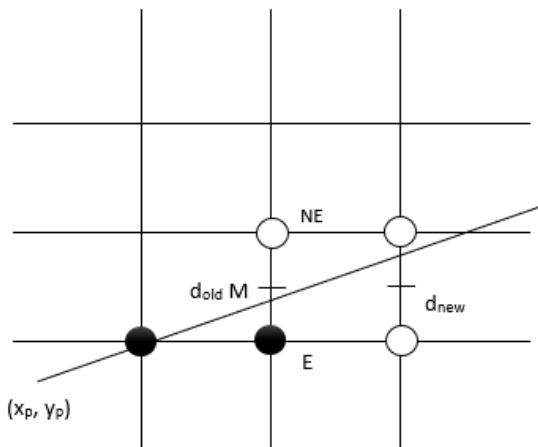


Figure Mid-point lies above the line, East pixel chosen

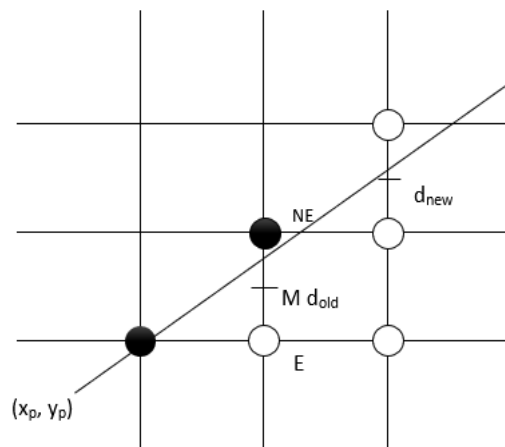


Figure Mid-point lies below the line, North-East pixel chosen

Summarizing the algorithm

First pixel (x_0, y_0)

First midpoint is $(x_0 + 1, y_0 + \frac{1}{2})$

$$\begin{aligned} F(x_0 + 1, y_0 + \frac{1}{2}) &= a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c \\ &= ax_0 + by_0 + c + a + b/2 \\ &= 0 + a + b/2 \end{aligned}$$

$$d_{\text{start}} = a + b/2$$

$$d_{\text{start}} = dy - \frac{dx}{2}$$

To eliminate the fraction in d_{start} , multiply original F by 2

$$F(x, y) = 2(ax + by + c)$$

Hence,

d_{start}	$= 2dy - dx$
Δ_E	$= 2dy$
Δ_{NE}	$= 2(dy - dx)$

Code:

```
#include <iostream>
#include <graphics.h>
using namespace std;
//Midpoint line drawing
void line1(int x1,int y1,int x2,int y2){

    int x,y,d0,d1,d2,a,b;
    y=y1;
    a=y1-y2;    //Algorithm of a in straight line equation
    b=x2-x1;    //Algorithm of b in straight line equation
    d0=2*a+b;    //Incremental initial value
    d1=2*a;    //Increment when >= 0
    d2=2*(a+b); //Increment when < 0
    for(x=x1;x<=x2;x++){
        putpixel(x,y,GREEN); //Brighten
        if(d0<0){
            y++;
            d0+=d2;
        }else{
            d0+=d1;
        }
    }
}
//Bresenham line drawing algorithm
void line2(int x1,int y1,int x2,int y2){

    int x,y,dx,dy,d;
    y=y1;
    dx=x2-x1;
    dy=y2-y1;
    d=2*dy-dx;    //Initial value of increment d
```



```

for(x=x1;x<=x2;x++){
    putpixel(x,y,GREEN); //Brighten
    if(d<0){
        d+=2*dy;
    }else{
        y++;
        d+=2*dy-2*dx;
    }

}

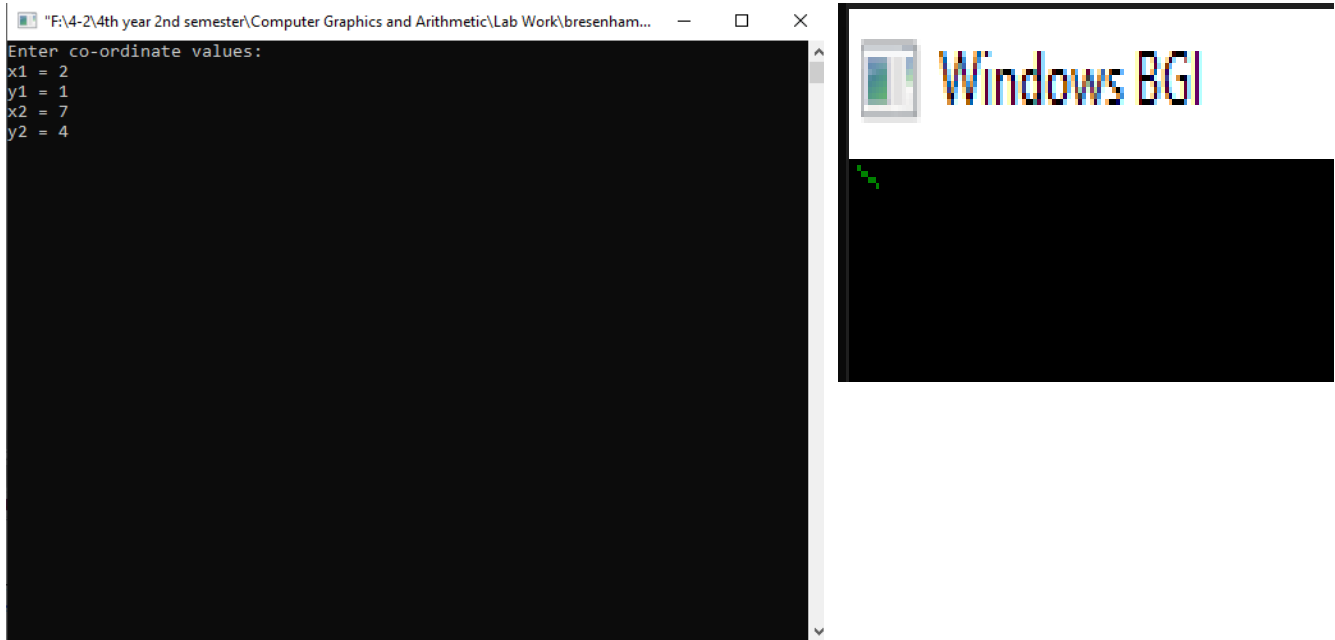
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TC\\BGI");
    int x1,y1,x2,y2;
    cout<<"Enter co-ordinate values: \n";
    cout<<"x1 = ";
    cin>>x1;
    cout<<"y1 = ";
    cin>>y1;
    cout<<"x2 = ";
    cin>>x2;
    cout<<"y2 = ";
    cin>>y2;

    line1(x1,y1,x2,y2); //Line drawing
    getch();           //Wait for user operation
    closegraph();      //Close graphics
    return 0;
}

```

Output:



Experiment Name: ii) Implement DDA algorithm for the line coordinates (2, 1) and (7, 4).

Theory:

To find the pixels lying on the line using DDA line algorithm, compute $y_i = mx_i + B$ for each x_i , increment x by 1 starting with the left endpoint. Intensify the pixel at $(x_i, \text{Round}(y_i))$, where $\text{Round}(y_i) = \text{Floor}(0.5 + y_i)$. Here the pixel whose distance to the true line is the smallest is chosen.

$$\begin{aligned} y_{i+1} &= mx_{i+1} + B \\ &= m(x_i + \Delta x) + B \\ &= mx_i + B + m\Delta x \\ &= y_i + m\Delta x \end{aligned}$$

And if $\Delta x = 1$, then

$$y_{i+1} = y_i + m\Delta x$$

Code:

```
// program for DDA line generation
#include <iostream>
#include <graphics.h>
```

```

#include<math.h>
using namespace std;
//Function for finding absolute value
int abs (int n)
{
    return ( (n>0) ? n : ( n * (-1)));
}

//DDA Function for line generation
void DDA(int Xo, int Yo, int X1, int Y1)
{
    // calculate dx & dy
    int dx = X1 - Xo;
    int dy = Y1 - Yo;

    // calculate steps required for generating pixels
    int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

    // calculate increment in x & y for each steps
    float Xinc = dx / (float) steps;
    float Yinc = dy / (float) steps;

    // Put pixel for each step
    float X = Xo;
    float Y = Yo;
    for (int i = 0; i <= steps; i++)
    {
        putpixel (round(X),round(Y),RED); // put pixel at (X,Y)
        X += Xinc;           // increment in x at each step
        Y += Yinc;           // increment in y at each step
        delay(100);          // for visualization of line-
        // generation step by step
    }
}

// Driver program
int main()
{
    int gd = DETECT, gm;

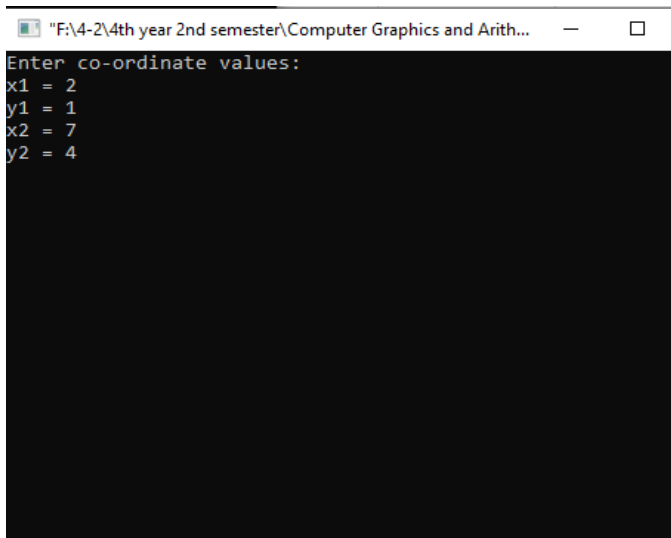
    // Initialize graphics function
    initgraph (&gd, &gm, "");

    int x1,y1,x2,y2;
    cout<<"Enter co-ordinate values: \n";
    cout<<"x1 = ";

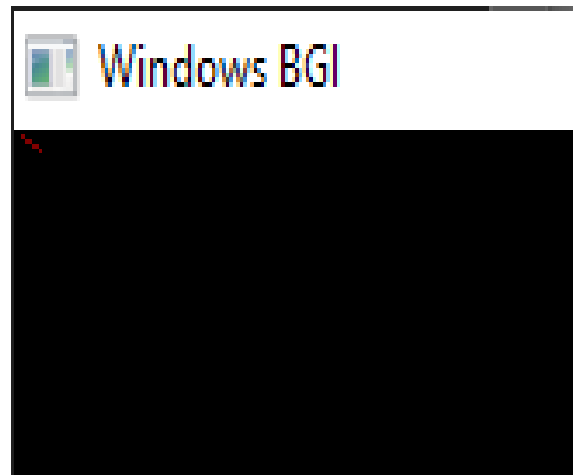
```

```
cin>>x1;
cout<<"y1 = ";
cin>>y1;
cout<<"x2 = ";
cin>>x2;
cout<<"y2 = ";
cin>>y2;
DDA(x1, y1, x2, y2);
getch();      //Wait for user operation
closegraph(); //Close graphics
return 0;
}
```

Output:



```
"F:\4-2\4th year 2nd semester\Computer Graphics and Arith..."
Enter co-ordinate values:
x1 = 2
y1 = 1
x2 = 7
y2 = 4
```



Experiment No.: 03

Experiment Name: Write code to fill a polygon by using scan line polygon filling algorithm.

Theory:

Scanline filling is the filling up of polygons using horizontal lines or scanlines. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. This algorithm works by intersecting the scanline with polygon edges and fills the polygon between pairs of intersections.

Components of Polygon fill:

1. **Edge Buckets:** It contains an edge's information. The entries of the edge bucket vary according to the data structure you have used.
2. **Edge Table (ET):** It contains all edges sorted by their smaller y coordinates. The ET is typically made by using the bucket sort with as many buckets as there are scan lines. Within each bucket, edges are kept in order of increasing x coordinates of the lower endpoint. Each entry in the ET contains the y_{\max} coordinate of the edge, the x coordinate of the bottom endpoint (x_{\min}), and the x increment $1/m$ (Fig 1). Filling is complete once all of the edges are removed from the ET.

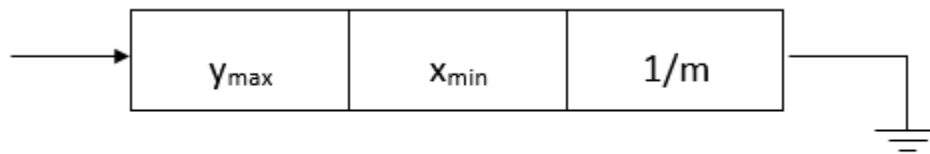


Figure 1 Each edge data item

3. **Active Edge Table (AET):** IT maintains the current edges being used to fill in the polygon. Edges are pushed into the AET from the Edge Table when an edge's y_{\min} is equal to the current scan line being processed. The Active List will be re-sorted after every pass.

In the Active edge table for any given polygon, the processing is done as follows:

1. Set y to the smallest y coordinate that has an entry in ET; i.e, y for the first non-empty bucket.
2. Initialize the AET to be empty.
3. Repeat until AET and ET are empty.
 - 3.1 Move from ET bucket y to the AET those edges whose $y_{\min} = y$ (entering edge).
 - 3.2 Remove from the AET those entries for which $y = y_{\max}$ (edges not involved in the next scan line), then sort the AET on x .
 - 3.3 Fill in desired pixel values on scan line y by using pairs of x coordinate from the AET.
 - 3.4 Increment y by 1 (to the coordinate of the next scan line).
 - 3.5 For each nonvertical edge remaining in the AET, update x for the new y .

Code:

```
/* Program to Fill a Polygon Using Scan-Line Fill Algorithm in C++.*/
```

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

//Declaration of class point
class point
{
    public:
    int x,y;
};

class poly
{
    private:
    point p[20];
    int inter[20],x,y;
    int v,xmin,ymin,xmax,ymax;
    public:
    int c;
    void read();
    void calcs();
    void display();
    void ints(float);
    void sort(int);
};

void poly::read()
{
    int i;
    cout<<"\n\tSCAN_FILL ALGORITHM";
    cout<<"\n Enter the no of vertices of polygon:";
    cin>>v;
    if(v>2)
    {
        for(i=0;i<v; i++) //ACCEPT THE VERTICES
        {
            cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";
            cout<<"\n\tx"<<(i+1)<<"=";
            cin>>p[i].x;
```

```

        cout<<"\n\ty"<<(i+1)<<"=";
        cin>>p[i].y;
    }
    p[i].x=p[0].x;
    p[i].y=p[0].y;
    xmin=xmax=p[0].x;
    ymin=ymax=p[0].y;
}
else
    cout<<"\n Enter valid no. of vertices.";
}
//FUNCTION FOR FINDING
void poly::calcs()
{ //MAX,MIN
    for(int i=0;i<v;i++)
    {
        if(xmin>p[i].x)
            xmin=p[i].x;
        if(xmax<p[i].x)
            xmax=p[i].x;
        if(ymin>p[i].y)
            ymin=p[i].y;
        if(ymax<p[i].y)
            ymax=p[i].y;
    }
}
//DISPLAY FUNCTION
void poly::display()
{
    int ch1;
    char ch='y';
    float s,s2;
    do
    {
        cout<<"\n\nMENU:";
        cout<<"\n\n\t1 . Scan line Fill ";
        cout<<"\n\n\t2 . Exit ";
        cout<<"\n\nEnter your choice:";
        cin>>ch1;
        switch(ch1)
        {
            case 1:
                s=ymin+0.01;
                delay(100);
                cleardevice();
                while(s<=ymax)

```

```

        {
            ints(s);
            sort(s);
            s++;
        }
        break;
    case 2:
        exit(0);
    }

    cout<<"Do you want to continue?: ";
    cin>>ch;
}while(ch=='y' || ch=='Y');
}

void poly::ints(float z) //DEFINE FUNCTION INTS
{
    int x1,x2,y1,y2,temp;
    c=0;
    for(int i=0;i<v;i++)
    {
        x1=p[i].x;
        y1=p[i].y;
        x2=p[i+1].x;
        y2=p[i+1].y;
        if(y2<y1)
        {
            temp=x1;
            x1=x2;
            x2=temp;
            temp=y1;
            y1=y2;
            y2=temp;
        }
        if(z<=y2&& z>=y1)
        {
            if((y1-y2)==0)
                x=x1;
            else // used to make changes in x. so that we can fill our polygon after cerain
distance
            {
                x=((x2-x1)*(z-y1))/(y2-y1);
                x=x+x1;
            }
            if(x<=xmax && x>=xmin)
                inter[c++]=x;
        }
    }
}

```



```

    }
}

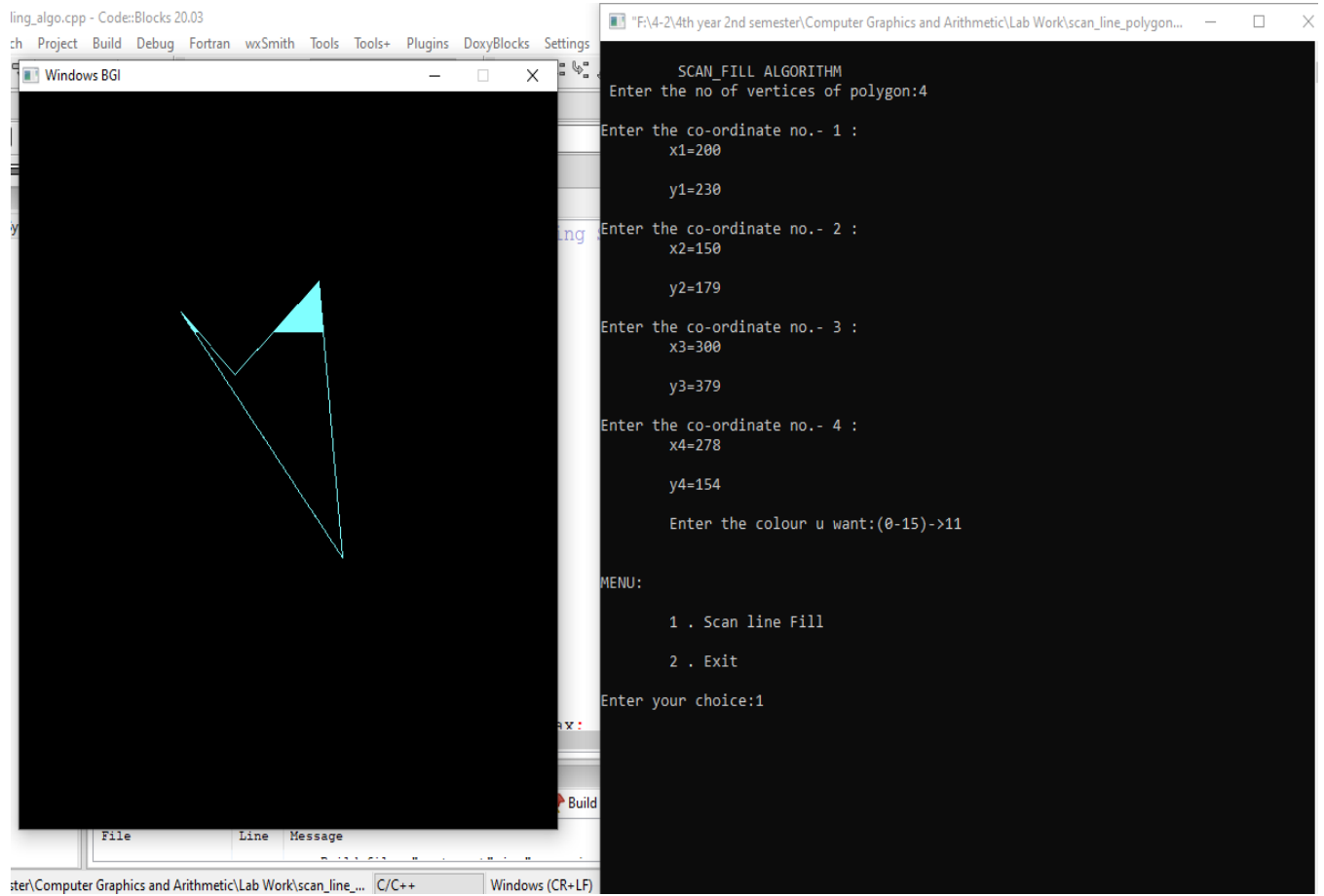
void poly::sort(int z) //SORT FUNCTION
{
    int temp,j,i;

    for(i=0;i<v;i++)
    {
        line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // used to make hollow outlines of a polygon
    }
    delay(100);
    for(i=0; i<c;i+=2)
    {
        delay(100);
        line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....
    }
}

int main() //START OF MAIN
{
    int cl;
    initwindow(500,600);
    cleardevice();
    poly x;
    x.read();
    x.calcs();
    cleardevice();
    cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour
    cin>>cl;
    setcolor(cl);
    x.display();
    closegraph(); //CLOSE OF GRAPH
    getch();
    return 0;
}

```

Output:



```
ling_algo.cpp - Code::Blocks 20.03
ch Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings

Windows BGI
SCAN_FILL ALGORITHM
Enter the no of vertices of polygon:4
Enter the co-ordinate no.- 1 :
x1=200
y1=230
Enter the co-ordinate no.- 2 :
x2=150
y2=179
Enter the co-ordinate no.- 3 :
x3=300
y3=379
Enter the co-ordinate no.- 4 :
x4=278
y4=154
Enter the colour u want:(0-15)->11
MENU:
1 . Scan line Fill
2 . Exit
Enter your choice:1
```

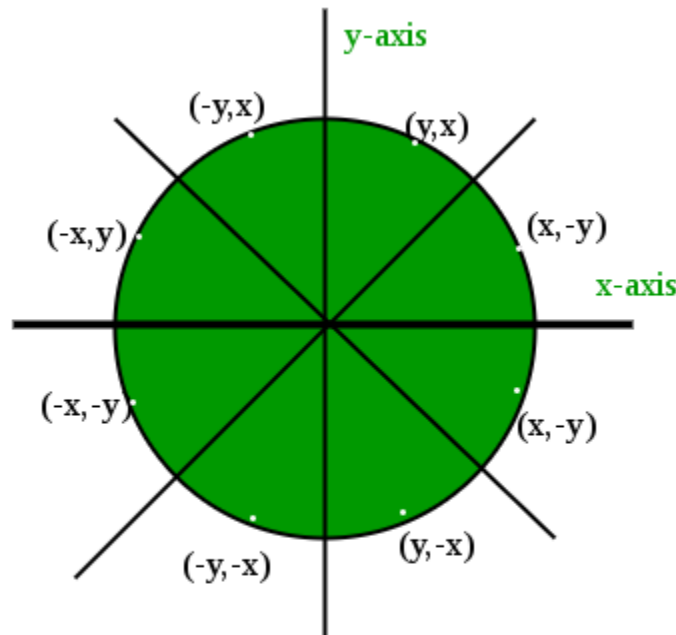
Experiment No.: 04

Experiment Name: Implement Mid-point circle drawing algorithm for the coordinate (5,-2) with radius 13.

Theory:

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.

For the whole 360 degrees of a circle, we will divide it into 8-parts each octant of 45 degrees. To do that we will use Bresenham's Circle Algorithm for the calculation of the locations of the pixels in the first octant of 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the 8 octants of the circle as shown below :



For a pixel (x,y) all possible pixels in 8 octants.

Now, we will see how to calculate the next pixel location from a previously known pixel location (x, y). In Bresenham's algorithm at any point (x, y) we have two options either to choose the next pixel in the east i.e. (x+1, y), or in the southeast i.e. (x+1, y-1).

Implicit Equation of a circle

$$F(x, y) = x^2 + y^2 - R^2$$

if $F(x, y) > 0 \rightarrow$ point outside the circle

if $F(x, y) < 0 \rightarrow$ point inside the circle

If the midpoint between the pixels E and SE is outside the circle, then pixel **SE** is closer to the circle.
 If the midpoint is inside the circle, pixel E is closer to the circle (Fig)

$$F(M) = F(x_p + 1, y_p - \frac{1}{2})$$

$$d_{old} = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

If $d \geq 0$, choose SE pixel

If $d < 0$, choose E pixel

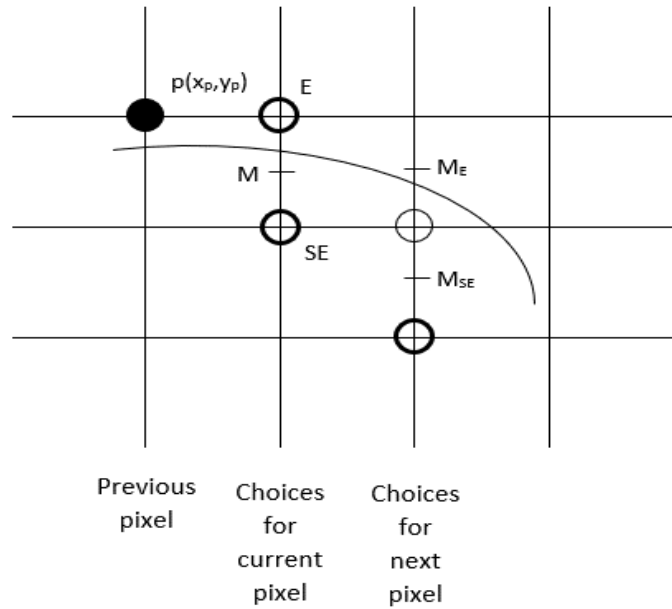


Figure Bresenham's Midpoint circle drawing.

$$\text{deltaE} = d_{new} - d_{old}$$

$$= F(x_p + 2, y_p - 1/2) - F(x_p + 1, y_p - 1/2)$$

$$= 2x_p + 3$$

$$\text{deltaSE} = F(x_p + 2, y_p - 3/2) - F(x_p + 1, y_p - 1/2)$$

$$= 2x_p - 2y_p + 5$$

$$\text{deltaE} = 2x_p + 3$$

$$\text{deltaSE} = 2(x_p - y_p) + 5$$

$$d_{start} = f(x_0 + 1, y_0 - 1/2) = f(1, R - 1/2) = f(1, R - 1/2)$$

$$= 5/4 - R$$

Compute Initial Condition

By limiting the algorithm to integer radii in the second octant, we know that the pixel lies on the circle at (0, R).

Next midpoint lies at $\left(1, R - \frac{1}{2}\right)$

$$F\left(1, R - \frac{1}{2}\right) = 1 + \left(R^2 - R + \frac{1}{4}\right) - R^2$$

$$d = \frac{5}{4} - R \quad \dots\dots\dots (i)$$

$\frac{5}{4}$ here is the real arithmetic.

To eliminate real arithmetic subtract $\frac{1}{4}$ on both sides, from equation (i)

$$d - \frac{1}{4} = 1 - R$$

Let new decision variable $h = d - \frac{1}{4}$

$$h = 1 - R$$

$$d < 0$$

$$d - \frac{1}{4} < 0$$

$$d < \frac{1}{4}$$

Since d starts with an integer value and is incremented by integer values (deltaE and deltaSE), we can change the comparison to just $d < 0$.

Code:

```
#include<iostream>
#include<conio.h>
#include<graphics.h>
using namespace std;
void drawCircle(int x, int y, int xc, int yc);
int main()
{
    int gd = DETECT, gm;
    int r, xc, yc, d, x, y;
    initgraph(&gd, &gm, "C:\\TC\\BGI");
    cout<<"Enter the center co-ordinates\n";
    cin>>xc>>yc;
    cout<<"Enter the radius of circle\n";
    cin>>r;
```

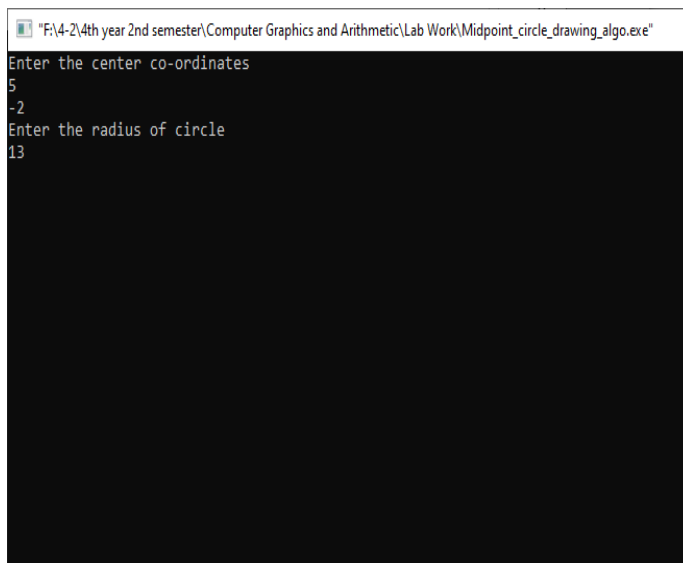
```

d = 1 - r;
x = 0;
y = r;
while(x < y)
{
    drawCircle(x,y,xc,yc);
    //++x;
    if(d < 0)
    {
        d = d + (2*x) + 3;
        x++;
    }
    else
    {
        //--y;
        //d = d + (2*x) + 1 - (2*y);
        d +=(x-y)*2 +5;
        x++;
        y--;
    }
}
getch();
closegraph();
}

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

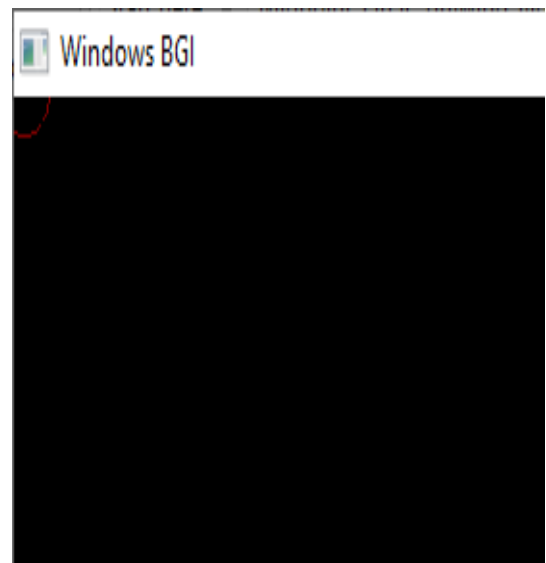
```

Output:



A screenshot of a Windows command prompt window. The title bar reads "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe". The prompt is at "C:\>". The user has entered "cd F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>". The user has entered "Midpoint_circle_drawing_algo.exe" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>". The user has entered "Enter the center co-ordinates" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>". The user has entered "5" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>". The user has entered "-2" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>". The user has entered "Enter the radius of circle" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>". The user has entered "13" and pressed Enter. The prompt is now at "F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>".

```
F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe
C:\>cd F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe
F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>Midpoint_circle_drawing_algo.exe
F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>Enter the center co-ordinates
5
-2
Enter the radius of circle
13
F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\Midpoint_circle_drawing_algo.exe>
```



Experiment No.: 05

Experiment Name: Write a menu-driven program to rotate, scale, and translate a line point, square, triangle about the origin.

Theory:

Rotation:

Rotations in computer graphics are a transformational operation. That means that it is a conversion from one coordinate space onto another.

The rotation of a 2D vector anti-clockwise in a plane is done as follows:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Thus, $x' = x * \cos\theta - y * \sin\theta$
and $y' = x * \sin\theta + y * \cos\theta$

For clockwise rotation put $\theta = (-\theta)$

Hence, $x' = x * \cos\theta + y * \sin\theta$
and $y' = y * \cos\theta - x * \sin\theta$

Scaling:

Matrix for Scaling,

$$S = \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Equations for scaling,

$$x' = x * S_x$$

$$y' = y * S_y$$

Translation:

Translation matrix:

$$\begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} \quad \text{Or} \quad \begin{vmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{vmatrix}$$

Equations for translation

$$x' = x + T_x$$

$$y' = y + T_y$$

Code:

```
#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<math.h>
using namespace std;
void disp(int n,float c[][3])
{
    float maxx,maxy;
    int i;
    maxx=getmaxx();
    maxy=getmaxy();
    maxx=maxx/2;
    maxy=maxy/2;
    i=0;
    while(i<n-1)
    {
        line(maxx+c[i][0],maxy-c[i][1],maxx+c[i+1][0],maxy-c[i+1][1]);
        i++;
    }
    i=n-1;
    line(maxx+c[i][0],maxy-c[i][1],maxx+c[0][0],maxy-c[0][1]);
    setcolor(GREEN);
    line(0,maxy,maxx*2,maxy);
    line(maxx,0,maxx,maxy*2);
    setcolor(WHITE);
}
void mul(int n,float b[][3],float c[][3],float a[][3])
{
    int i,j,k;
    for(i=0; i<n; i++)
        for(j=0; j<3; j++)
            a[i][j]=0;
    for(i=0; i<n; i++)
        for(j=0; j<3; j++)
            for(k=0; k<3; k++)
            {
                a[i][j]=a[i][j]+(c[i][k]*b[k][j]);
            }
}
void translation(int n,float c[][3],float tx,float ty)
{
    int i;
    for(i=0; i<n; i++)
    {
```

```

        c[i][0]=c[i][0]+tx;
        c[i][1]=c[i][1]+ty;
    }
}
void scaling(int n,float c[][3],float sx,float sy)
{
    float b[10][3],a[10][3];
    int i,j;
    for(j=0; j<3; j++)
        b[i][j]=0;
    b[0][0]=sx;
    b[1][1]=sy;
    b[2][2]=1;
    mul(n,b,c,a);
    setcolor(RED);
    disp(n,a);
}
void rotation(int n,float c[][3],float ra)
{
    int i=0,j;
    float b[10][3],xp,yp,a[10][3];
    xp=c[0][0];
    yp=c[0][1];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
    b[0][0]=b[1][1]=cos(ra*3.14/180);
    b[0][1]=sin(ra*3.14/180);
    b[1][0]=-sin(ra*3.14/180);
    b[2][0]=(-xp*cos(ra*3.14/180))+(yp*sin(ra*3.14/180))+xp;
    b[2][1]=(-xp*sin(ra*3.14/180))-(yp*cos(ra*3.14/180))+yp;
    b[2][2]=1;
    mul(n,b,c,a);
    setcolor(RED);
    disp(n,a);
}
void refthx(int n,float c[][3])
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
    b[0][0]=b[2][2]=1;
    b[1][1]=-1;
    mul(n,b,c,a);

```

```

    setcolor(RED);
    disp(n,a);
}
void refthy(int n,float c[][3])
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
    b[1][1]=b[2][2]=1;
    b[0][0]=-1;
    mul(n,b,c,a);
    setcolor(RED);
    disp(n,a);
}
void reforge(int n,float c[][3])
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
    b[0][0]=b[1][1]=-1;
    b[2][2]=1;
    mul(n,b,c,a);
    setcolor(RED);
    disp(n,a);
}
void refthyx(int n,float c[][3])
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
    b[0][1]=b[1][0]=b[2][2]=1;
    mul(n,b,c,a);
    setcolor(RED);
    disp(n,a);
}
void refthynege(int n,float c[][3])
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)

```

```

        for(j=0; j<3; j++)
            b[i][j]=0;
        b[0][1]=b[1][0]=-1;
        b[2][2]=1;
        mul(n,b,c,a);
        setcolor(RED);
        disp(n,a);
    }
void shearx(int n,float c[][3],float shx)
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
        b[0][0]=b[1][1]=b[2][2]=1;
        b[1][0]=shx;
        mul(n,b,c,a);
        setcolor(RED);
        disp(n,a);
    }
void sheary(int n,float c[][3],float shy)
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
        b[0][0]=b[1][1]=b[2][2]=1;
        b[0][1]=shy;
        mul(n,b,c,a);
        setcolor(RED);
        disp(n,a);
    }
int main()
{
    int i,j,k,cho,n,gd=DETECT,gm;
    float c[10][3],tx,ty,sx,sy,ra;
    initgraph(&gd,&gm," ");
    cout<<"Enter no. of vertices \t";
    cin>>n;
    for (i=0; i<n; i++)
    {
        cout<<"Enter coordinates of vertex \n",i+1;
        cin>>c[i][0]>>c[i][1];
        c[i][2]=1;
    }

```

```

}
do
{
    cleardevice();
    cout<<"\n\t\t MENU ";
    cout<<"\n\t\t 1)Translation";
    cout<<"\n\t\t 2)Scaling";
    cout<<"\n\t\t 3)Rotation";
    cout<<"\n\t\t 4)Reflection";
    cout<<"\n\t\t 5)Shear";
    cout<<"\n\t\t 6)EXIT";
    cout<<"\n\t Enter your Choice";
    cin>>cho;
    switch(cho)
    {
    case 1:
        cout<<"\n\t Enter translation factor for X and Y axis:\t\n";
        cin>>tx>>ty;
        cleardevice();
        setcolor(15);
        disp(n,c);
        translation(n,c,tx,ty);
        setcolor(15);
        disp(n,c);
        getch();
        break;
    case 2:
        cout<<"\n\t Enter scaling factor for X and Y axis:\t";
        cin>>sx>>sy;
        cleardevice();
        setcolor(15);
        disp(n,c);
        scaling(n,c,sx,sy);
        getch();
        break;
    case 3:
        cout<<"\n\t Enter rotation angle :\t";
        cin>>ra;
        cleardevice();
        disp(n,c);
        rotation(n,c,ra);
        getch();
        break;
    case 4:
        int ch;
        do

```

```

{
    cleardevice();
    cout<<"\n\t\t MENU ";
    cout<<"\n\t\t 1)Reflection about x axis";
    cout<<"\n\t\t 2)Reflection about x axis";
    cout<<"\n\t\t 3)Reflection about origin";
    cout<<"\n\t\t 4)Reflection about the line y=x";
    cout<<"\n\t\t 4)Reflection about the line y=-x";
    cout<<"\n\t Enter your Choice :";
    cin>>ch;
    switch(ch)
    {
    case 1:
        cleardevice();
        setcolor(15);
        disp(n,c);
        refthx(n,c);
        getch();
        break;
    case 2:
        cleardevice();
        setcolor(15);
        disp(n,c);
        refthy(n,c);
        getch();
        break;
    case 3:
        cleardevice();
        setcolor(15);
        disp(n,c);
        reforg(n,c);
        getch();
        break;
    case 4:
        cleardevice();
        setcolor(15);
        disp(n,c);
        refthyx(n,c);
        getch();
        break;
    case 5:
        cleardevice();
        setcolor(15);
        disp(n,c);
        refthynegx(n,c);
        getch();
    }
}

```

```

        break;
    default:
        cout<<"\n\t Invalid Choice !!!";
        break;
    }
}
while(cho!=5);
case 5:
    int cha;
    float shx,shy;
    do
    {
        cleardevice();
        cout<<"\n\t\t MENU ";
        cout<<"\n\t\t 1)X-shear";
        cout<<"\n\t\t 2)Y-shear";
        cout<<"\n\t Enter your Choice: \n";
        cin>>cha;
        switch(cha)
        {
            case 1:
                cout<<"\n\t Enter Shear factor: \n";
                cin>>shx;
                cleardevice();
                setcolor(15);
                disp(n,c);
                shearx(n,c,shx);
                getch();
                break;
            case 2:
                cout<<"\n\t Enter Shear factor: \n";
                cin>>shy;
                cleardevice();
                setcolor(15);
                disp(n,c);
                shearx(n,c,shy);
                getch();
                break;
            default:
                cout<<"\n\t Invalid Choice !!!";
                break;
        }
    }
    while(cho!=2);
case 6:
    exit(o);

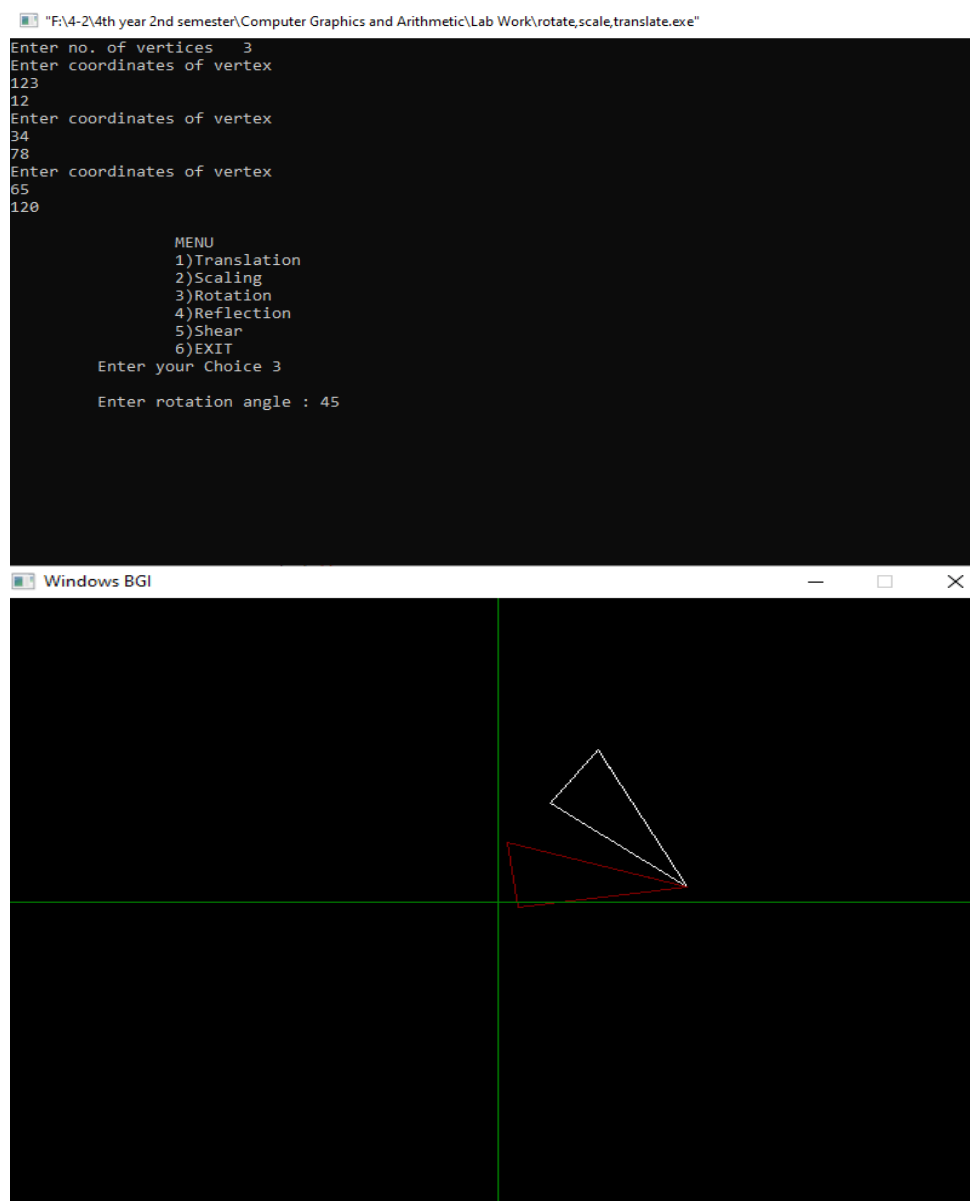
```

```

        break;
    default:
        cout<<"\n\t Invalid Choice !!!";
        break;
    }
}
while(cho!=6);
getch();
closegraph();
}

```

Output:



Experiment No.: 06

Experiment Name: The coordinates of an object is bounded by (1,1), (3,4), (5,7), and (10,3). The object is being rotated about a point (3, 4) by 30° in a clockwise direction and scaled by 2 units in X-direction and 3 units in Y-direction. Write down the code to show the final coordinates of the object.

Theory:

Rotation: Rotations in computer graphics are a transformational operation. That means that it is a conversion from one coordinate space onto another.

The rotation of a 2D vector anti-clockwise in a plane is done as follows:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Thus, $x' = x * \cos\theta - y * \sin\theta$
and $y' = x * \sin\theta + y * \cos\theta$

For clockwise rotation put $\theta = (-\theta)$

Hence, $x' = x * \cos\theta + y * \sin\theta$
and $y' = y * \cos\theta - x * \sin\theta$

Scaling: Matrix for Scaling,

$$S = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equations for scaling,

$$x' = x * Sx$$

$$y' = y * Sy$$

Code:

```
#include<stdio.h>
#include<math.h>
#include<graphics.h>
using namespace std;
pair<double, double> rotation_in_clockwise(double x1, double y1, double theta)
{
    double temp=x1;
    x1=x1*cos(theta)+y1*sin(theta);
    y1=y1*cos(theta)-temp*sin(theta);

    return {x1,y1};
}
```

```

int main()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"");

    double x1=1,y1=1,y2=4,x2=3,x3=5,y3=7,x4=10,y4=3;
    double pivot_x=3, pivot_y=4;
    int mdx=getmaxx()/2;
    int mdy=getmaxy()/2;
    int factor_for_display=10;
    mdx-=250;
    line(mdx+x1*factor_for_display,mdy-y1*factor_for_display,mdx+x2*factor_for_display,mdy-
y2*factor_for_display);
    line(mdx+x2*factor_for_display,mdy-y2*factor_for_display,mdx+x3*factor_for_display,mdy-
y3*factor_for_display);
    line(mdx+x3*factor_for_display,mdy-y3*factor_for_display,mdx+x4*factor_for_display,mdy-
y4*factor_for_display);
    line(mdx+x1*factor_for_display,mdy-y1*factor_for_display,mdx+x4*factor_for_display,mdy-
y4*factor_for_display);

    int textx=x1,texty=y1+80;

    outtextxy(textx,texty,"Original Shape:");
    //for clockwise rotation
    double theta=30;
    pair<double,double> pr=rotation_in_clockwise(x1-pivot_x,y1-pivot_y,theta);
    x1=pr.first+pivot_x;
    y1=pr.second+pivot_y;

    pr=rotation_in_clockwise(x2-pivot_x,y2-pivot_y,theta);
    x2=pr.first+pivot_x;
    y2=pr.second+pivot_y;

    pr=rotation_in_clockwise(x3-pivot_x,y3-pivot_y,theta);
    x3=pr.first+pivot_x;
    y3=pr.second+pivot_y;

    pr=rotation_in_clockwise(x4-pivot_x,y4-pivot_y,theta);
    x4=pr.first+pivot_x;
    y4=pr.second+pivot_y;
    mdx+=160;
    mdy+=20;
    line(mdx+x1*factor_for_display,mdy-y1*factor_for_display,mdx+x2*factor_for_display,mdy-
y2*factor_for_display);
    line(mdx+x2*factor_for_display,mdy-y2*factor_for_display,mdx+x3*factor_for_display,mdy-
y3*factor_for_display);

```

```

    line(mdx+x3*factor_for_display,mdy-y3*factor_for_display,mdx+x4*factor_for_display,mdy-
y4*factor_for_display);
    line(mdx+x1*factor_for_display,mdy-y1*factor_for_display,mdx+x4*factor_for_display,mdy-
y4*factor_for_display);

    outtextxy(textx+220,texty,"After Rotation:");

    //for scaling

    int x_scale_factor=2,y_scale_factor=3;
    x1=x1*x_scale_factor;
    y1=y1*y_scale_factor;

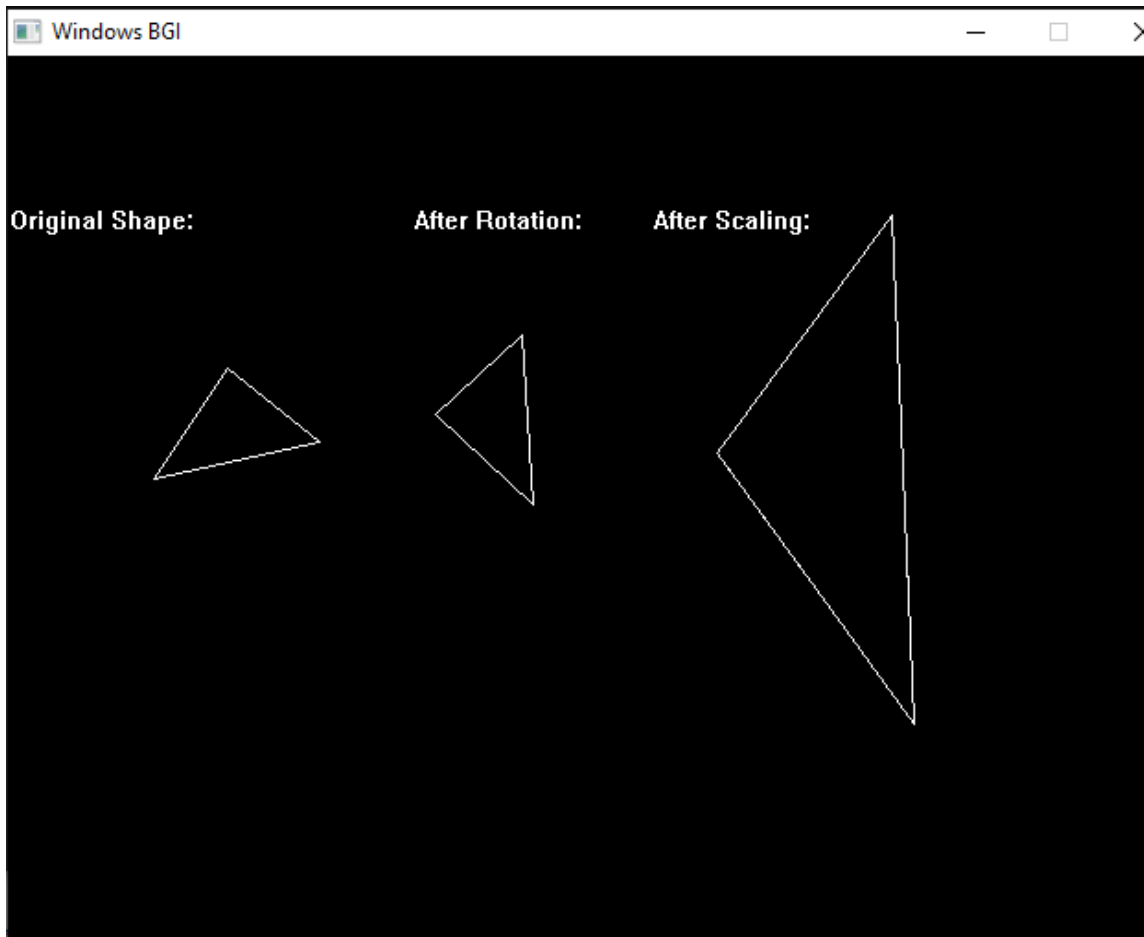
    x2=x2*x_scale_factor;
    y2=y2*y_scale_factor;

    x3=x3*x_scale_factor;
    y3=y3*y_scale_factor;

    x4=x4*x_scale_factor;
    y4=y4*y_scale_factor;
    mdx+=150;
    mdy+=150;
    line(mdx+x1*factor_for_display,mdy-y1*factor_for_display,mdx+x2*factor_for_display,mdy-
y2*factor_for_display);
    line(mdx+x2*factor_for_display,mdy-y2*factor_for_display,mdx+x3*factor_for_display,mdy-
y3*factor_for_display);
    line(mdx+x3*factor_for_display,mdy-y3*factor_for_display,mdx+x4*factor_for_display,mdy-
y4*factor_for_display);
    line(mdx+x1*factor_for_display,mdy-y1*factor_for_display,mdx+x4*factor_for_display,mdy-
y4*factor_for_display);
    outtextxy(textx+350,texty,"After Scaling:");
    //line(x1,y1,x2,y2);
    getch();
}

```

Output:



Experiment No.: 07

Experiment Name: Write a program to perform line clipping.

Theory:

This Algorithm was developed by Liang and Barsky. It is used for line clipping as it is more efficient than Cyrus Beck algorithm and Cohen Sutherland algorithm because it uses more efficient parametric equations to clip the given line.

These parametric equations are given as:

These parameter equations are:

$$x = x_1 + tdx$$

$$x = x_1 + tdx$$

$$y = y_1 + tdy, 0 \leq t \leq 1$$

$$y = y_1 + tdy, 0 \leq t \leq 1$$

$$\text{Where } dx = x_2 - x_1 \text{ \& } dy = y_2 - y_1$$

$$dx = x_2 - x_1 \text{ \& } dy = y_2 - y_1$$

Liang Barsky line clipping algorithm uses 4 inequalities with 2 parameters p & q which are defined in the algorithm below.

The Liang Barsky line clipping algorithm uses 4 inequalities and 2 parameters p&q, which are defined in the algorithm below.

Algorithm(Algorithm)

- Read 2 endpoints of line as p1 (x1, y1) & p2 (x2, y2).
- Read the two endpoints of the line as p1(x1, y1) and p2(x2, y2).
- Read 2 corners (left-top & right-bottom) of the clipping window as (xwmin, ywmin, xwmax, ywmax).
- Read the 2 corners (upper left corner and lower right corner) of the clipping window as (xwmin, ywmin, xwmax, ywmax).
- Calculate values of parameters pi and qi for i = 1, 2, 3, 4 such that
- Calculate the values of the parameters pi and qi for i = 1, 2, 3, 4, so that
 - p1 = -dx, q1 = x1 - xwmin
 - p1 = -dx, q1 = x1 - xwmin
 - p2 = dx, q2 = xwmax - x1
 - p2 = dx, q2 = xwmax - x1
 - p3 = -dy, q3 = y1 - ywmin
 - p3 = -dy, q3 = y1 - ywmin
 - p4 = dy, q4 = ywmax - y1
 - p4 = dy, q4 = ywmax - y1
- if pi = 0 then the line is parallel to i-th boundary
- If pi = 0, the line is parallel to the i-th boundary
 - if qi < 0 then line is completely outside boundary so discard line
 - If qi < 0, the line is completely outside the boundary, so the line is discarded
 - else, check whether line is horizontal or vertical and then check the line endpoints with the corresponding boundaries.

Otherwise, check whether the line is horizontal or vertical, and then check the end of the line with the corresponding boundary.

- Initialize t1 & t2 as
- Initialize t1 and t2 as
t1 = 0 & t2 = 1
t1 = 0 & t2 = 1
- Calculate values for q_i/p_i for $i = 1, 2, 3, 4$.
- Calculate the value of q_i / p_i when $i = 1, 2, 3, 4$.
- Select values of q_i/p_i where $p_i < 0$ and assign maximum out of them as t1.
- Choose the value of q_i / p_i in the case of $p_i < 0$, and assign the maximum value as t1.
- Select values of q_i/p_i where $p_i > 0$ and assign minimum out of them as t2.
- Choose the value of q_i / p_i in the case of $p_i > 0$, and specify the minimum value as t2.
- if ($t1 < t2$) { $xx1 = x1 + t1dx$
- If ($t1 < t2$) ($xx1 = x1 + t1dx$
xx2 = $x1 + t2dx$
xx2 = $x1 + t2dx$
yy1 = $y1 + t1dy$
yy1 = $y1 + t1dy$
yy2 = $y1 + t2dy$
yy2 = $y1 + t2dy$
line (xx1, yy1, xx2, yy2) }
Line(xx1, yy1, xx2, yy2)}
- Stop.
- stop.

Code:

```
#include<iostream>
#include<graphics.h>
using namespace std;
int main()
{
    int i,gd=DETECT,gm;
    int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2,dx,dy;
    float t1,t2,p[4],q[4],temp;

    x1=120;
    y1=120;
    x2=300;
    y2=300;

    xmin=100;
    ymin=100;
    xmax=250;
    ymax=250;
```

```

initgraph(&gd,&gm,NULL);
rectangle(xmin,ymin,xmax,ymax);
line(x1, y1, x2, y2);
delay(1000);
cleardevice();

dx=x2-x1;
dy=y2-y1;

p[0]=-dx;
p[1]=dx;
p[2]=-dy;
p[3]=dy;

q[0]=x1-xmin;
q[1]=xmax-x1;
q[2]=y1-ymin;
q[3]=ymax-y1;

rectangle(xmin,ymin,xmax,ymax);
delay(200);

for(i=0; i<4; i++)
{
    if(p[i]==0)
    {
        cout<<"line is parallel to one of the clipping boundary";
        if(q[i]>=0)
        {
            if(i<2)
            {
                if(y1<ymin)
                {
                    y1=ymin;
                }

                if(y2>ymax)
                {
                    y2=ymax;
                }

                line(x1,y1,x2,y2);
            }

            if(i>1)
            {

```

```

        if(x1<xmin)
        {
            x1=xmin;
        }

        if(x2>xmax)
        {
            x2=xmax;
        }

        line(x1,y1,x2,y2);
    }
}
}

t1=0;
t2=1;

for(i=0; i<4; i++)
{
    temp=q[i]/p[i];

    if(p[i]<0)
    {
        if(t1<=temp)
            t1=temp;
    }
    else
    {
        if(t2>temp)
            t2=temp;
    }
}

if(t1<t2)
{
    xx1 = x1 + t1 * p[1];
    xx2 = x1 + t2 * p[1];
    yy1 = y1 + t1 * p[3];
    yy2 = y1 + t2 * p[3];
    line(xx1,yy1,xx2,yy2);
}

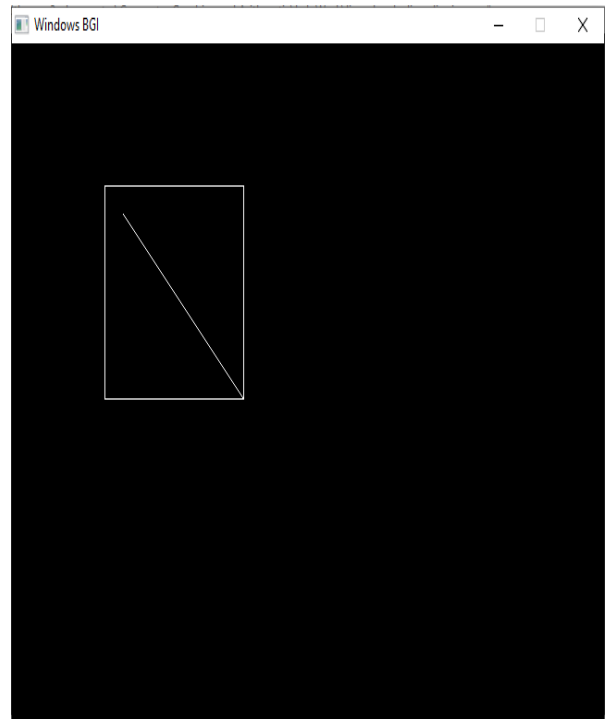
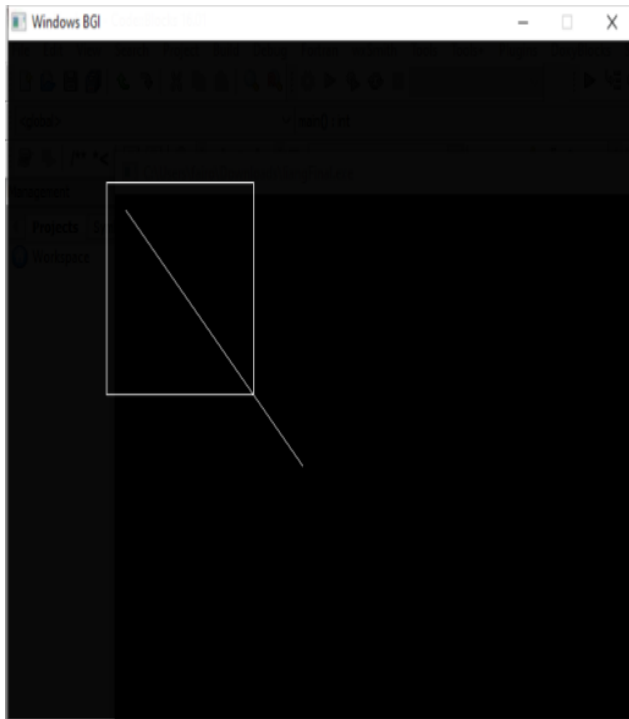
delay(500);
getch();

```



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

Output:



Experiment No.: 08

Experiment Name: Write a program to shear a hexagon along

i. X-axis

ii. Y-axis

Theory:

In a two-dimensional plane, the object size can be changed along X direction as well as Y direction.

So, there are two versions of shearing-

1. Shearing in X direction
2. Shearing in Y direction

Consider a point object o has to be sheared in a 2D plane.

Let-

- Initial coordinates of the object O = (xold, yold)
- Shearing parameter towards X direction = shx
- Shearing parameter towards Y direction = shy
- New coordinates of the object o after shearing = (xnew, ynew)

Shearing in X axis is achieved by using the following shearing equations-

$$x_{new} = x_{old} + shx \times y_{old}$$

$$y_{new} = y_{old}$$

Shearing in Y axis is achieved by using the following shearing equations-

$$x_{new} = x_{old}$$

$$y_{new} = y_{old} + shy \times x_{old}$$

Code:

```
#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<math.h>
using namespace std;
void disp(int n,float c[][3])
{
    float maxx,maxy;
    int i;
    maxx=getmaxx();
```

```

maxy=getmaxy();
maxx=maxx/2;
maxy=maxy/2;
i=0;
while(i<n-1)
{
    line(maxx+c[i][0],maxy-c[i][1],maxx+c[i+1][0],maxy-c[i+1][1]);
    i++;
}
i=n-1;
line(maxx+c[i][0],maxy-c[i][1],maxx+c[0][0],maxy-c[0][1]);
setcolor(GREEN);
line(0,maxy,maxx*2,maxy);
line(maxx,0,maxx,maxy*2);
setcolor(WHITE);
}
void mul(int n,float b[][3],float c[][3],float a[][3])
{
    int i,j,k;
    for(i=0; i<n; i++)
        for(j=0; j<3; j++)
            a[i][j]=0;
    for(i=0; i<n; i++)
        for(j=0; j<3; j++)
            for(k=0; k<3; k++)
            {
                a[i][j]=a[i][j]+(c[i][k]*b[k][j]);
            }
}

void shearx(int n,float c[][3],float shx)
{
    int i=0,j;
    float b[10][3],a[10][3];
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            b[i][j]=0;
    b[0][0]=b[1][1]=b[2][2]=1;
    b[1][0]=shx;
    mul(n,b,c,a);
    setcolor(RED);
    disp(n,a);
}
void sheary(int n,float c[][3],float shy)
{

```

```

int i=0,j;
float b[10][3],a[10][3];
for(i=0; i<3; i++)
    for(j=0; j<3; j++)
        b[i][j]=0;
b[0][0]=b[1][1]=b[2][2]=1;
b[0][1]=shy;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);
}
int main()
{
    int i,j,k,cho,n,gd=DETECT,gm;
    float c[10][3],tx,ty,sx,sy,ra;
    initgraph(&gd,&gm," ");
    cout<<"Enter no. of vertices \t";
    cin>>n;
    for (i=0; i<n; i++)
    {
        cout<<"Enter coordinates of vertex \n",i+1;
        cin>>c[i][0]>>c[i][1];
        c[i][2]=1;
    }

    int cha;
    float shx,shy;
    do
    {
        cleardevice();
        cout<<"\n\t\t MENU ";
        cout<<"\n\t\t 1)X-shear";
        cout<<"\n\t\t 2)Y-shear";
        cout<<"\n\t Enter your Choice: \t";
        cin>>cha;
        switch(cha)
        {
            case 1:
                cout<<"\n\t Enter Shear factor: \t";
                cin>>shx;
                cleardevice();
                setcolor(15);
                disp(n,c);
                shearx(n,c,shx);
                getch();
                break;

```

```

case 2:
    cout<<"\n\t Enter Shear factor: \t";
    cin>>shy;
    cleardevice();
    setcolor(15);
    disp(n,c);
    shearx(n,c,shy);
    getch();
    break;
default:
    cout<<"\n\t Invalid Choice !!!";
    break;
}
}
while(cho!=2);

getch();
closegraph();
}

```

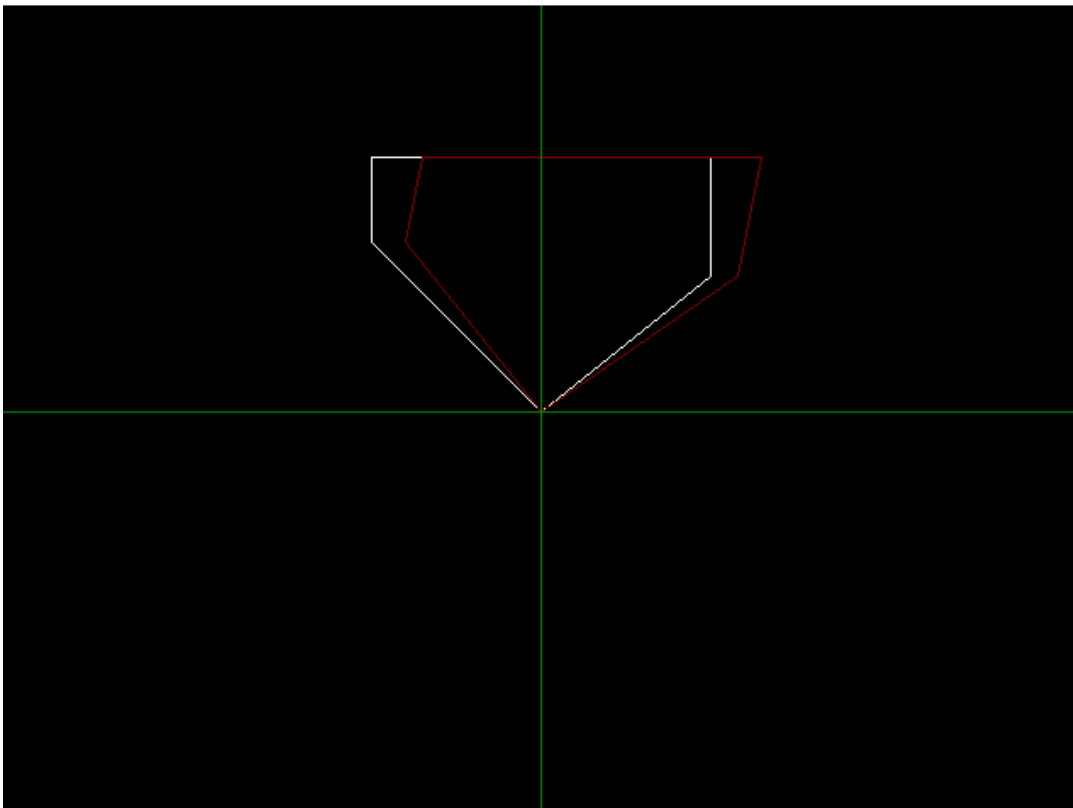
Output:

"F:\4-2\4th year 2nd semester\Computer Graphics and Arithmetic\Lab Work\shearing.exe"

```
Enter no. of vertices 6
Enter coordinates of vertex
0
150
Enter coordinates of vertex
100
150
Enter coordinates of vertex
100
80
Enter coordinates of vertex
0
0
Enter coordinates of vertex
-100
100
Enter coordinates of vertex
-100
150
```

```
        MENU
        1)X-shear
        2)Y-shear
Enter your Choice: 1
Enter Shear factor: .2
```

Windows BGI



Experiment No.: 09

Experiment Name: Write a program to implement Bezier Curve.

Theory:

The curve is drawn by using control points. In this, Approximate tangents act as control points that are used to generate the desired Bezier. It is a parametric curve that follows the Bernstein polynomial as the basic function.

Properties:

- ✓ Bezier curve is always contained within a polygon called the convex hull of its control points.
- ✓ Bezier's curve generally follows the shape of its defining polygon.
- ✓ The first and last points of the curve are coincident with the first and last points of the defining polygon.
- ✓ The degree of the polynomial defining the curve segment is one less than the total number of control points.
- ✓ The order of the polynomial defining the curve segment is equal to the total number of control points.
- ✓ Bezier curve exhibits the variation diminishing property.
- ✓ It means the curve does not oscillate about any straight line more often than the defining polygon.

Bezier Curve Equation-

$$P(t) = \sum_{i=0}^n B_i J_{n,i}(t)$$

Bezier Curve Equation

Here,

- t is any parameter where $0 \leq t \leq 1$
- $P(t)$ = Any point lying on the bezier curve
- B_i = i th control point of the bezier curve
- n = degree of the curve
- $J_{n,i}(t)$ = Blending function = $C(n,i)t^i(1-t)^{n-i}$ where $C(n,i) = n! / i!(n-i)!$

Code:

```
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
int 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);
    }

    for(t=0.0; t<=1.0; t=t+0.001)
    {
        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);
    }
    getch();
    closegraph();
}

```

Output:



Experiment No.: 10

Experiment Name: Write a program to implement animation using C function.

Theory:

In this program, we first draw a yellow color ball on a screen having a center at (x, y) and then erases it using `cleardevice` function. We again draw this ball at the center (x, y + 5), or (x, y - 5) depending upon whether a ball is moving down or up. This will look like a bouncing ball. We will repeat above steps until the user press any key on the keyboard.

We will use the below-mentioned functions in this program.

Initgraph : It initializes the graphics system by loading the passed graphics driver then changing the system into graphics mode.

Getmaxx : It returns the maximum X coordinate in current graphics mode and driver.

Setcolor : It changes the current drawing color. The default color is white. Each color is assigned a number like BLACK is 0 and RED is 4. Here we are using color constants defined inside `graphics.h` header file.

Setfillstyle : It sets the current fill pattern and fills color.

Circle : It draws a circle with radius r and center at (x, y).

Floodfill : It is used to fill a closed area with the current fill pattern and fill color. It takes any point inside a closed area and the color of the boundary as input.

Cleardevice : It clears the screen, and sets the current position to (0, 0).

Kbit : It is used to determine whether a key is pressed or not. It returns a non-zero value if a key is pressed otherwise zero.

Delay : It is used to suspend the execution of a program for an M millisecond.

Closegraph : It unloads the graphics drivers and sets the screen back to text mode.

Code:

```
#include<graphics.h>
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
using namespace std;
int main()
{

    int gd = DETECT, gm;
    int i, x, y, flag=0;
    initgraph(&gd, &gm, "C:\\TC\\BGI");

    /* get mid positions in x and y-axis */
    x = getmaxx()/2;
    y = 30;
```

```

while (!kbhit())
{
    if(y >= getmaxy()-30 || y <= 30)
        flag = !flag;
    /* draws the gray board */
    setcolor(YELLOW);
    setfillstyle(SOLID_FILL, YELLOW);
    circle(x, y, 30);
    floodfill(x, y, YELLOW);

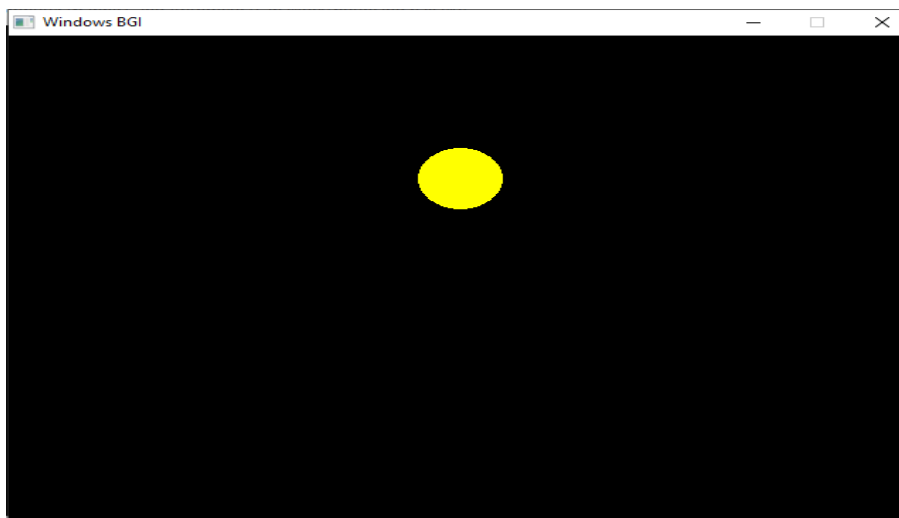
    /* delay for 30 milli seconds */
    delay(30);

    /* clears screen */
    cleardevice();
    if(flag)
    {
        y = y + 5;
    }
    else
    {
        y = y - 5;
    }
}

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

```

Output:



Experiment No.: 11

Experiment Name: Write a program for man object moving.

Theory:

A simple code to make an animation using the basic build-in function of graphics.h library. The animation will show moving a person in a particular direction with an umbrella.

Code:

```
#include<graphics.h>
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
using namespace std;
int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TC\\BGI");

    int i,x,y,j;

    // man
    for(j=1; j<600; j=j+5)
    {
        line(0,400,800,400);
        circle(30+j,280,20); //head
        line(30+j,300,30+j,350); //body
        line(30+j,330,70+j,330); //hand
        line(30+j,335,70+j,339); //hand
        if(j%2==0)
        {
            line(30+j,350,25+j,400); //left leg
            line(30+j,350,10+j,400); // right
        }
        else
        {
            line(30+j,350,35+j,400); //transition
            delay(20);
        }
    }
    ///umbrela
    line(70+j,190,70+j,350);
    pieslice(70+j,200,0,180,100);
    // rain
    for(i=0; i<300; i++)
```

```

{
    x= rand()%800+1;
    y= rand()%800+1;
    outtextxy(x,y,"/");
}
delay(170);
cleardevice();
}
getch();
closegraph();
}

```

Output:

