Computer Graphics

Indraprastha College for Women
University of Delhi

B. Sc. (CS) 3rd Year

(2019 – 22)

Submitted By

Name: Pragati Chaudhary

Roll No: 19/CS/31

Mentor

Ms. Nikita Jain

CONTENTS: -

1. (i) Write a program to implement Bresenham's line drawing algorithm.

CODE:

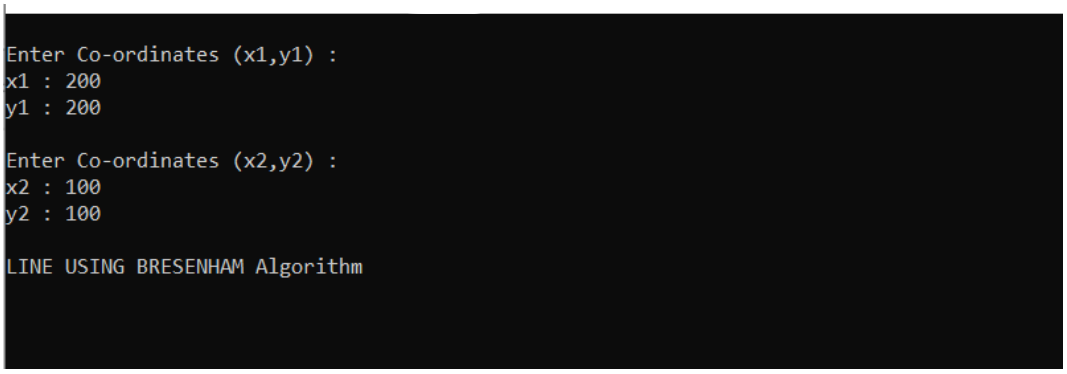
```
#include<iostream>
#include<graphics.h>
#include<conio.h>
using namespace std;

//Bresenham function to implement bresenham's algo for 0<m<1
void Bresenham(int xa, int ya, int xb, int yb)
{
  //calculating the constants dx and dy
  int dx = abs (xa - xb), dy = abs (ya - yb);

  //calculating the decision parameter
  int p = 2 * dy - dx;
  int twoDy = 2 * dy, twoDyDx = 2*(dy - dx);
  int x, y, xEnd,x_mid,y_mid;

  x_mid=getmaxx()/2;
  y_mid=getmaxy()/2;

  //determinig which point to use as start which as end
  if(xa > xb)
  {
    x = xb;
    y = yb;
    xEnd = xa;
  }
```

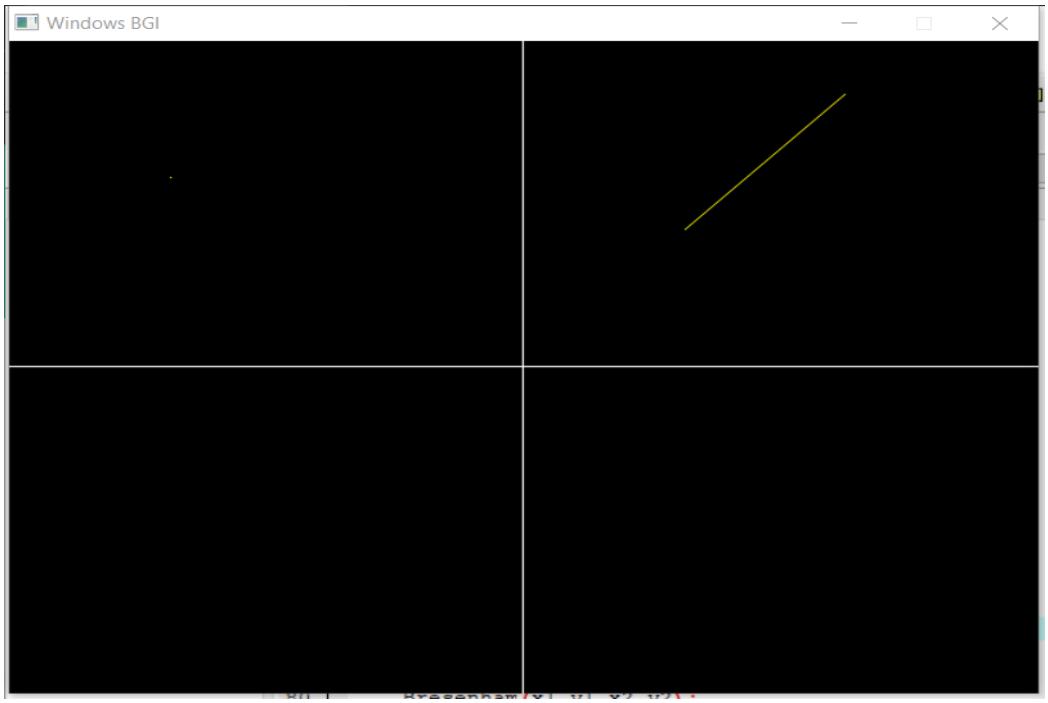
```
   else
   {
      x = xa;
      y = ya;
      xEnd = xb;
   }

   putpixel (x, y, YELLOW);
   while (x < xEnd)
   {
      x++;
      if(p < 0)
                        {
         p += twoDy; //calculating p for next point
      }
      else
     {
         y++;
         p += twoDyDx; //calculating p for next point
     }
   putpixel (x_mid+x,y_mid-y, YELLOW);
   }
}

//main function
int main()
{
   int gdriver = DETECT , gmode,error;
   initgraph(&gdriver, &gmode, (char*)"");

            int x1,y1,x2,y2,x_mid,y_mid;

   //input for  points of the line
   cout<<"\nEnter Co-ordinates (x1,y1) :";
   cout<<"\nx1 : ";
   cin>>x1;
   cout<<"y1 : ";
   cin>>y1;

   cout<<"\nEnter Co-ordinates (x2,y2) :";
   cout<<"\nx2 : ";
   cin>>x2;
   cout<<"y2 : ";
   cin>>y2;

   cout<<"\nLINE USING BRESENHAM Algorithm";

//Creating the quadrants of the graph
   x_mid=getmaxx()/2;
   y_mid=getmaxy()/2;
```

```
  line(x_mid,0,x_mid,getmaxy());
  line(0,y_mid,getmaxx(),y_mid);

  //calling the bresenham_algo function
  Bresenham(x1,y1,x2,y2);

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

OUTPUTS:

I - Quadrant





III - Quadrant

```
Enter Co-ordinates (x1,y1) :
x1 : -50
y1 : -100

Enter Co-ordinates (x2,y2) :
x2 : -100
y2 : -200

LINE USING BRESENHAM Algorithm
```

(ii) Write a program to implement DDA line drawing algorithm.

CODE:

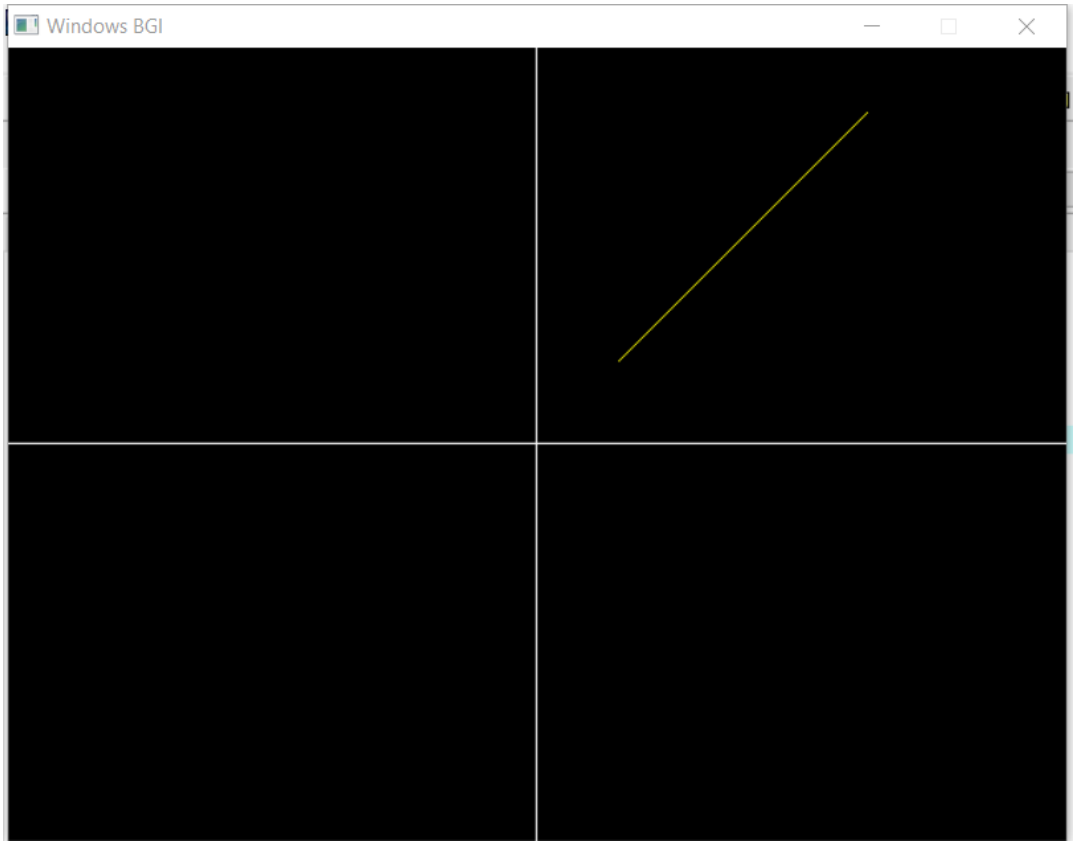
```
#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<math.h>

#define ROUND(a)((int)(a+0.5))
using namespace std;

// Variables for changing the origin
float x_mid, y_mid;

//lineDAA function to implement DDA algorithm...
int lineDDA(int xa, int ya, int xb, int yb)
{
	//calculating dx and dy
	int dx = xb-xa;
	int dy = yb-ya;
	int steps, k;
	float xIncr, yIncr;
	float x = xa, y = ya;

	// difference(dx,dy) with the greater magnitude determines the value of parameter steps
	if(abs(dx) > abs(dy))
	{
		steps = abs(dx);
	}
	else
	{
		steps=abs(dy);
	}
	xIncr = dx/(float) steps;
	yIncr = dy/(float) steps;

	putpixel(ROUND(x) + x_mid, y_mid - ROUND(y), YELLOW);

	//looping to generate next pixcel position step times
	for(k = 0; k < steps; k++)
	{
		x += xIncr;
		y += yIncr;
		putpixel(ROUND(x) + x_mid, y_mid - ROUND(y), YELLOW);
	}
	return 0;
}
```

```
//main function
int main()
{
  int gd= DETECT, gmode;
  initgraph(&gd,&gmode, "");

  //shifting the origin to the middle of screen
  float X = getmaxx(), Y = getmaxy();
  x_mid = X/2;
  y_mid = Y/2;

  int x1, y1, x2, y2;

  cout<<"\nEnter Co-ordinates (x1,y1) :";
  cout<<"\nx1 : ";
  cin>>x1;
  cout<<"y1 : ";
  cin>>y1;

  cout<<"\nEnter Co-ordinates (x2,y2) :";
  cout<<"\nx2 : ";
  cin>>x2;
  cout<<"y2 : ";
  cin>>y2;
  cout<<"\nLINE USING DDA Algorithm";

  line(x_mid, 0, x_mid, Y);
  line(0, y_mid, X, y_mid);

  //calling the lineDAA fuction
  lineDDA(x1,y1,x2,y2);

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

OUTPUTS:

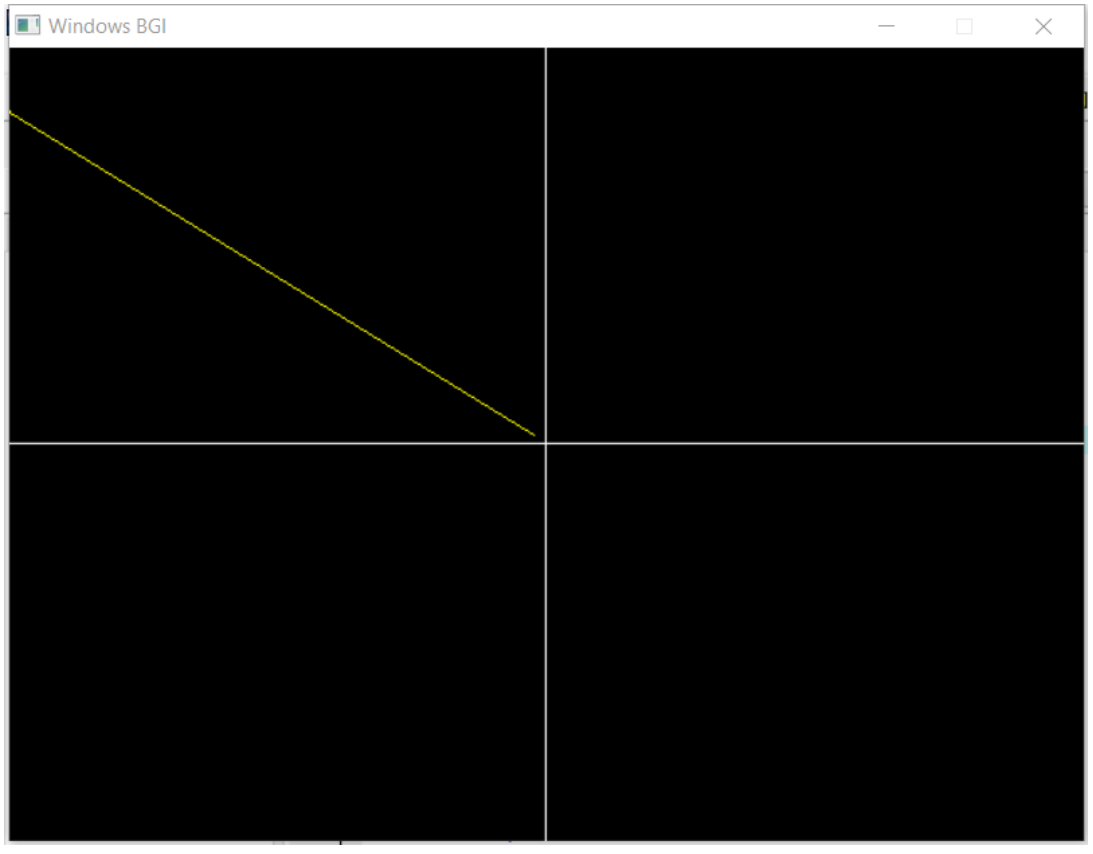
I - Quadrant

```
Enter Co-ordinates (x1,y1) :
x1 : 200
y1 : 200

Enter Co-ordinates (x2,y2) :
x2 : 50
y2 : 50

LINE USING DDA Algorithm
```



II - Quadrant

```
Enter Co-ordinates (x1,y1) :
x1 : -8
y1 : 5

Enter Co-ordinates (x2,y2) :
x2 : -800
y2 : 500

LINE USING DDA Algorithm
```

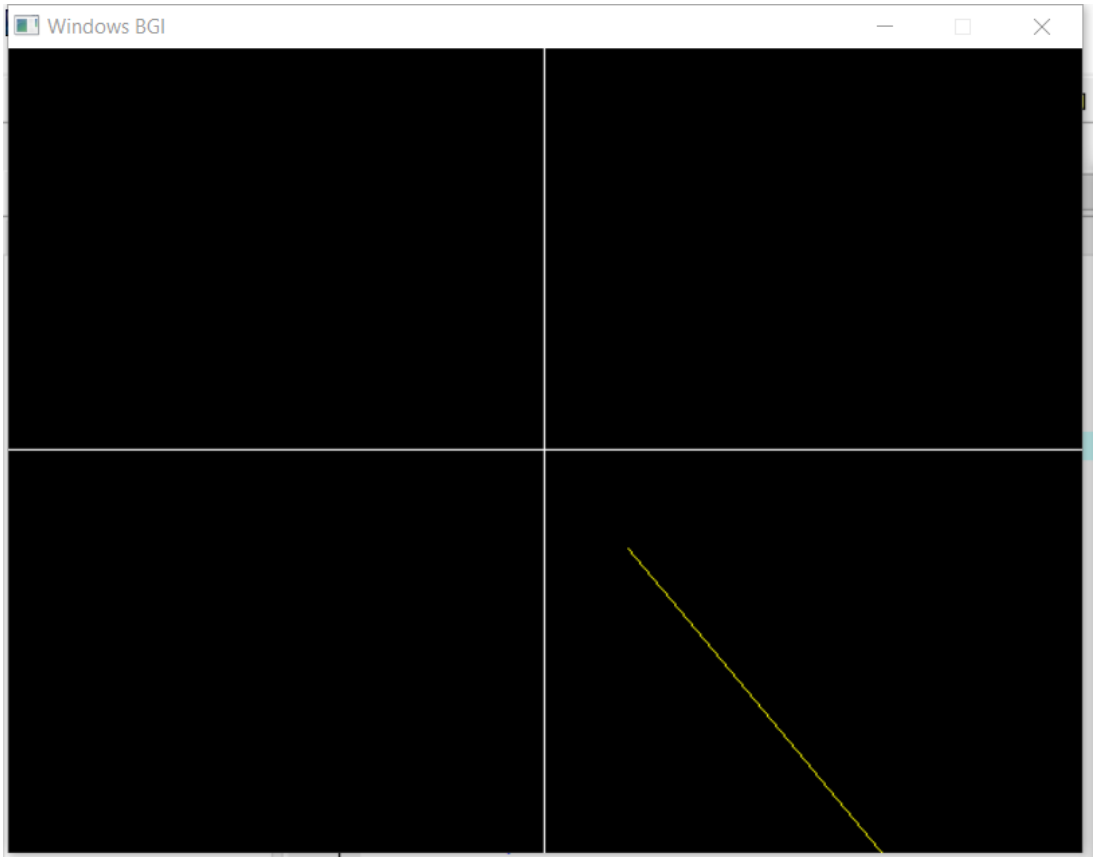


III - Quadrant

```
Enter Co-ordinates (x1,y1) :
x1 : -50
y1 : -50

Enter Co-ordinates (x2,y2) :
x2 : -100
y2 : -100

LINE USING DDA Algorithm
```

IV - Quadrant

```
Enter Co-ordinates (x1,y1) :
x1 : 50
y1 : -60

Enter Co-ordinates (x2,y2) :
x2 : 500
y2 : -600

LINE USING DDA Algorithm
```



2. (i) Write a program to implement mid-point circle drawing algorithm.

CODE:

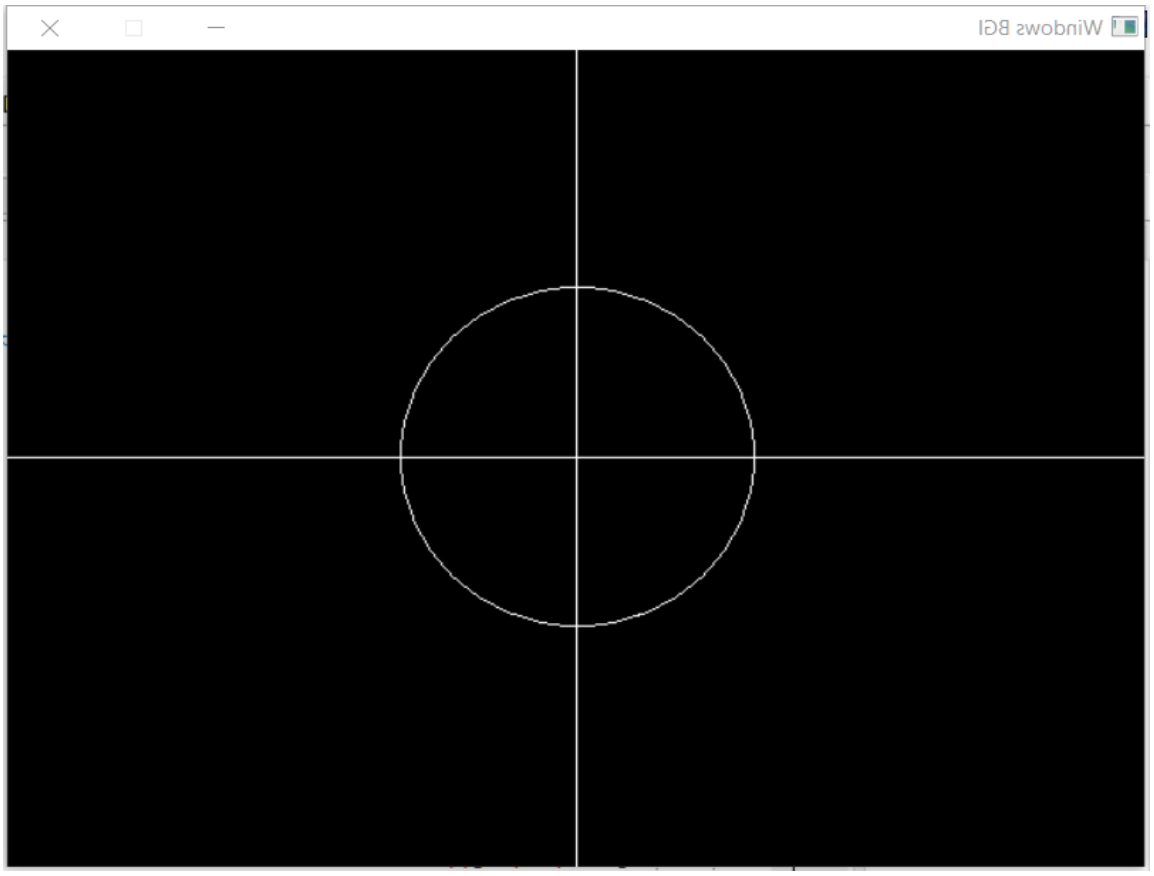
```
#include<iostream>
#include<graphics.h>
#include<math.h>

using namespace std;
void circlePlotPoints (int, int, int, int);
int x_mid,y_mid;

//Circle function to implement mid point circle's algorithm...
void Circle(int xCenter, int yCenter, int radius)
{
          int x = 0;
          int y = radius;
          int p = 1 - radius;

          // calculating all the perimeter points of the circle in the first octant
          while (x <= y)
          {
            //plotting first set of points
            circlePlotPoints (x, y, xCenter, yCenter);

           //if p lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside
we plot the pixel (x-1, y+1)
                    if (p < 0)
                    {
                      p += (2*x)+1;
                    }

                     else
          {
            p +=(2*(x-y))+1;
            y--;
     }
     x++ ;
          }
 }
// displaying the calculated points in the first octant along   with their mirror points in the other octants
void circlePlotPoints(int x, int y, int xCenter, int yCenter)
{
          putpixel (xCenter + x, yCenter + y, YELLOW);
          putpixel (xCenter - x, yCenter + y, YELLOW);
          putpixel (xCenter + x, yCenter - y, YELLOW);
          putpixel (xCenter - x, yCenter - y, YELLOW);
          putpixel (xCenter + y, yCenter + x, YELLOW);
          putpixel (xCenter - y, yCenter + x, YELLOW);
          putpixel (xCenter + y, yCenter - x, YELLOW);
```

```
            putpixel (xCenter - y, yCenter - x, YELLOW);
}

//main function
int main()
{
   int x , y;
   float r;
   int gd = DETECT , gm;
   initgraph(&gd, &gm, (char*)"");

   //inputs for the circle
   cout<<"\nEnter Co-ordinates (x,y) :";
   cout<<"\nx : ";
   cin>>x;
   cout<<"y : ";
   cin>>y;

   //radius of the circle
   cout<<"\n Enter the radius= ";
   cin>>r;

   cout<<"\nDRAWING CIRCLE USING MID POINT CIRCLE Algorithm";

   //Creating the quadrants of the graph
   x_mid = getmaxx()/2;
   y_mid = getmaxy()/2;
   line(x_mid , 0 , x_mid , getmaxy());
   line(0 , y_mid , getmaxx() , y_mid);

   //calling circle funtion
   circle(x + x_mid , y_mid - y , r);

    getch();
    closegraph();
    return 0;

}
```

OUTPUTS:

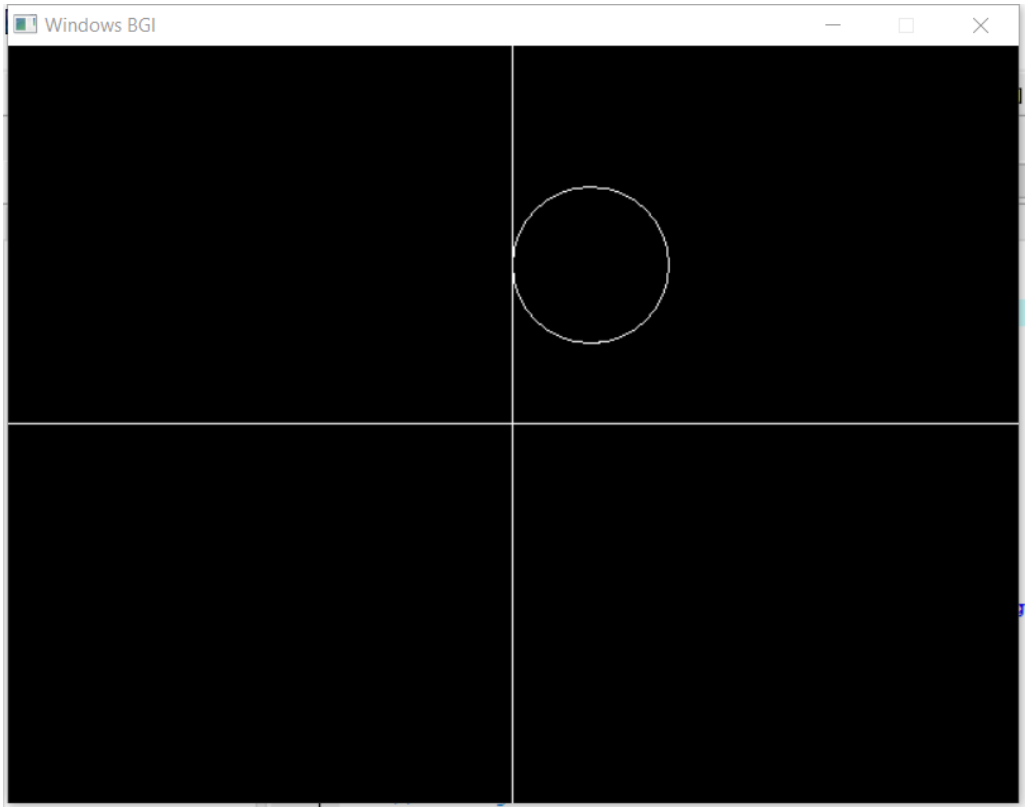
Circle at origin: (0,0)

```
Enter Co-ordinates (x,y) :
x : 0
y : 0

 Enter the radius= 100

DRAWING CIRCLE USING MID POINT CIRCLE Algorithm
```



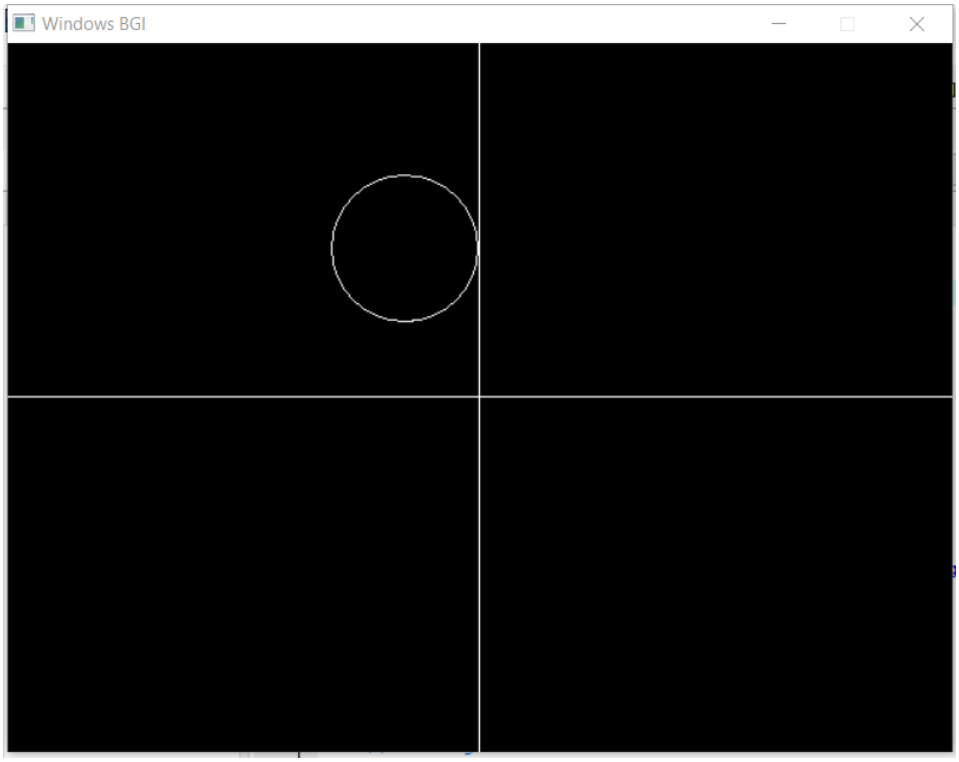
I - Quadrant

```
Enter Co-ordinates (x,y) :
x : 50
y : 100

 Enter the radius= 50

DRAWING CIRCLE USING MID POINT CIRCLE Algorithm
```



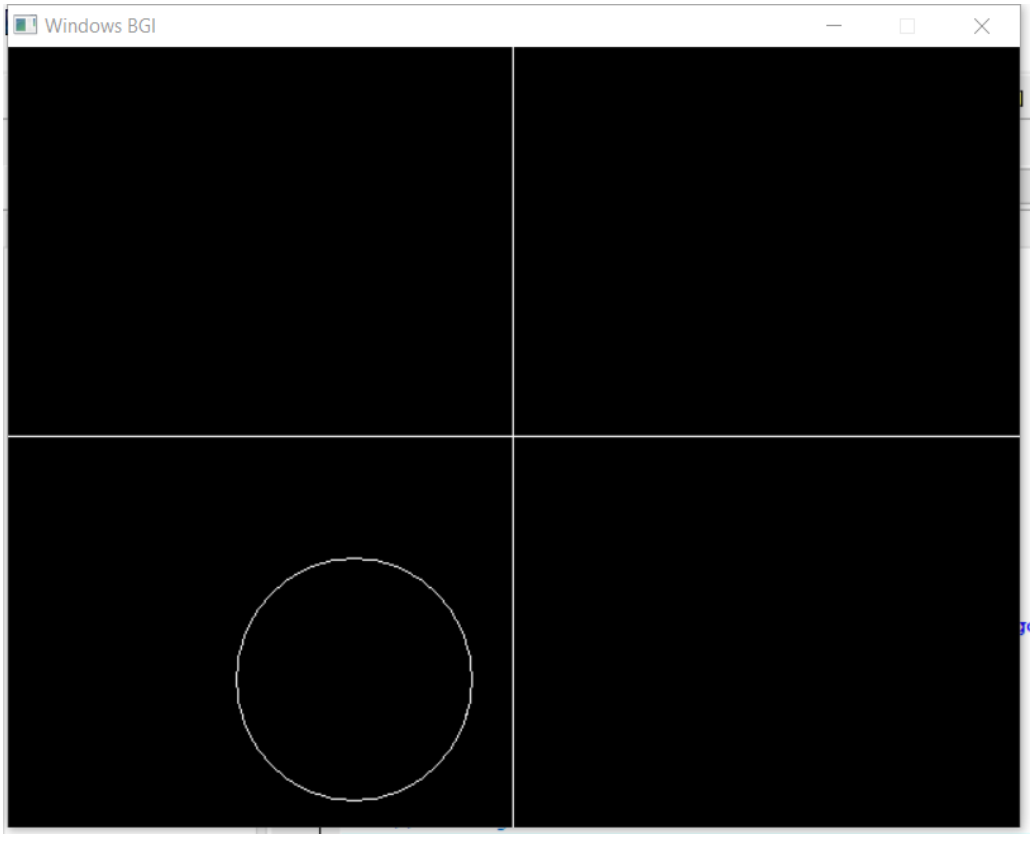
II - Quadrant

```
Enter Co-ordinates (x,y) :
x : -50
y : 100

 Enter the radius= 50

DRAWING CIRCLE USING MID POINT CIRCLE Algorithm
```



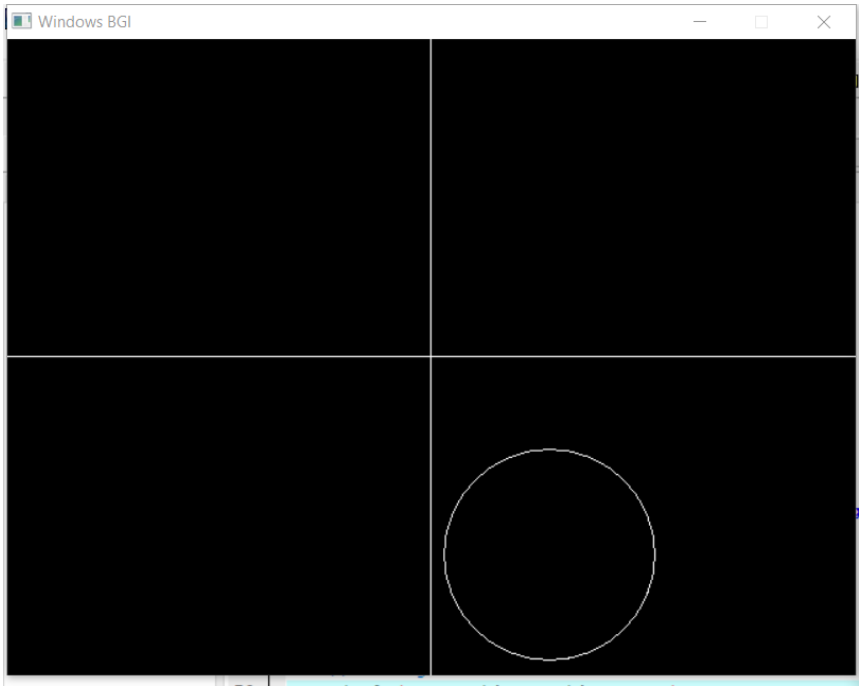
III- Quadrant

```
Enter Co-ordinates (x,y) :
x : -100
y : -150

 Enter the radius= 75

DRAWING CIRCLE USING MID POINT CIRCLE Algorithm
```



IV - Quadrant

```
Enter Co-ordinates (x,y) :
x : 90
y : -150

 Enter the radius= 80

DRAWING CIRCLE USING MID POINT CIRCLE Algorithm
```



(ii) Write a program to implement mid-point ellipse drawing algorithm.

CODE:

```
#include<iostream>
#include<graphics.h>
#include<math.h>

using namespace std;
#define ROUND(a) ((int) (a+0.5))

void ellipsePlotPoints(int, int, int, int);
int x_mid ,y_mid;

//Ellipse function to implement mid point circle's algorithm...
void Ellipse(int xCenter, int yCenter, int Rx, int Ry)
{
  int Rx2 = Rx*Rx;
  int Ry2 = Ry*Ry;
  int twoRx2 = 2*Rx2;
  int twoRy2 = 2*Ry2;
  int p;
  int x = 0;
  int y = Ry;
  int px = 0;
  int py = twoRx2 *y;

  //plotting first set of points
  ellipsePlotPoints(xCenter, yCenter, x, y);

  //Midpoint ellipse algorithm plots points of an ellipse on the first quadrant by dividing the quadrant into two regions

  //region 1
  p = ROUND(Ry2 - (Rx2 * Ry) + (0.25 * Rx2));

  while (px < py)
  {
   x++;
   px += twoRy2;

          if (p < 0)
   {
             p += Ry2 + px;
    }

          else
          {
                    y--;
```

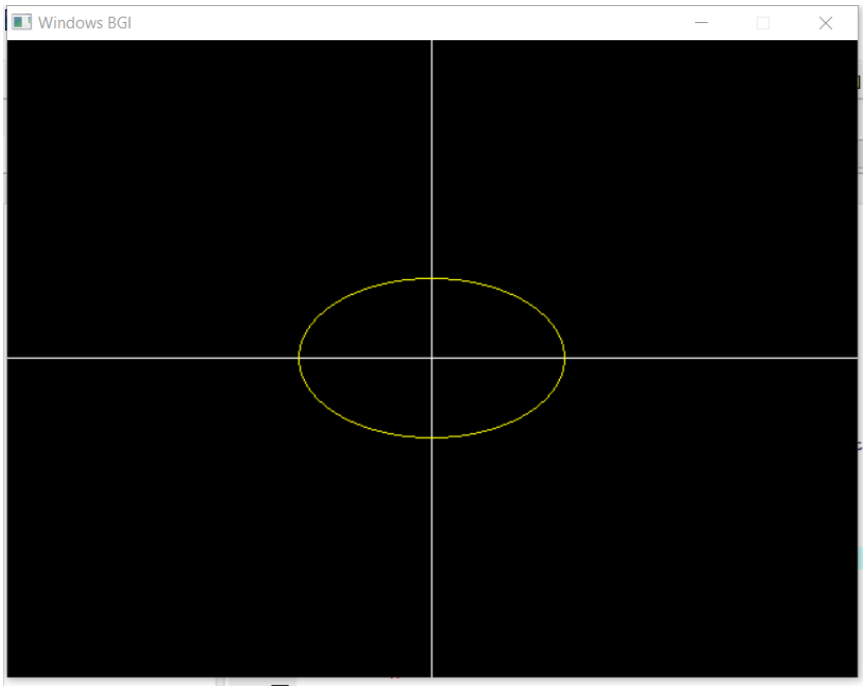
```
      py -= twoRx2;
      p += Ry2 + px - py;
            }
  ellipsePlotPoints(xCenter, yCenter, x,y);
 }
   //Region 2
   p = ROUND (Ry2*(x+0.5)*(x+0.5) + Rx2*(y-1)*(y-1) - Rx2*Ry2);
   while (y > 0)
           {
    y--;
    py -= twoRx2;

            if (p > 0)
   {
             p += Rx2 - py;
           }

           else
           {
      x++;
      px += twoRy2;
      p += Rx2 - py + px;
   }

    ellipsePlotPoints(xCenter, yCenter, x, y);
   }
 }

//function to plot points of ellipse in symmentry
void ellipsePlotPoints (int xCenter, int yCenter, int x, int y)
{
          putpixel (xCenter + x, yCenter + y, YELLOW);
          putpixel (xCenter- x, yCenter + y, YELLOW);
          putpixel (xCenter+ x, yCenter - y, YELLOW);
          putpixel (xCenter - x, yCenter - y, YELLOW);

}

//main function
int main()
{
          int x , y;
   float r,r2;
   int gd = DETECT , gm;
   initgraph(&gd, &gm, (char*)"");

   //inputs for the circle
   cout<<"\nEnter Co-ordinates (x,y) :";
   cout<<"\nx : ";
```

```
   cin>>x;
   cout<<"y : ";
   cin>>y;

   //radius of the circle
   cout<<"\n Enter the radius 1= ";
   cin>>r;

   cout<<"\n Enter the  radius 2= ";
   cin>>r2;

   cout<<"\nDRAWING ELLIPSE USING MID POINT ELLIPSE Algorithm";

  //creating the quadrants of the graph at the center.
   x_mid = getmaxx()/2;
   y_mid = getmaxy()/2;

   line(x_mid , 0 , x_mid , getmaxy());
   line(0 , y_mid , getmaxx() , y_mid);

   //calling of Ellipse function...
   Ellipse(x + x_mid , y_mid - y , r,r2);

    getch();
    closegraph();
    return 0;

}
```

OUTPUTS:

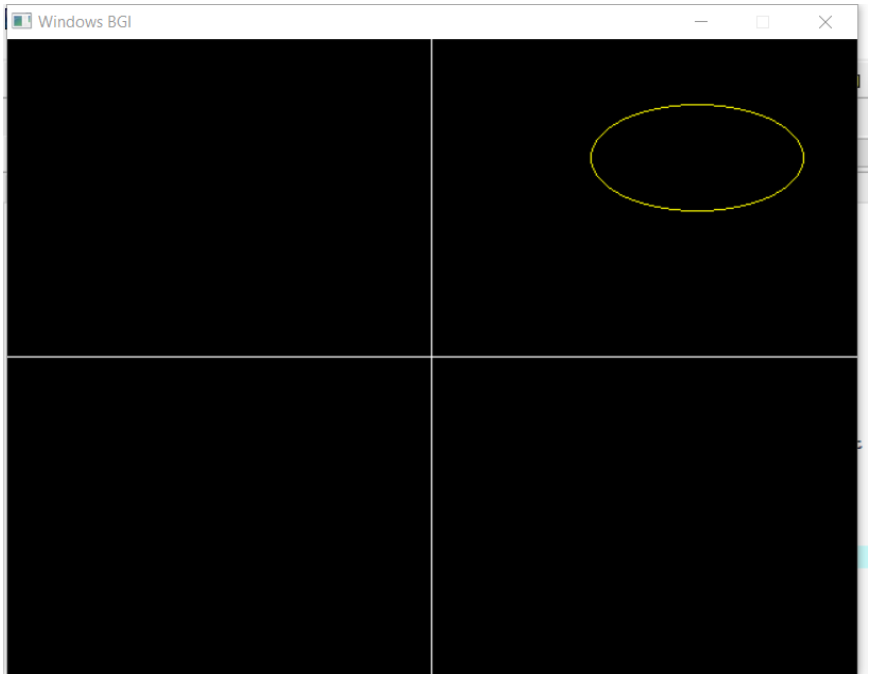
Ellipse at origin : (0,0)

```
Enter Co-ordinates (x,y) :
x : 0
y : 0

 Enter the radius 1= 100

 Enter the  radius 2= 60

DRAWING ELLIPSE USING MID POINT ELLIPSE Algorithm
```



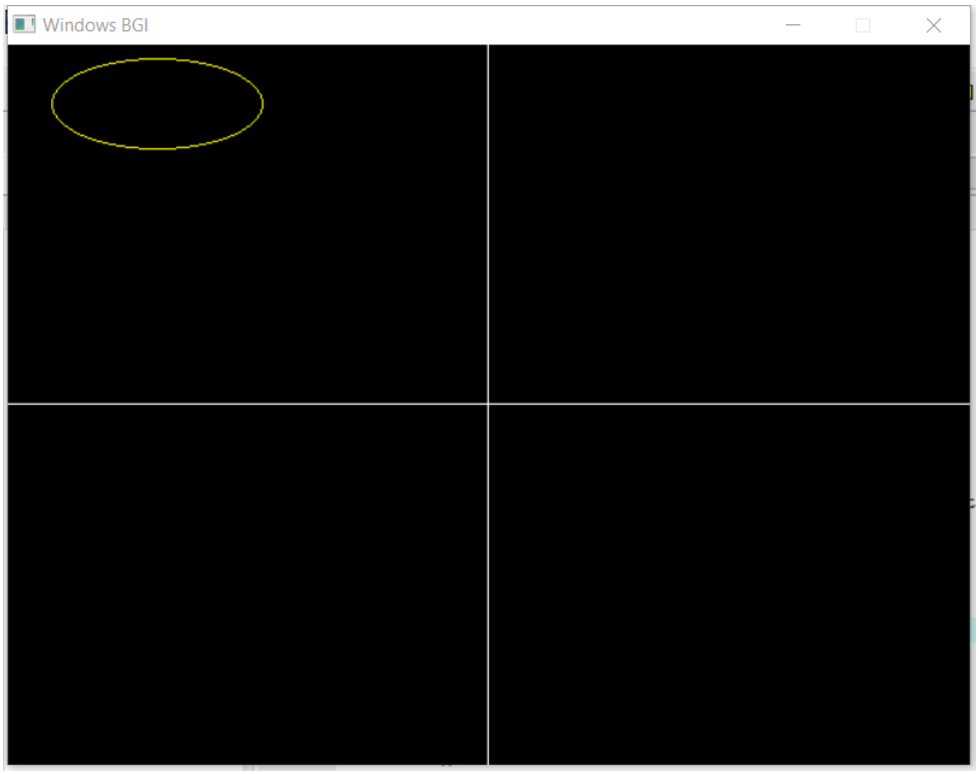
I – QUADRANT

```
Enter Co-ordinates (x,y) :
x : 200
y : 150

 Enter the radius 1= 80

 Enter the  radius 2= 40

DRAWING ELLIPSE USING MID POINT ELLIPSE Algorithm
```



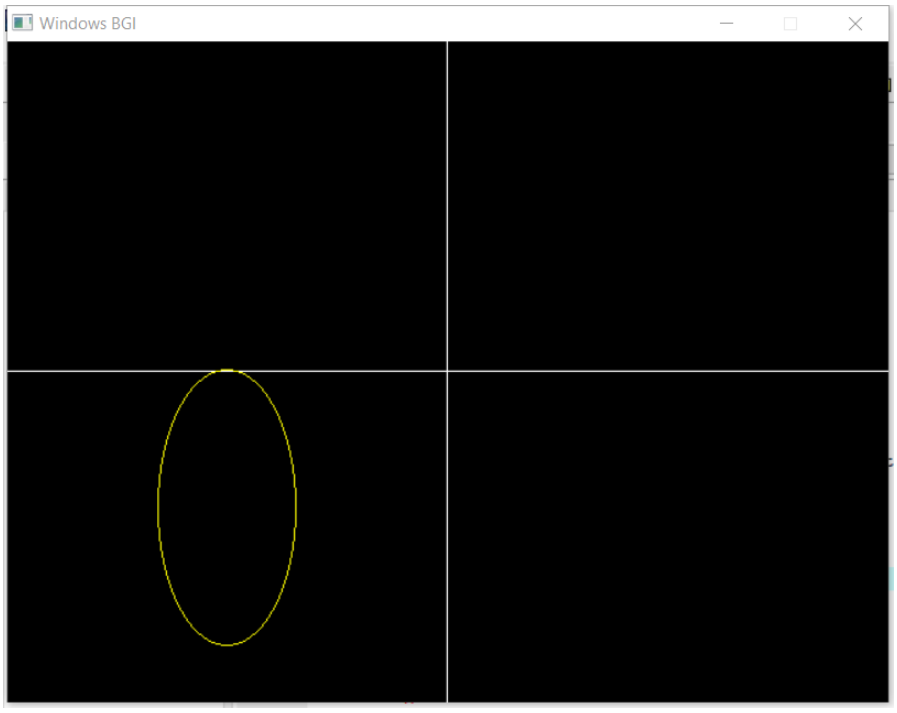
II – QUADRANT

```
Enter Co-ordinates (x,y) :
x : -220
y : 200

 Enter the radius 1= 70

 Enter the  radius 2= 30

DRAWING ELLIPSE USING MID POINT ELLIPSE Algorithm
```



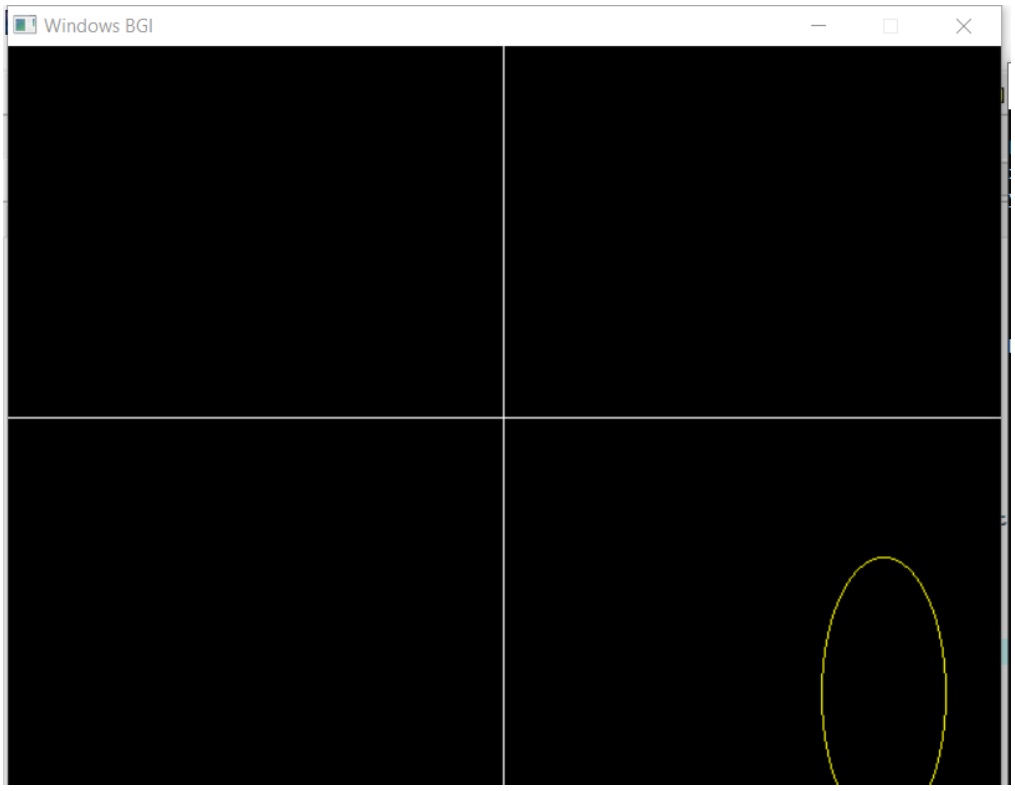
III - QUADRANT

```
Enter Co-ordinates (x,y) :
x : -160
y : -99

 Enter the radius 1= 50

 Enter the  radius 2= 100

DRAWING ELLIPSE USING MID POINT ELLIPSE Algorithm
```



IV – QUADRANT

```
Enter Co-ordinates (x,y) :
x : 245
y : -178

 Enter the radius 1= 40

 Enter the  radius 2= 88

DRAWING ELLIPSE USING MID POINT ELLIPSE Algorithm
```



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

CODE:

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

using namespace std;

#define ROUND(a) ((int)(a+0.5))
#define INSIDE(a) (!a)
#define REJECT(a, b) (a&b)
#define ACCEPT(a, b) ( ! (a|b) )

/* Bit masks encode a point's position relative to the clip edges. A  point's status is encoded by
OR'ing together appropriate bit masks  */
int LEFT_EDGE =1 ;
int RIGHT_EDGE =20 ;
int BOTTOM_EDGE =4 ;
int TOP_EDGE =80;

//defining struct for dcpt
struct dcPt
{
   int x, y;
};

//defining struct for wcpt2
struct wcPt2
{
   int x,y;
};

//fuction for drawing the line
void lineDDA (int xa, int ya, int xb, int yb)
{
   cout<<"DDA line called "<<endl;

   //calculating dx and dy
   int dx = xb - xa, dy = yb - ya, steps, k;
   float xIncrement, yIncrement, x = xa, y = ya;

   // difference(dx,dy) with the greater magnitude determines the value  of parameter steps
   if (abs (dx) > abs (dy))
      steps = abs (dx) ;
```

```
   else
      steps = abs (dy);

   xIncrement = dx/(float) steps;
   yIncrement = dy/(float) steps;
   putpixel (ROUND(x), ROUND(y),YELLOW );

   //looping to generate next pixcel position step times
   for (k=0; k<steps; k++)
           {
              x += xIncrement;
              y += yIncrement;
              putpixel (ROUND(x), ROUND(y),YELLOW );
   }
   cout<<"DAA line returned "<<endl;
}
unsigned char encode (wcPt2 pt, dcPt winMin, dcPt winMax)
{
   unsigned char code=0;
   if (pt.x <winMin.x)
      code = code | LEFT_EDGE;
   if (pt.x > winMax.x)
      code = code | RIGHT_EDGE;
   if (pt .y < winMin. y)
      code = code | BOTTOM_EDGE;
   if (pt .y > winMax. y)
      code = code | TOP_EDGE;
   return (code) ;
}
void swapPts (wcPt2 * p1, wcPt2 * p2)
{
   wcPt2 tmp;
   tmp=*p1 ;
   *p1=*p2;
   *p2=tmp;
}
void swapCodes (unsigned char * c1, unsigned char * c2)
{
   unsigned char tmp;
   tmp = *c1;
   *c1 = *c2;
   *c2 = tmp;
}

//fuction for implementing clip line algo
void clipLine (dcPt winMin, dcPt winMax, wcPt2 p1, wcPt2 p2)
{
```

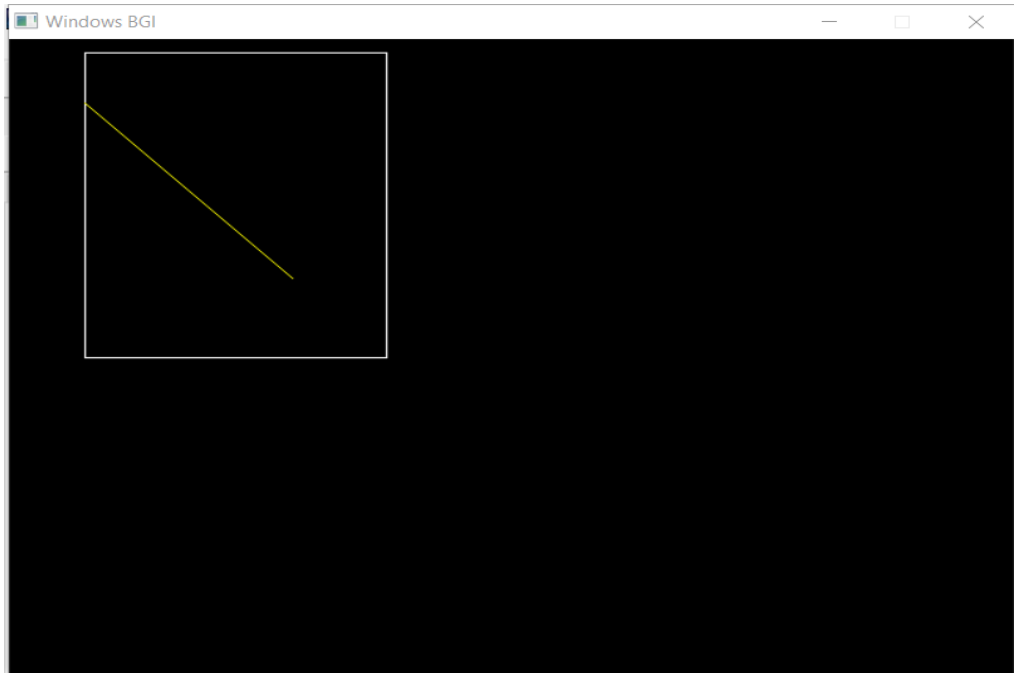
```
unsigned char code1, code2;
int done = FALSE, draw = FALSE;
float m;  //slope of line
while (!done)
        {
   code1 = encode (p1, winMin, winMax);
   code2 = encode (p2, winMin, winMax);
   if (ACCEPT (code1, code2 ) )
                  {
      done = TRUE;
      draw = TRUE;
   }
   else
          {
      if (REJECT (code1, code2))
      {
         done = TRUE;
      }
      else
            {
         if (INSIDE (code1))
         {
            swapPts (&p1, &p2) ;
            swapCodes (&code1, &code2);
         }
         if (p2.x != p1.x)
         {
            m = (p2.y - p1.y) / (p2.x - p1.x); //calculating the slope
         }
         if (code1 & LEFT_EDGE)
                     {
            p1.y += (winMin.x - p1.x)*m;
            p1.x = winMin.x;
         }
         else
               {
            if (code1 & RIGHT_EDGE)
                        {
               p1.y += (winMax.x - p1.x)* m;
               p1.x = winMax.x;
            }
            else
                  {
               if (code1 & BOTTOM_EDGE)
                           {
                  if (p2.x != p1.x)
                  {
```

```
                    p1.x += (winMax.y - p1.y) / m;
                  }
                  p1.y = winMax.y;
                }
                else
                      {
                  if (code1 & TOP_EDGE)
                                {
                    if (p2.x != p1.x)
                    {
                      p1.x += (winMax.y - p1.y) / m;
                    }
                    p1.y = winMax.y;
                  }
                }
              }
            }
          }
        }
  if(draw)
  {
    //calling the lineDDA fuction to draw the line after the cliping
    lineDDA(ROUND(p1.x),ROUND(p1.y), ROUND(p2.x), ROUND(p2.y));
        }
}

//main function
int main()
{
  int gdriver = DETECT,gmode;
  initgraph(&gdriver,&gmode,(char*)"");
  dcPt winMin,winMax;
  wcPt2 p1, p2;

  //values oh winMin(window min ) and winMax(window max) winMin.x = 10;
  winMin.y = 10;
  winMax.x = 240;
  winMax.y = 240;

  //values of p1 and p2
  p1.x = 10;
  p1.y = 10;
  p2.x = 180;
  p2.y = 180;

  cout<<"p1.x :"<<p1.x;
```

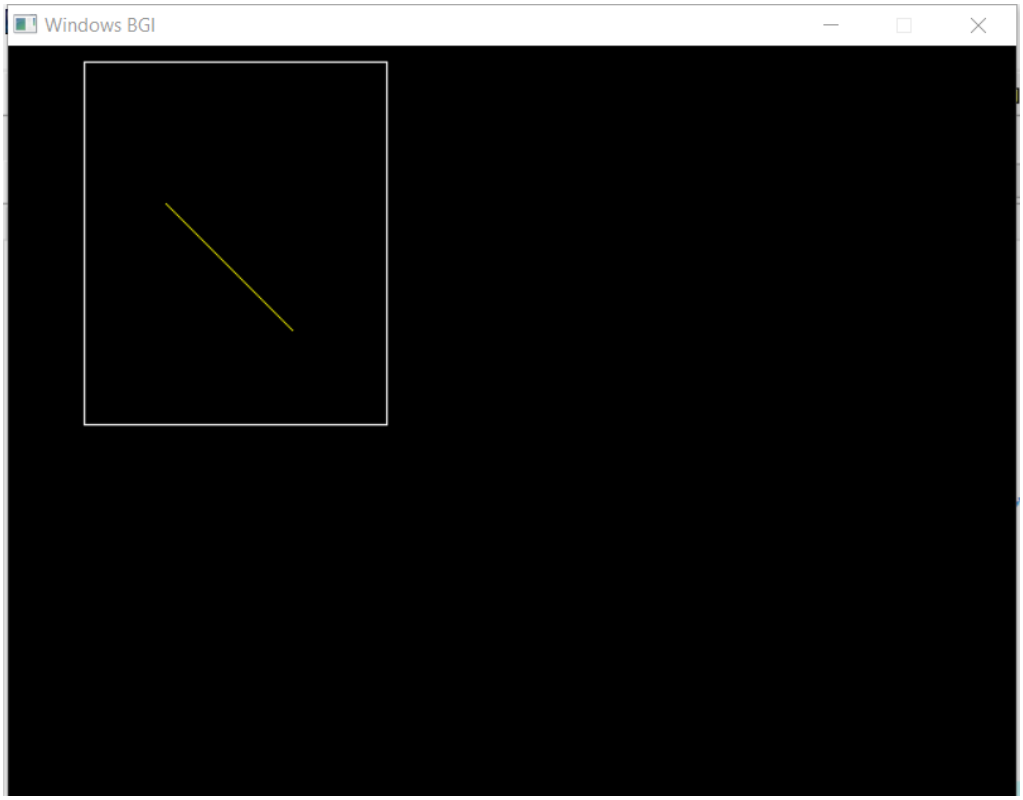
```
  cout<<"\np1.y :"<<p1.y;
  cout<<"\np2.x :"<<p2.x;
  cout<<"\np2.y :"<<p2.y;

  cout<<"\n*******COHEN AND SUTHERLAND LINE CLIPPING *******"<<endl;

  //drawing the rectangle
  rectangle(winMin.x,winMin.y,winMax.x,winMax.y);

  //calling clip line function
  clipLine (winMin, winMax, p1, p2);

  getch();
  return 0;
}
```

OUTPUT :

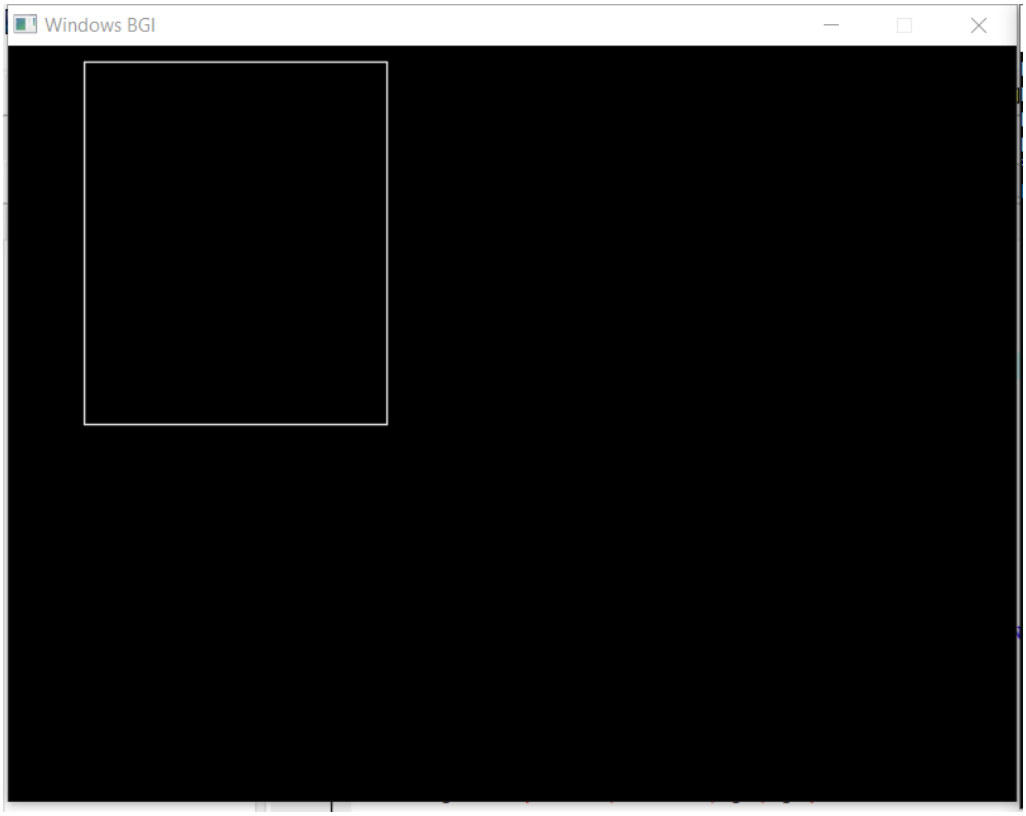
```
p1.x :10
p1.y :10
p2.x :180
p2.y :180
*******COHEN AND SUTHERLAND LINE CLIPPING *******
DDA line called
DAA line returned
```



```
p1.x :100
p1.y :100
p2.x :180
p2.y :180
*******COHEN AND SUTHERLAND LINE CLIPPING *******
DDA line called
DAA line returned
```



```
p1.x :100
p1.y :500
p2.x :180
p2.y :180
*******COHEN AND SUTHERLAND LINE CLIPPING *******
FAILED!
```



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

```
#include<iostream>
#include<conio.h>
#include<graphics.h>
using namespace std;
#define round(a) ((int)(a+0.5))

int k;
float xmin,ymin,xmax,ymax,arr[20],m; // Coordinates for the Rectangular Window

//Clipping Against Left Edge if the Window
void clipleft(float x1,float y1,float x2,float y2)
{
   //Four conditions(in->in, in->out, out->in, out->out)
   if(x2-x1)
      m=(y2-y1)/(x2-x1);
   else
      m=100000;
   if(x1 >= xmin && x2 >= xmin)
   {
      arr[k]=x2;
      arr[k+1]=y2;
      k+=2;
   }
   if(x1 < xmin && x2 >= xmin)
   {
      arr[k]=xmin;
      arr[k+1]=y1+m*(xmin-x1);
      arr[k+2]=x2;
      arr[k+3]=y2;
      k+=4;
   }
   if(x1 >= xmin  && x2 < xmin)
   {
      arr[k]=xmin;
      arr[k+1]=y1+m*(xmin-x1);
      k+=2;
   }
}

//Clipping Against Top Edge of The Window
void cliptop(float x1,float y1,float x2,float y2)
{
   if(y2-y1)
      m=(x2-x1)/(y2-y1);
   else
```

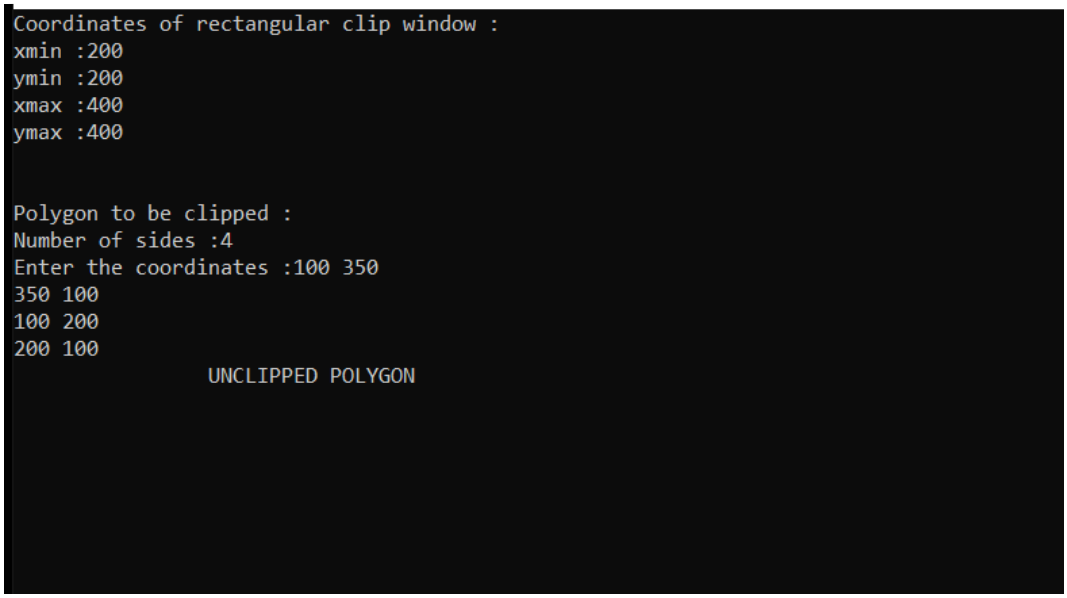
```
    m=100000;
  if(y1 <= ymax && y2 <= ymax)
  {
    arr[k]=x2;
    arr[k+1]=y2;
    k+=2;
  }
  if(y1 > ymax && y2 <= ymax)
  {
    arr[k]=x1+m*(ymax-y1);
    arr[k+1]=ymax;
    arr[k+2]=x2;
    arr[k+3]=y2;
    k+=4;
  }
  if(y1 <= ymax  && y2 > ymax)
  {
    arr[k]=x1+m*(ymax-y1);
    arr[k+1]=ymax;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

//Clipping Against Right Edge of the Window
void clipright(float x1,float y1,float x2,float y2)
{
  if(x2-x1)
    m=(y2-y1)/(x2-x1);
  else
    m=100000;
  if(x1 <= xmax && x2 <= xmax)
  {
    arr[k]=x2;
    arr[k+1]=y2;
    k+=2;
  }
  if(x1 > xmax && x2 <= xmax)
  {
    arr[k]=xmax;
    arr[k+1]=y1+m*(xmax-x1);
    arr[k+2]=x2;
    arr[k+3]=y2;
    k+=4;
  }
  if(x1 <= xmax  && x2 > xmax)
  {
    arr[k]=xmax;
```

```
      arr[k+1]=y1+m*(xmax-x1);
      k+=2;
   }
}

//Clipping Against Bottom Edge of the Window
void clipbottom(float x1,float y1,float x2,float y2)
{
   if(y2-y1)
      m=(x2-x1)/(y2-y1);
   else
      m=100000;
   if(y1 >= ymin && y2 >= ymin)
   {
      arr[k]=x2;
      arr[k+1]=y2;
      k+=2;
   }
   if(y1 < ymin && y2 >= ymin)
   {
      arr[k]=x1+m*(ymin-y1);
      arr[k+1]=ymin;
      arr[k+2]=x2;
      arr[k+3]=y2;
      k+=4;
   }
   if(y1 >= ymin  && y2 < ymin)
   {
      arr[k]=x1+m*(ymin-y1);
      arr[k+1]=ymin;
      k+=2;
   }
}

//Main Function
int main()
{
   int gd=DETECT,gm,n,poly[20];
   initgraph(&gd,&gm,(char*)"");
   float xi,yi,xf,yf,polyy[20];

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

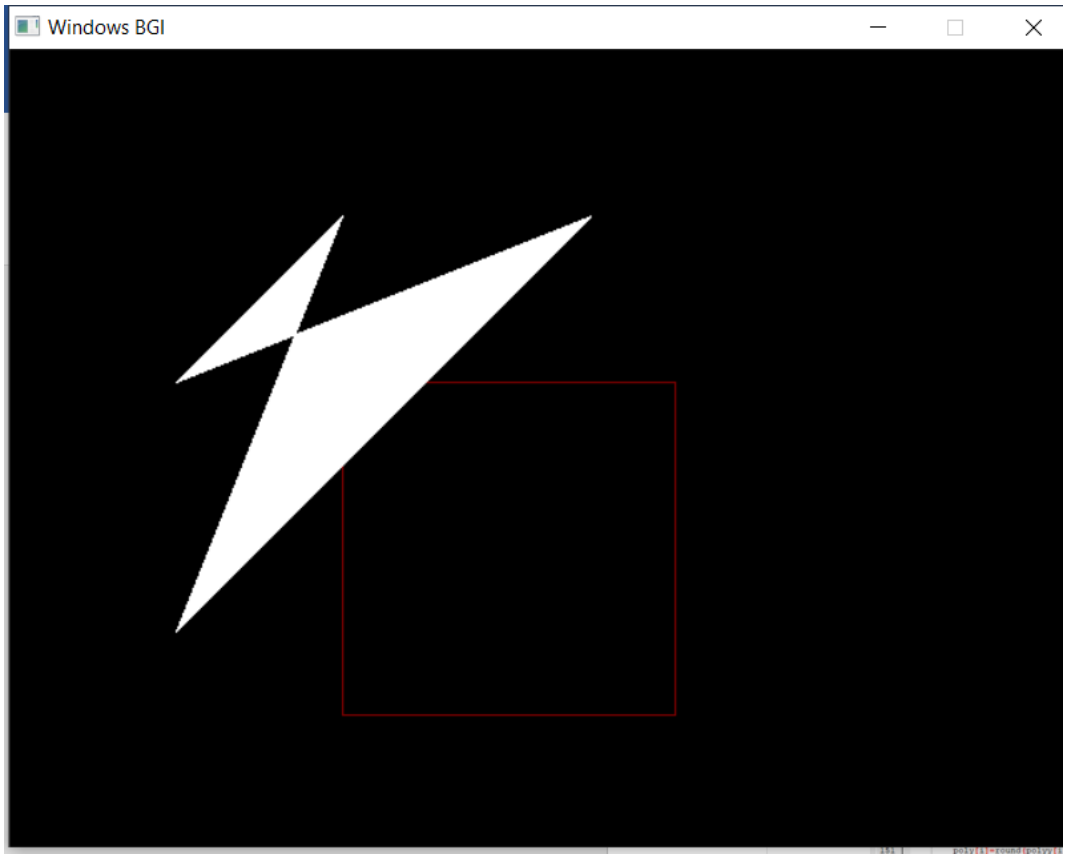
```
cout<<"ymax :";
cin>>ymax;
cout<<"\n\nPolygon to be clipped :\nNumber of sides :";
cin>>n;
cout<<"Enter the coordinates :";
int i;
for(i=0;i < 2*n;i++)
                cin>>polyy[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
for(i=0;i < 2*n+2;i++)
                poly[i]=round(polyy[i]);

//setting color to the box
setcolor(RED);
rectangle(xmin,ymax,ymax,ymin);
cout<<"\t\tUNCLIPPED POLYGON";
setcolor(WHITE);
fillpoly(n,poly);
        getch();
cleardevice();
k=0;
for(i=0;i < 2*n;i+=2)
                clipleft(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
                polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
                cliptop(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
                polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
                clipright(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
                polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
```

```
                clipbottom(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
  for(i=0;i < k;i++)
                poly[i]=round(arr[i]);
  if(k)
                fillpoly(k/2,poly);
  setcolor(RED);
  rectangle(xmin,ymax,xmax,ymin);
  cout<<"\tCLIPPED POLYGON";

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

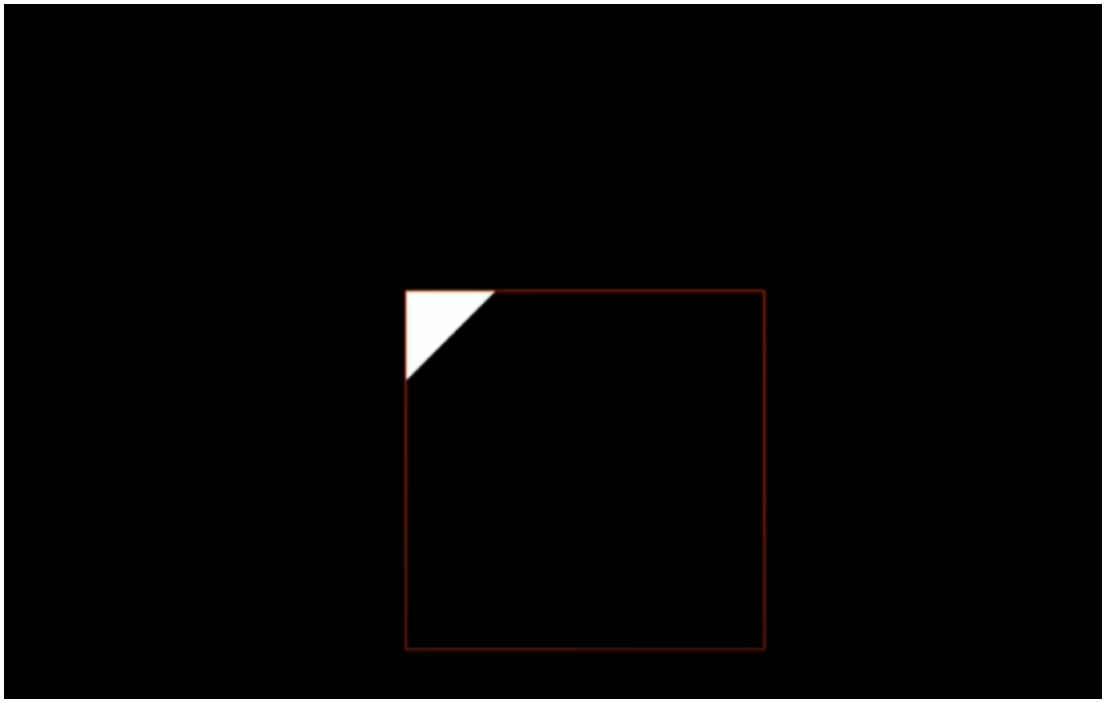
OUTPUT:



POLYGON BEFORE CLIPPING :



POLYGON AFTER CLIPPING :



5. Write a program to fill a polygon using Scan line fill algorithm.

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

const int WINDOW_HEIGHT = 1000;

typedef struct tdcPt
{
	int x;
	int y;
}dcPt;

typedef struct tEdge
{
	int yUpper;
	float xIntersect, dxPerScan;
	struct tEdge *next;
}Edge;

// Vertices: Array of structures.
dcPt vertex[5] = {{220, 340}, {210, 190}, {310, 240}, {370, 250}, {360, 320}};

// Inserts edge into list in order of increasing xIntersect field.
void insertEdge(Edge *list, Edge *edge)
{
	Edge *p, *q = list;
	p = q->next;

	while (p != NULL)
	{
		if (edge->xIntersect < p->xIntersect)
			p = NULL;
		else
		{
			q = p;
			p = p->next;
		}
	}
	edge->next = q->next;
	q->next = edge;
}

// For an index, return y-coordinate of next nonhorizontal line
int yNext(int k, int cnt, dcPt *pts)
```

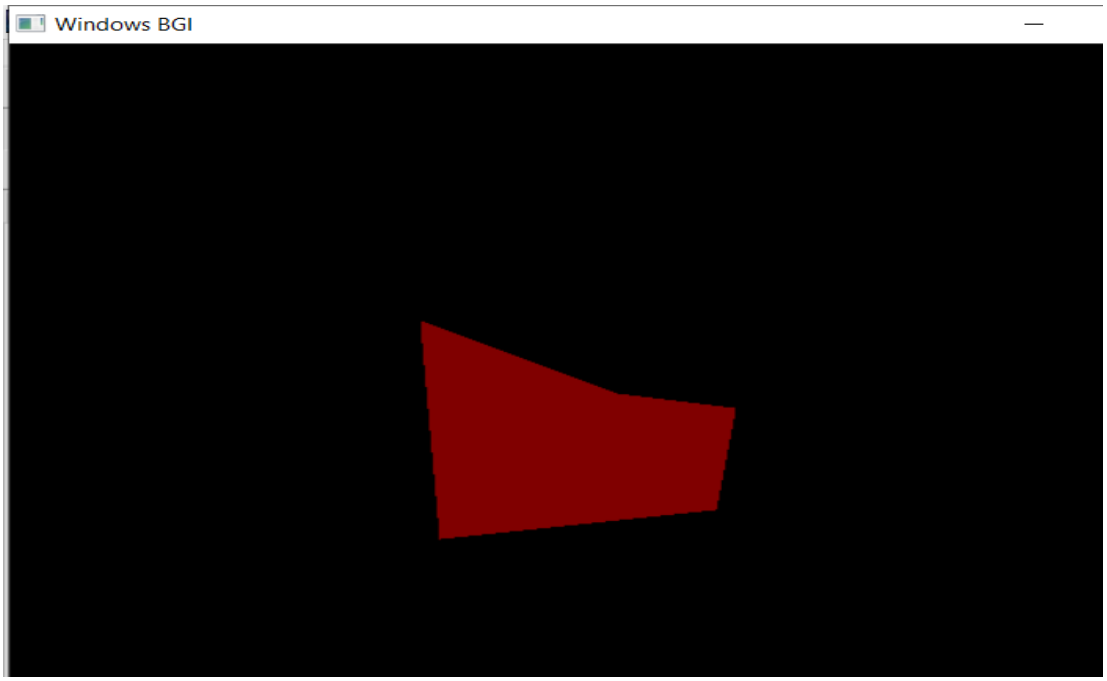
```
{
        int j;

        if ((k + 1) > (cnt - 1))
                j = 0;
        else
                j = k + 1;

        while(pts[k].y == pts[j].y)
        {
                if ((j + 1) > (cnt - 1))
                        j = 0;
                else
                        j++;
        }
        return (pts[j].y);
}

/* Store lower-y coordinate and inverse slope for each edge. Adjust
and store upper-y coordinate for edges that are the lower member
of a monotically increasing or decreasing pair of edges */
void makeEdgeRec(dcPt lower, dcPt upper, int yComp, Edge *edge, Edge *edges[])
{
        edge->dxPerScan = (float) (upper.x - lower.x) / (upper.y - lower.y);
        edge->xIntersect = lower.x;
        if (upper.y < yComp)
                edge->yUpper = upper.y - 1;
        else
                edge->yUpper = upper.y;
        insertEdge(edges[lower.y], edge);
}

void buildEdgeList(int cnt, dcPt *pts, Edge *edges[])
{
        Edge *edge;
        dcPt v1, v2;
        int i, yPrev = pts[cnt - 2].y;

        v1.x = pts[cnt - 1].x; v1.y = pts[cnt - 1].y;
        for(int i = 0; i < cnt; i++)
        {
                v2 = pts[i];
                if (v1.y != v2.y)                                       // non-horizontal line
                {
                        edge = (Edge *) malloc (sizeof(Edge));
                        if (v1.y < v2.y)                                        // up-going edge
                                makeEdgeRec(v1, v2, yNext(i, cnt, pts), edge, edges);
```

```
                        else                                    // down-going edge
                                makeEdgeRec(v2, v1 , yPrev, edge, edges);
                }
                yPrev = v1.y;
                v1 = v2;
        }
}

void buildActiveList(int scan, Edge *active, Edge *edges[])
{
        Edge *p, *q;

        p = edges[scan]->next;
        while (p)
        {
                q = p->next;
                insertEdge(active, p);
                p = q;
        }
}

void fillScan(int scan, Edge *active)
{
        Edge *p1, *p2 ;
        int i;

        p1 = active->next;
        while (p1)
        {
                p2 = p1->next;
                for(i = p1->xIntersect; i < p2->xIntersect; i++)
                        putpixel((int) i, scan, RED);
                p1 = p2->next;
        }
}

void deleteAfter(Edge *q)
{
        Edge *p = q->next;
        q->next = p->next;
        free(p);
}

/* Delete completed edges. Update 'xIntersect' field for others */
void updateActiveList(int scan, Edge *active)
{
        Edge *q = active, *p = active->next;
```

```
	while (p)
	{
		if (scan >= p->yUpper)
		{
			p = p->next;
			deleteAfter(q);
		}
		else
		{
			p->xIntersect = p->xIntersect + p->dxPerScan;
			q = p;
			p = p->next;
		}
	}
}

void resortActiveList(Edge *active)
{
	Edge *q, *p = active->next;
	active->next = NULL;
	while (p)
	{
		q = p->next;
		insertEdge(active, p);
		p = q;
	}
}

void scanFill(int cnt, dcPt *pts)
{
	Edge *edges[WINDOW_HEIGHT], *active;
	int i, scan;

	for (i = 0; i < WINDOW_HEIGHT; i++)
	{
		edges[i] = (Edge *) malloc (sizeof(Edge));;
		edges[i]->next = NULL;
	}

	buildEdgeList(cnt, pts, edges);
	active = (Edge *) malloc (sizeof(Edge));;
	active->next = NULL;

	for (scan = 0; scan < WINDOW_HEIGHT; scan++)
	{
		buildActiveList(scan, active, edges);
```

```
                if (active->next)
                {
                        fillScan(scan, active);
                        updateActiveList(scan, active) ;
                        resortActiveList(active);
                }
        }
        free(edges[WINDOW_HEIGHT]);
        free(active);
}

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

        float X = getmaxx(), Y = getmaxy();
        float x_mid = X / 2;
        float y_mid = Y / 2;

        cleardevice();
        scanFill(5, vertex);

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

OUTPUT:



6. Write a program to apply various 2-D Transformations on a 2-D object (TRIANGLE).

CODE:

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

int mat[3][3];

//DDA line function
void dda_line(int x1 , int y1 , int x2 , int y2 , int col)
{
   int dx , dy , st;
   dx = x2 - x1;
   dy = y2 - y1;
   float y , x , xinc , yinc;
   int xmid , ymid;
   xmid = getmaxx()/2;
   ymid = getmaxy()/2;
   if(abs(dx) > abs(dy))
          {
      st = abs(dx);
   }
   else
          {
      st = abs(dy);
   }

   xinc = dx / st;
   yinc = dy / st;
   x = x1;
   y = y1;
   for(int i=0 ; i<st ; i++)
          {
      x += xinc;
      y += yinc;
      putpixel(ceil(x) + xmid , ymid - ceil(y),col);
   }
}
```

```
//Rotation
void rotate()
{
   int xmid , ymid;
   xmid = getmaxx()/2;
   ymid = getmaxy()/2;
   line(xmid , 0 , xmid , getmaxy());
   line(0 , ymid , getmaxx() , ymid);
   int c[3][2] ,l , m, i , j , k;
   int a[3][2]={{200,200},{200,100},{100,200}};
   int t[2][2]={{0,1},{-1,0}};
   for( i = 0 ; i < 3 ; i++)
           {
      for(j=0 ; j<2 ; j++)
                     {
         c[i][j]=0;
      }
   }
   //Original Triangle
   dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
   dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
   dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
   for ( i=0;i<3;i++)
           {
      for ( j=0;j<2;j++)
                     {
         for ( k=0;k<2;k++)
                              {
            c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
         }
      }
   }
   //Transformed Triangle
   dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
   dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
   dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

//Reflection
void reflection()
{
   int xmid , ymid;
   xmid = getmaxx()/2;
   ymid = getmaxy()/2;
   line(xmid , 0 , xmid , getmaxy());
   line(0 , ymid , getmaxx() , ymid);
   int c[3][2] ,l , m, i , j , k;
```

```
   int a[3][2]={{200,200},{200,100},{100,200}};
   int t[2][2]={{0,-1},{-1,0}};
   for( i = 0 ; i < 3 ; i++)
           {
      for(j=0 ; j<2 ; j++)
                     {
         c[i][j]=0;
      }
   }
   //Original Triangle
   dda_line (a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
   dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
   dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
   for ( i=0;i<3;i++)
           {
      for ( j=0;j<2;j++)
                     {
         for ( k=0;k<2;k++)
                                 {
            c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
         }
      }
   }
   //Transformed Triangle
   dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
   dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
   dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

//Scaling
void scaling()
{
   int xmid , ymid;
   xmid = getmaxx()/2;
   ymid = getmaxy()/2;
   line(xmid , 0 , xmid , getmaxy());
   line(0 , ymid , getmaxx() , ymid);

   int c[3][2] ,l , m, i , j , k;
   int a[3][2]={{20,20},{20,10},{10,20}};
   int t[2][2]={{5,0},{0,5}};
   for( i = 0 ; i < 3 ; i++)
           {
      for(j=0 ; j<2 ; j++)
              {
         c[i][j]=0;
      }
```

```
    }
    //Original Triangle
    dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
    dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
    dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
    for ( i=0;i<3;i++)
            {
       for ( j=0;j<2;j++)
                    {
          for ( k=0;k<2;k++)
                                {
             c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
          }
       }
    }
    //Transformed Triangle
    dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
    dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
    dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

void multi(int a[3][3] , int b[3][3] )
{
    int i , j ,k;
    int c[3][3];
    for( i = 0 ; i < 3 ; i++)
            {
       for(j=0 ; j< 3 ; j++)
                    {
          c[i][j]=0;
       }
    }
    for ( i=0;i<3;i++)
            {
       for ( j=0;j<3;j++)
                    {
          for ( k=0;k<3;k++)
                                {
             c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
          }
       }
    }
    for( i = 0 ; i < 3 ; i++)
            {
       for(j=0 ; j< 3 ; j++)
                    {
          mat[i][j]=c[i][j];
```

```
    }
  }
}

//Reflection About an arbitrary line
void reflection_arbitrary()
{
  int xmid , ymid;
  xmid = getmaxx()/2;
  ymid = getmaxy()/2;
  line(xmid , 0 , xmid , getmaxy());

  line(0 , ymid , getmaxx() , ymid);
  int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};
  int t[3][3]={{1,0,0},{0,1,0},{0,0,1}};
  int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
  int ref[3][3]={{1,0,0},{0,-1,0},{0,0,1}};
  int rinv[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
  int tinv[3][3]={{1,0,0},{0,1,0},{0,1,1}};
  //Original Triangle
  dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
  dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
  dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
  multi(t,r);
  multi(mat,ref);
  multi(mat,rinv);
  multi(mat,tinv);
  multi(a,mat);
  //Transformed Triangle
  dda_line(mat[0][0],mat[0][1],mat[1][0],mat[1][1],GREEN);
  dda_line(mat[1][0],mat[1][1],mat[2][0],mat[2][1],GREEN);
  dda_line(mat[2][0],mat[2][1],mat[0][0],mat[0][1],GREEN);
}

//Rotation About an arbitrary point
void rotation_arbitrary()
{
  int xmid , ymid;
  xmid = getmaxx()/2;
  ymid = getmaxy()/2;
  line(xmid , 0 , xmid , getmaxy());
  line(0 , ymid , getmaxx() , ymid);
  int c[3][3] , i , j , k;
  int l[1][3]={{200,200,1}};
  int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};
  int t[3][3]={{1,0,0},{0,1,0},{-133,-133,1}};
  int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
```

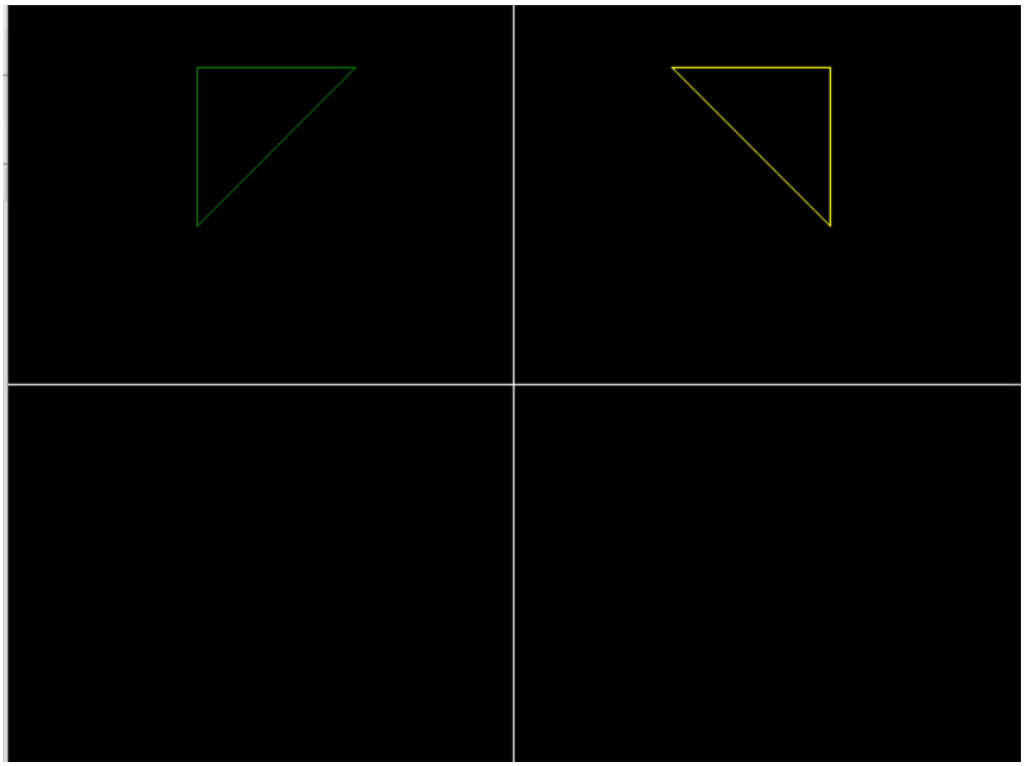
```
   int tinv[3][3]={{1,0,0},{0,1,0},{133,133,1}};
   //Original Triangle
   dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
   dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
   dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
   multi(t,r);
   multi(mat,tinv);
   for( i = 0 ; i < 3 ; i++)
          {
      for(j=0 ; j<3 ; j++)
                    {
         c[i][j]=0;
      }
   }
   for ( i=0;i<3;i++)
          {
      for ( j=0;j<3;j++)
                    {
         for ( k=0;k<3;k++)
                              {
            c[i][j]=c[i][j]+(a[i][k]*mat[k][j]);
         }
      }
   }
   //Transformed Triangle
   dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
   dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
   dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

//main function
int main()
{
   int gdriver = DETECT , gmode , errorcode;
   initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");
   int n , m;
   cout<<" 1.Rotation \n 2.Reflection \n 3.Scaling \n 4.Reflection about an arbitrary axis \n";
   cout<<" 5.Rotation about an arbitrary point\n";
   cout<<" Enter your choice : ";
   cin>>n;
   switch(n)
          {
      case 1 : rotate();
      break;
      case 2 : reflection();
      break;
      case 3 : scaling();
```

```
      break;
      case 4 : reflection_arbitrary();
      break;
      case 5 : rotation_arbitrary();
      break;
      default : cout<<"Invalid Choice\n";
   }
   getch();
}
```

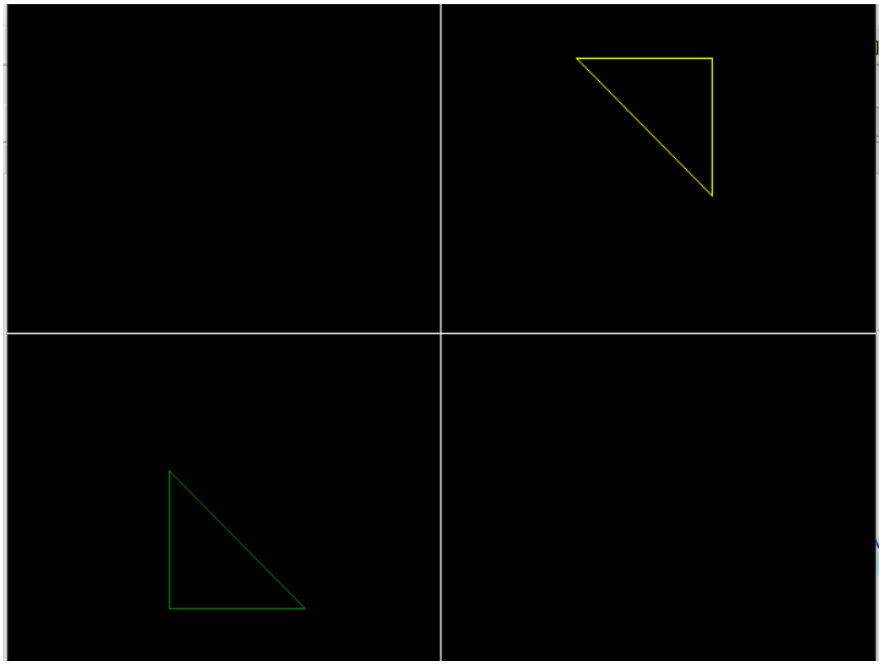
OUTPUT:

1. Rotation

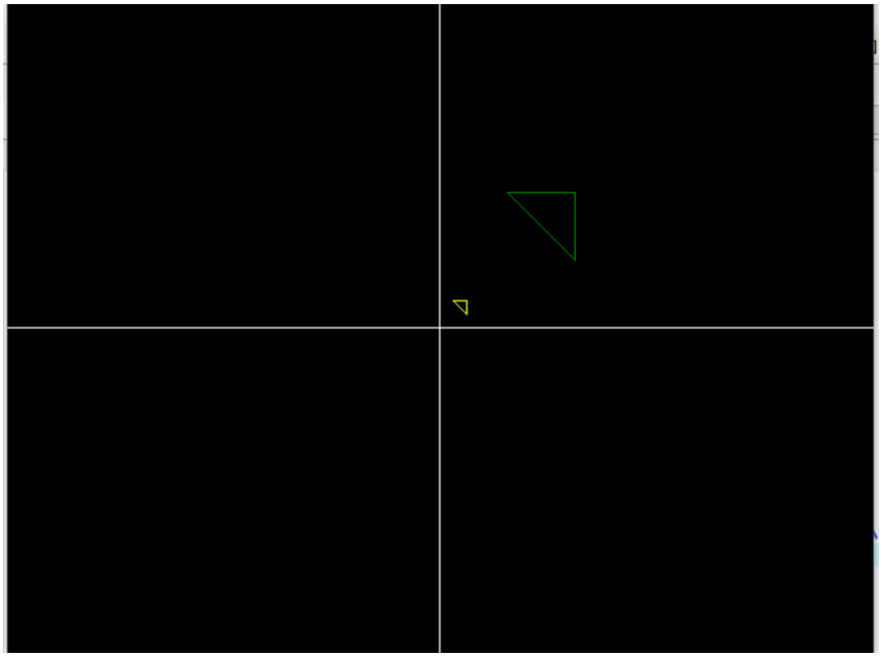
```
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 1
```



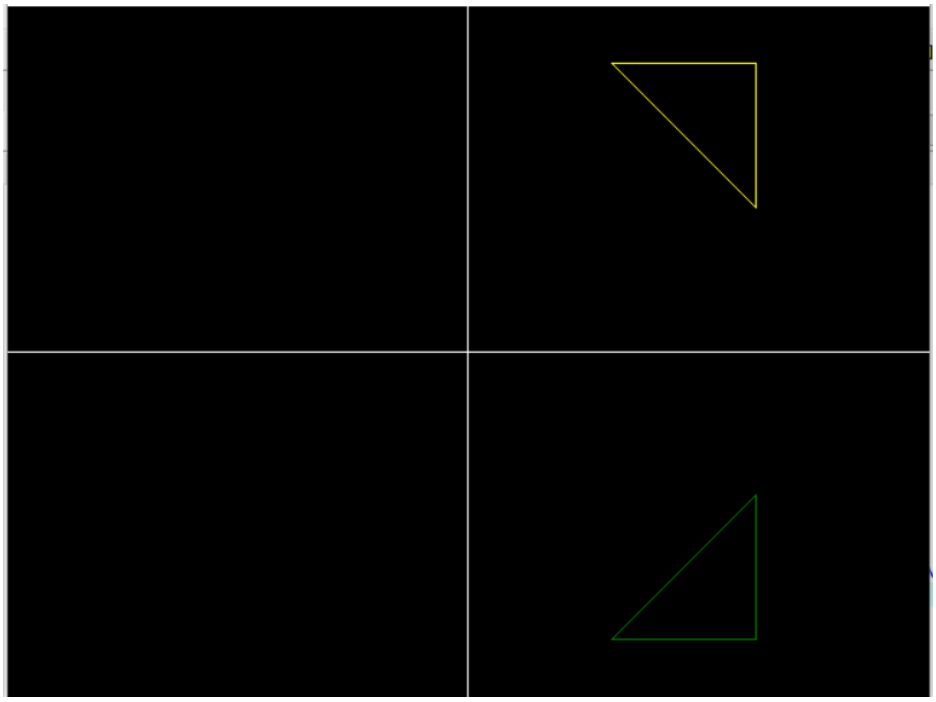
2.Reflection



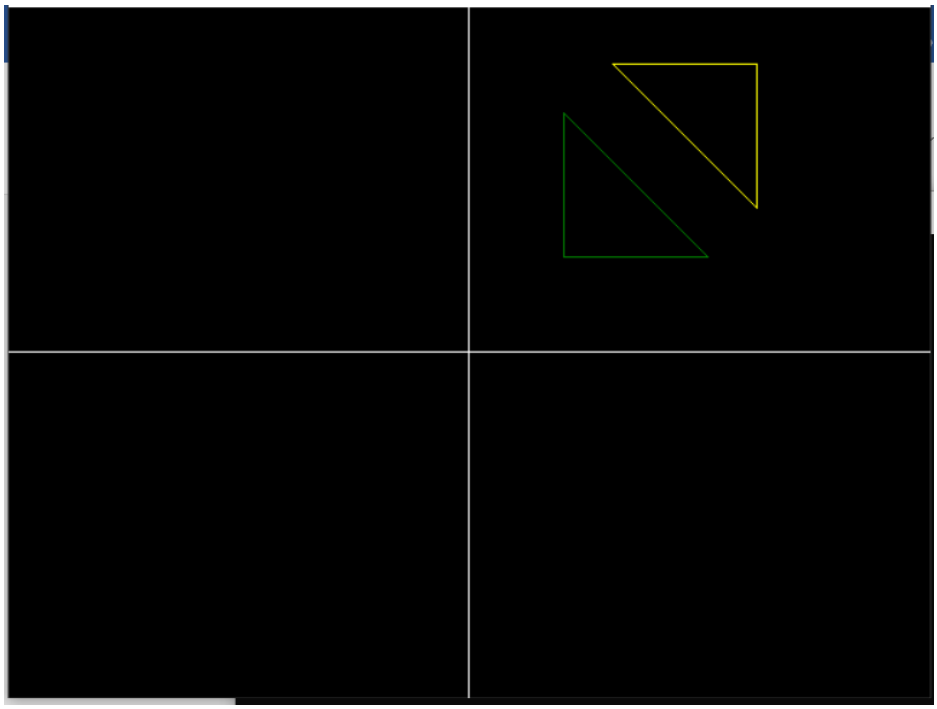
3.Scaling



4.Reflection about an arbitrary axis



5.Rotation about an arbitrary point



7. Write a program to apply various 3D Transformations on a 3D object and then apply parallel and perspective projection on it.

CODE:

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

using namespace std;

  int gd=DETECT,gm;
  double x,x2,y,y2;

 //Creating draw cube function for drawing cube
  void draw_cube(double edge[20][3])
  {
     initgraph(&gd,&gm,(char*)"");
     int i;
     clearviewport();
     for(i=0;i < 19;i++)
     {
        x=edge[i][0]+edge[i][2]*(cos(2.3562));
        y=edge[i][1]-edge[i][2]*(sin(2.3562));
        x2=edge[i+1][0]+edge[i+1][2]*(cos(2.3562));
        y2=edge[i+1][1]-edge[i+1][2]*(sin(2.3562));
        line(x+320,240-y,x2+320,240-y2);
     }
     line(320,240,320,25);
     line(320,240,550,240);
     line(320,240,150,410);
     getch();
     closegraph();
  }
 //Scaling Function
  void scale(double edge[20][3])
  {
     double a,b,c;
     int i;
     cout<<"Enter The Scaling Factors "<<endl;
     cin>>a>>b>>c;
     initgraph(&gd,&gm,"..\bgi");
     clearviewport();
```

```
    for(i=0;i < 20;i++)
    {  // Scaling Factors a, b, c at X, Y, Z
       edge[i][0]=edge[i][0]*a;
       edge[i][1]=edge[i][1]*b;
       edge[i][2]=edge[i][2]*c;
    }
    draw_cube(edge);
    closegraph();
}
 // Creating Translation function
 void translate(double edge[20][3])
 {
    int a,b,c;
    int i;
    cout<<"Enter The Translation Factors"<<endl;
    cin>>a>>b>>c;
    initgraph(&gd,&gm,"..\bgi");
    clearviewport();
    for(i=0;i < 20;i++)
    {
       //Three Translation Factors a, b, c
       edge[i][0]+=a;
       edge[i][0]+=b;
       edge[i][0]+=c;
    }
    draw_cube(edge);
    closegraph();
 }

// Creating Rotation About an Axes function
 void rotate(double edge[20][3])
 {
    int ch;
    int i;
    double temp,theta,temp1;

    cout<<"Rotation About"<<endl;
    cout<<"1 X-Axis "<<endl;
    cout<<"2 Y-Axis"<<endl;
    cout<<"3 Z-Axis "<<endl;
    cout<<"Enter Your Choice "<<endl;
    cin>>ch;
    switch(ch)
    {  //For X-axis
       case 1:
                cout<<" Enter The Angle ";
                cin>>theta;
```

```
                theta=(theta*3.14)/180;
                for(i=0;i < 20;i++)
                {
                   edge[i][0]=edge[i][0];
                   temp=edge[i][1];
                   temp1=edge[i][2];
                   //Transformation Matrix For X-axis
                   edge[i][1]=temp*cos(theta)-temp1*sin(theta);
                   edge[i][2]=temp*sin(theta)+temp1*cos(theta);
                }
                draw_cube(edge);
                break;
        //For Y-axis
        case 2:
                cout<<" Enter The Angle ";
                cin>>theta;
                theta=(theta*3.14)/180;
                for(i=0;i < 20;i++)
                {
               edge[i][1]=edge[i][1];
              temp=edge[i][0];
              temp1=edge[i][2];
        //Transformation Matrix For Y-axis
         edge[i][0]=temp*cos(theta)+temp1*sin(theta);
         edge[i][2]=-temp*sin(theta)+temp1*cos(theta);
                }
                draw_cube(edge);
                break;
        //For Z-axis
        case 3:
         cout<<" Enter The Angle ";
         cin>>theta;
         theta=(theta*3.14)/180;
           for(i=0;i < 20;i++)
           {
            edge[i][2]=edge[i][2];
              temp=edge[i][0];
              temp1=edge[i][1];
         //Transformation Matrix For Z-axis
          edge[i][0]=temp*cos(theta)-temp1*sin(theta);
         edge[i][1]=temp*sin(theta)+temp1*cos(theta);

}
                draw_cube(edge);
                break;
     }
  }
```

```
// Creating Reflection About an Axes function
void reflect(double edge[20][3])
{
   int ch;
   int i;

   cout<<"Reflection About "<<endl;
   cout<<"1 X-Axis"<<endl;
   cout<<"2 Y-Axis "<<endl;
   cout<<"3 Z-Axis "<<endl;
   cout<<"Enter Your Choice "<<endl;
   cin>>ch;
   switch(ch)
   {  //For X-axis
      case 1:
            for(i=0;i < 20;i++)
            {
               edge[i][0]=edge[i][0];
               edge[i][1]=-edge[i][1];
               edge[i][2]=-edge[i][2];
            }
            draw_cube(edge);
            break;
      //For Y-axis
      case 2:
            for(i=0;i < 20;i++)
            {
               edge[i][1]=edge[i][1];
               edge[i][0]=-edge[i][0];
               edge[i][2]=-edge[i][2];
            }
            draw_cube(edge);
            break;
      //For Z-axis
      case 3:
            for(i=0;i < 20;i++)
            {
               edge[i][2]=edge[i][2];
               edge[i][0]=-edge[i][0];
               edge[i][1]=-edge[i][1];
            }
            draw_cube(edge);
            break;
   }
}
// Creating Perspective Projection About an Axes function
void perspect(double edge[20][3])
```

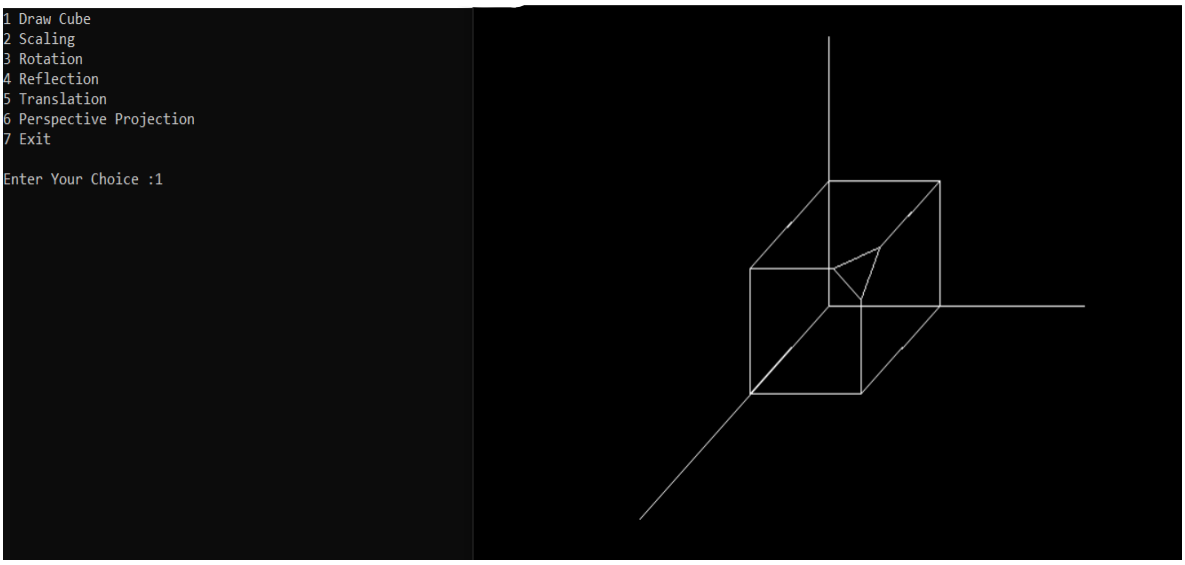
```
{
   int ch;
   int i;
   double p,q,r;
   cout<<"Perspective Projection About"<<endl;
   cout<<"1 X-Axis "<<endl;
   cout<<"2 Y-Axis "<<endl;
   cout<<"3 Z-Axis"<<endl;
   cout<<"Enter Your Choice :"<<endl;
   cin>>ch;
   switch(ch)
   {
      //For X-axis
      case 1:
       cout<<" Enter P :";
       cin>>p;
      for(i=0;i < 20;i++)
      {
         edge[i][0]=edge[i][0]/(p*edge[i][0]+1);
         edge[i][1]=edge[i][1]/(p*edge[i][0]+1);
         edge[i][2]=edge[i][2]/(p*edge[i][0]+1);
      }
                draw_cube(edge);
                break;
      //For Y-axis
      case 2: cout<<" Enter Q :";
        cin>>q;
      for(i=0;i < 20;i++)
       {
       edge[i][1]=edge[i][1]/(edge[i][1]*q+1);
       edge[i][0]=edge[i][0]/(edge[i][1]*q+1);
       edge[i][2]=edge[i][2]/(edge[i][1]*q+1);
 }
                draw_cube(edge);
                break;
 //For Z-axis
 case 3:
        cout<<" Enter R :";
        cin>>r;
        for(i=0;i < 20;i++)
        {
         edge[i][2]=edge[i][2]/(edge[i][2]*r+1);
         edge[i][0]=edge[i][0]/(edge[i][2]*r+1);
         edge[i][1]=edge[i][1]/(edge[i][2]*r+1);
        }
       draw_cube(edge);
       break;
```

```
    }
    closegraph();
 }
//Main Function
 int main()
 {
    int choice;
    double edge[20][3]={
                        100,0,0,
                        100,100,0,
                        0,100,0,
                        0,100,100,
                        0,0,100,
                        0,0,0,
                        100,0,0,
                        100,0,100,
                        100,75,100,
                        75,100,100,
                        100,100,75,
                        100,100,0,
                        100,100,75,
                        100,75,100,
                        75,100,100,
                        0,100,100,
                        0,100,0,
                        0,0,0,
                        0,0,100,
                        100,0,100
                      };
    while(1)
    {

       cout<<"1 Draw Cube "<<endl;
       cout<<"2 Scaling "<<endl;
       cout<<"3 Rotation "<<endl;
       cout<<"4 Reflection "<<endl;
       cout<<"5 Translation "<<endl;
       cout<<"6 Perspective Projection "<<endl;
       cout<<"7 Exit "<<endl;
       cout<<"\nEnter Your Choice :";
       cin>>choice;
       switch(choice)
       {
          case 1:
                draw_cube(edge);
                break;
```

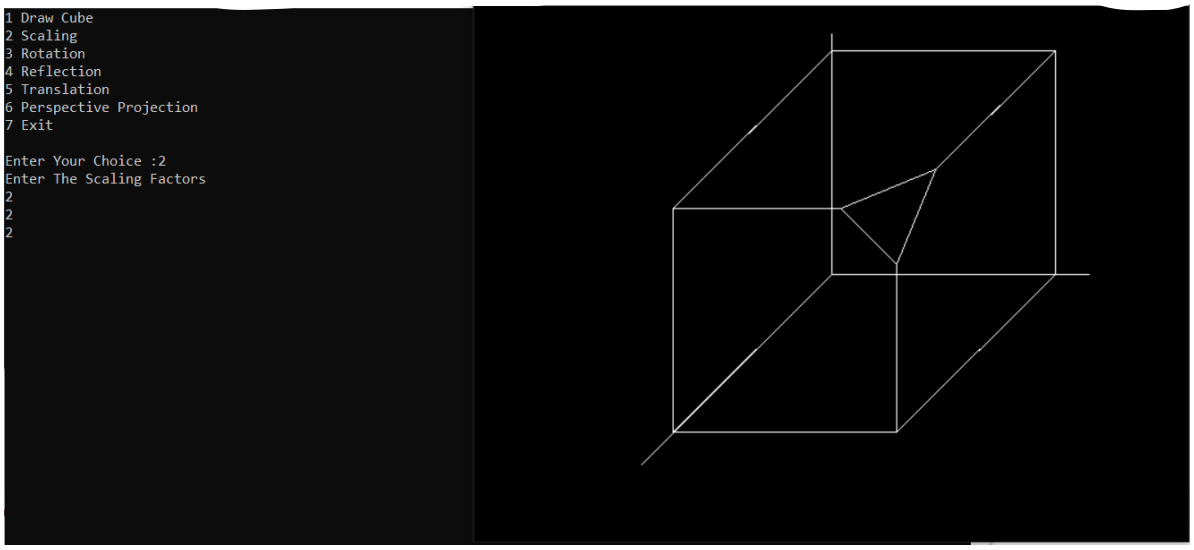
```
        case 2:
              scale(edge);
              break;

        case 3:
              rotate(edge);
              break;

        case 4:
              reflect(edge);
              break;

        case 5:
              translate(edge);
              break;

        case 6:
              perspect(edge);
              break;

        case 7:
              exit(0);

        default:
              cout<<" Press A Valid Key...!!! ";
              getch();
              break;
      }
   closegraph();
   }
   return 0;
}
```

OUTPUT:

ORIGINAL CUBE:

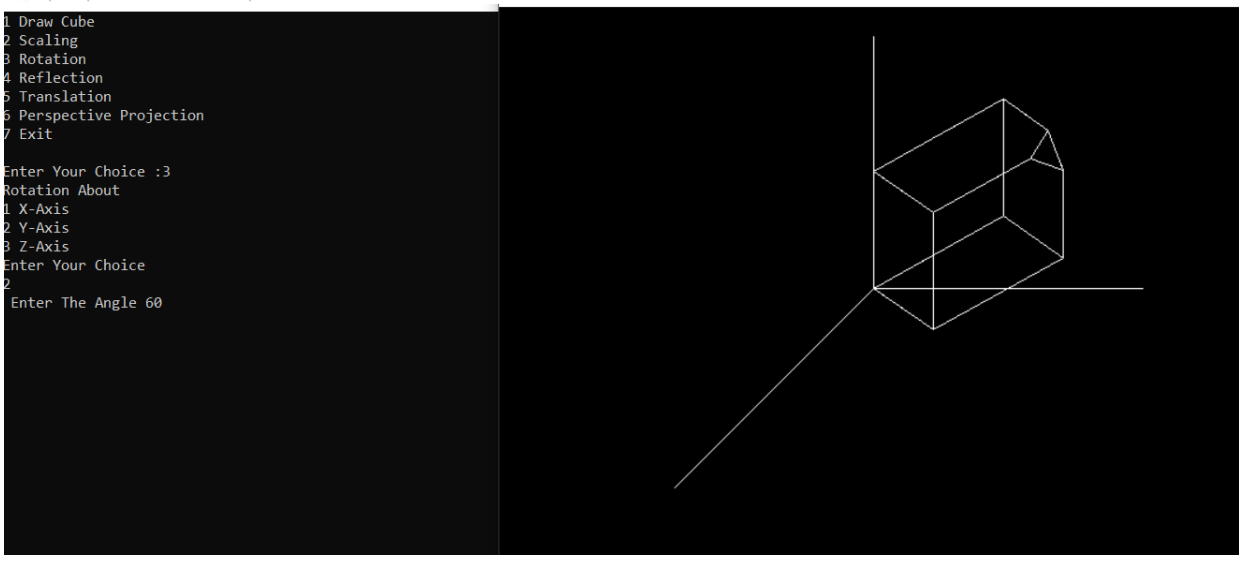


SCALING OF CUBE WITH FACTOR 2 UNIT :

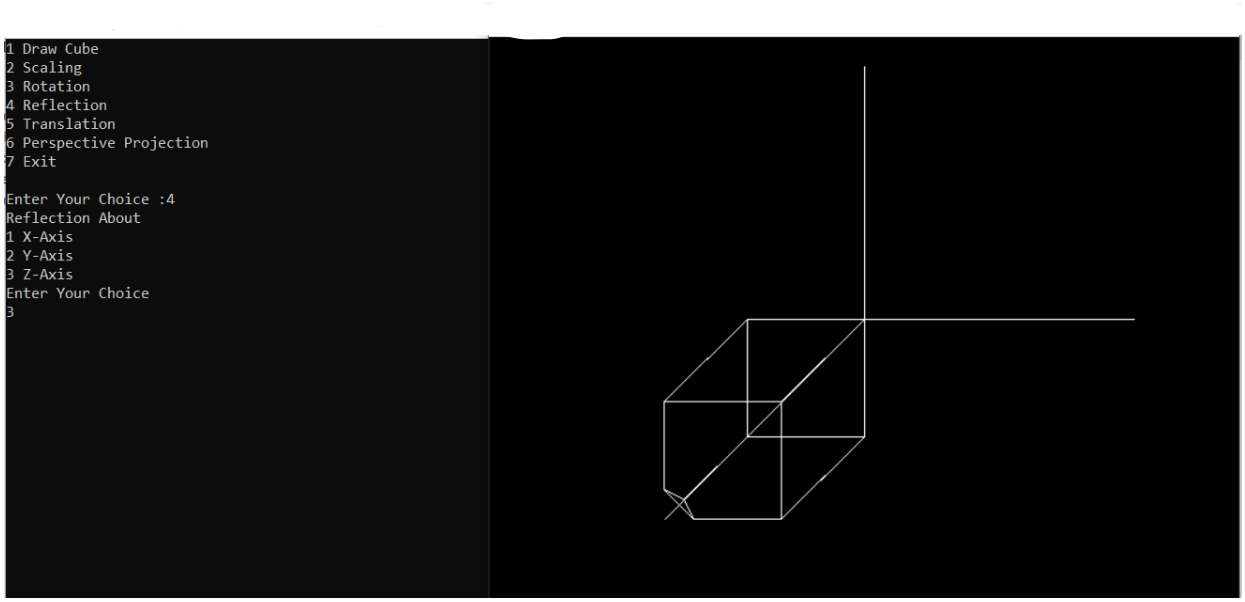


ROTATION OF A CUBE ABOUT X-AXIS WITH ANGLE 30 DEGREE

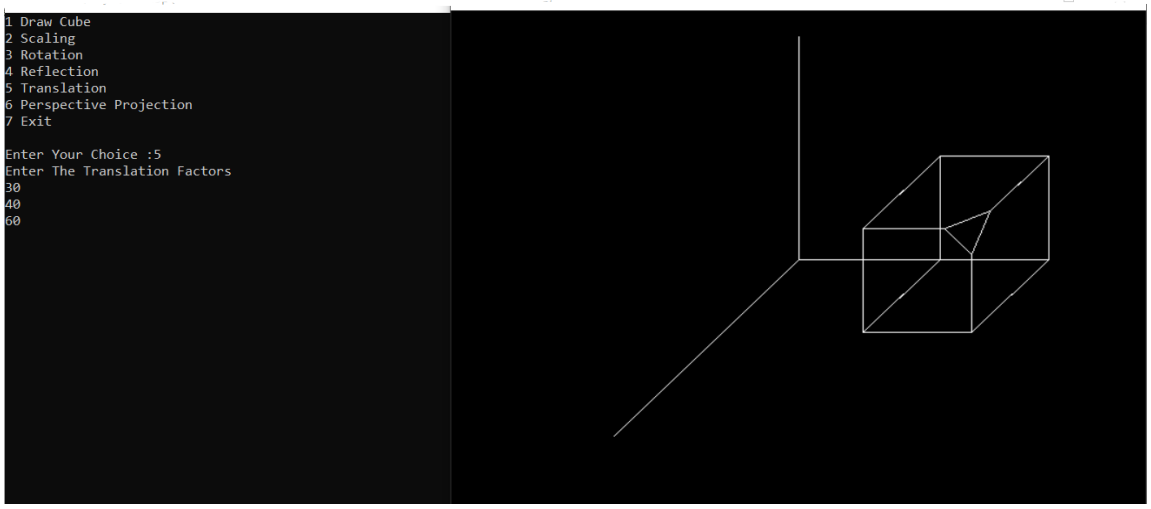
ROTATION OF A CUBE ABOUT Y-AXIS WITH ANGLE 60 DEGREE



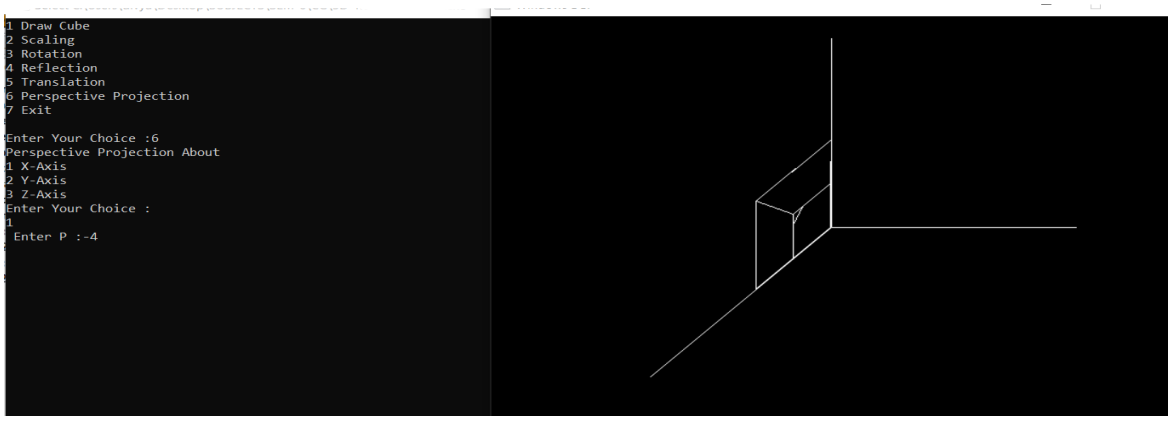
REFLECTION OF A CUBE ABOUT Z-AXIS



TRANSLATION OF A CUBE WITH FACTORS (30, 40, 60)



PERSPECTIVE PROJECTION ABOUT X-AXIS WITH P=-4



8. Write a program to draw Hermite and Bezier curve.

CODE:

```
#include<iostream>
#include<graphics.h>
#include<math.h>

using namespace std;

//creating Bezier curve function
void bezier_curve(int x[4], int y[4])
{
  double t;

  for(t=0.0;t<1.0;t=t+0.0005)
   {
      //Curve Equation of x and y coordinates by using blending function
      double xt=pow(1-t,3)*x[0]+3*t*pow(1- t,2)*x[1]+3*pow(t,2)*(1-t)*x[2]+pow(t,3)*x[3];
      double yt=pow(1-t,3)*y[0]+3*t*pow(1- t,2)*y[1]+3*pow(t,2)*(1-t)*y[2]+pow(t,3)*y[3];
      putpixel(xt,yt,YELLOW);
   }

   for(int i=0;i<3;i++)
   {
    line(x[i],y[i],x[i+1],y[i+1]);
   }
}

//creating Hermite curve function
 void hermite_curve(int x1,int y1,int x2,int y2,double t1,double t4)
{
   float x,y,t;
    for(t=0.0;t<=1.0;t+=0.001)
    {
      //x and y equation
      x=(2*t*t*t-3*t*t+1)*x1+(-2*t*t*t+3*t*t)*x2+(t*t*t- 2*t*t+t)*t1+(t*t*t-t*t)*t4;
      y=(2*t*t*t-3*t*t+1)*y1+(-2*t*t*t+3*t*t)*y2+(t*t*t- 2*t*t+1)*t1+(t*t*t-t*t)*t4;
      putpixel(x,y,YELLOW);
    }
   putpixel(x1,y1,GREEN); putpixel(x2,y2,GREEN); line(x1,y1,x2,y2);
}

//main function
int main()
{
  int gd = DETECT , gm;
   initgraph(&gd, &gm, (char*)"");
```

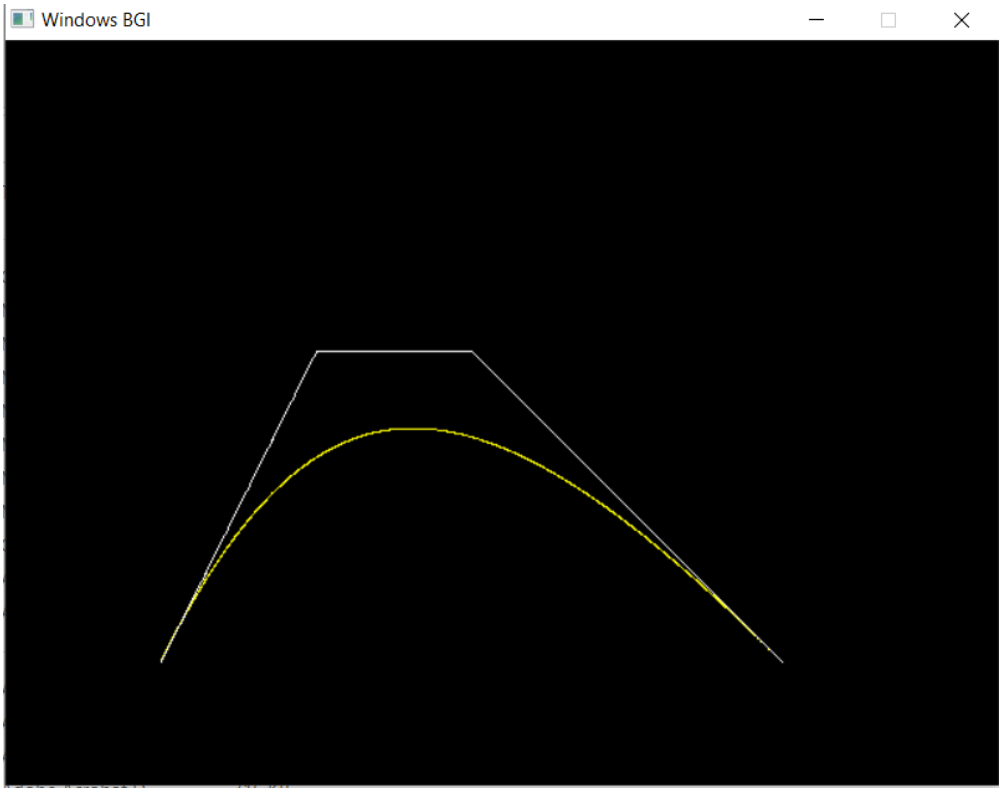
```
int x1 , y1 , x2 , y2 , n;
double t1,t4;

int x[4],y[4],i;

cout<<" 1.Bezier Curve \n 2.Hermite Curve\n";
cout<<"\n Enter your choice : ";
cin>>n;
if(n==1)
{
   //input coordinates of x and y for Bezier curve
   cout<<"Enter x and y coordinates \n";
    for(i=0;i<4;i++)
    {
      cout<<"x"<<i+1<<" : ";
      cin>>x[i];
      cout<<"y"<<i+1<<" : ";
      cin>>y[i];
      cout<<endl;
   }
//calling Bezier curve function
bezier_curve(x,y);
}

else if(n==2)
{  //input coordinates  for hermite curve
   cout<<"Enter the x coordinate of 1st hermite point : ";
    cin>>x1;
   cout<<"Enter the y coordinate of 1st hermite point : ";
   cin>>y1;
   cout<<"Enter the x coordinate of 4th hermite point : ";
   cin>>x2;
   cout<<"Enter the y coordinate of 4th hermite point : ";
   cin>>y2;
   cout<<"Enter tangent at p1 : ";
   cin>>t1;
   cout<<"Enter tangent at p4 : ";
   cin>>t4;
   //calling hermite curve function
   hermite_curve(x1,y1,x2,y2,t1,t4);
}
else
{
  cout<<"\n Invalid Choice";
}

getch();
```

```
    return 0;
}
```

OUTPUT:

BEZIER CURVE :

```
 1.Bezier Curve
 2.Hermite Curve

 Enter your choice : 1
Enter x and y coordinates
x1 : 100
y1 : 400

x2 : 200
y2 : 200

x3 : 300
y3 : 200

x4 : 500
y4 : 400
```



HERMITE CURVE :

```
 1.Bezier Curve
 2.Hermite Curve

 Enter your choice : 2
Enter the x coordinate of 1st hermite point : 200
Enter the y coordinate of 1st hermite point : 300
Enter the x coordinate of 4th hermite point : 300
Enter the y coordinate of 4th hermite point : 100
Enter tangent at p1 : 70
Enter tangent at p4 : 78
```