# A
# Practical File
# On

# Computer Graphics


## Bachelor of Engineering
## (2016-2020)
## Computer Science Engineering

**Submitted By**

Vrishank Gupta
COE SEM-IV
Sec-3
2016UCO1677

**DIVISION OF COMPUTER SCIENCE ENGINEERING**

**NETAJI SUBHAS INSTITUTE OF TECHNOLOGY, DWARKA**

# INDEX

## 1. Line drawing using DDA approach

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

void line_dda(int x1, int y1, int x2, int y2)
{
   int dx = x2 - x1;
   int dy = y2 - y1;

   int steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);

   float x = x1, y = y1;
   float xinc = dx / (float)steps;
   float yinc = dy / (float)steps;

   for (int i = 0; i <= steps; ++i) {
      putpixel(round(x), round(y), WHITE);
      x += xinc;
      y += yinc;
   }
}
int main()
{
   initwindow(800, 500);

   int x1, y1, x2, y2;
   cout << "Enter the x and y coordinates of the first point:
   "; cin >> x1 >> y1;
   cout << "Enter the x and y coordinates of the second point:
   "; cin >> x2 >> y2;
   line_dda(x1, y1, x2, y2);
   getch();
   return 0;
}
```

OUTPUT:

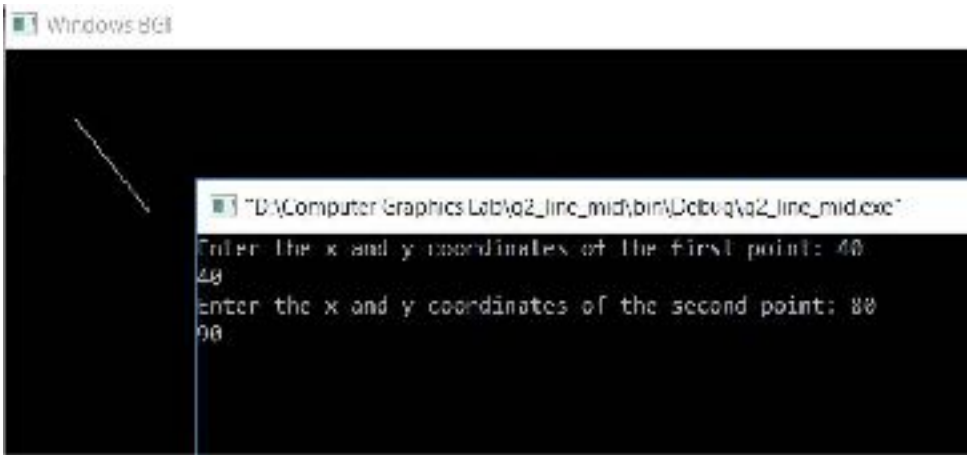
## 2. Line drawing using midpoint approach

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

//for drawing line using mid-point algorithm which handles all the
cases void line_mpt (int x1, int y1, int x2, int y2, int color = WHITE)
{

   int dx = abs(x2 - x1), dy = abs(y2 - y1), xsign, ysign;
   bool compare;

   if (abs (x2 - x1) > abs (y2 - y1)) {       //major moment in x
      if ((x2 - x1) * (y2 - y1) > 0) {     //slope is +ve: mx - y + c = 0
         xsign = -1, ysign = 1, compare = 1;
         if (x1 > x2) {
            swap(x1, x2); swap(y1, y2);
         }
      }
      else {     //slope is -ve: y + mx - c = 0
         xsign = 1, ysign = -1, compare = 0;
         if (x1 < x2) {
            swap(x1, x2); swap(y1, y2);
         }
      }

      int x = x1, y = y1;
      int del = (dy * ysign) + (dx * xsign) / 2;
      putpixel(x, y, color);

      while (x != x2) {
         x -= xsign;
         if ((compare ? del < 0 : del > 0)) {
            del += (dy * ysign);
         }
         else {
            del += ((dy * ysign) + (dx * xsign));
            y++;
         }
         putpixel(x, y, color);
      }
   }
   else {     //major moment in y
      if ((x2 - x1) * (y2 - y1) > 0) {     //slope is +ve: mx - y + c = 0
         xsign = -1, ysign = 1, compare = 1;
         if (x1 > x2) {
            swap(x1, x2); swap(y1, y2);
         }
      }
      else {     //slope is -ve: y + mx - c = 0
         xsign = 1, ysign = -1, compare = 0;
         if (x1 < x2) {
            swap(x1, x2); swap(y1, y2);
         }
      }
```

```
int x = x1, y = y1;
int del = (dx * xsign) + (dy * ysign) / 2;
```

```
        putpixel(x, y, color);

        while (y != y2) {
            y ++;
            if ((compare ? del > 0 : del < 0)) {
                del += (dx * xsign);
            }
            else {
                del += ((dx * xsign) + (dy * ysign));
                x -= xsign;
            }
            putpixel(x, y, color);
        }
    }
}

int main()
{
    initwindow(800, 500);
    int x1, y1, x2, y2;
    cout << "Enter the x and y coordinates of the first point:
    "; cin >> x1 >> y1;
    cout << "Enter the x and y coordinates of the second point:
    "; cin >> x2 >> y2;
    line_mpt(x1, y1, x2, y2);
    getch();
    return 0;
}
```

OUTPUT:



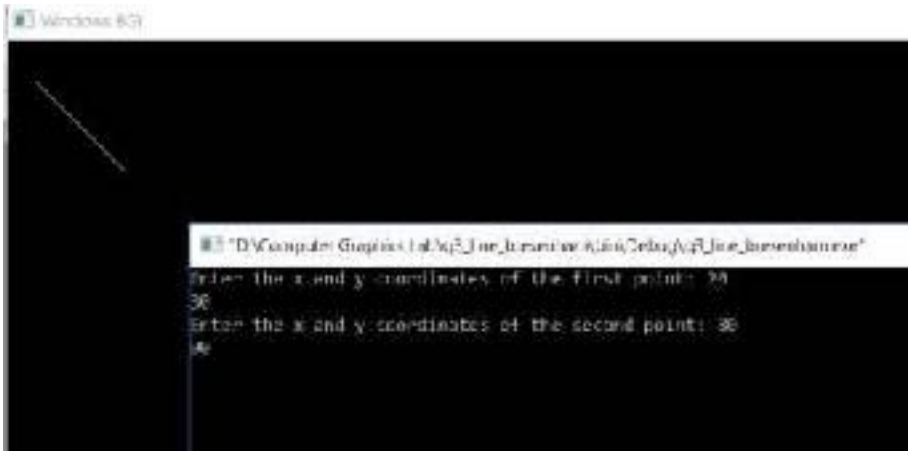
## 3. Drawing line using bresenham approach

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

using namespace std;

void myline (int x1, int y1, int x2, int y2)
{
   int dx = x2 - x1;
   int dy = y2 - y1;

   if (dy == 0) {
      while (x1 - x2) {
         putpixel (x1, y1, WHITE);
         if (dx > 0)
            x1++;
         else
            x1--;
      }
   }
   else if (dx == 0) {
      while (y1 - y2) {
         putpixel (x1, y1, WHITE);
         if (dy > 0)
            y1++;
         else
            y1--;
      }
   }

   else if (abs(dy) <= abs(dx)) {
      putpixel (x1, y1, WHITE);
      int d = 2 * abs(dy) - abs(dx);
      while (x1 - x2) {
         if (d <= 0) {
            d += 2 * (abs(dy));
         }
         else {
            d += 2 * (abs(dy) - abs(dx));
            if(dy > 0)
               y1++;
            else
               y1--;
         }
         if (dx > 0)
            x1++;
         else
            x1--;
         putpixel(x1, y1, WHITE);
      }
   }
   else if(abs(dx) < abs(dy)){
      putpixel(x1, y1, WHITE);
      int d = abs(dy) - 2 * abs(dx);
      while (y1 - y2) {
```

```
if (d > 0) {
```

```
            d += 2 * (-abs(dx));
        }
        else {
            d += 2 * (abs(dy) - abs(dx));
            if (dx > 0)
                x1++;
            else
                x1--;
        }
        if(dy > 0)
            y1++;
        else
            y1--;
        putpixel(x1, y1, WHITE);
    }
  }
}

int main()
{
  initwindow(800, 500);
  int x1, y1, x2, y2;
  cout << "Enter the x and y coordinates of the first point:
  "; cin >> x1 >> y1;
  cout << "Enter the x and y coordinates of the second point: ";
  cin >> x2 >> y2;
  myline(x1, y1, x2, y2);
  getch();
  return 0;
}
```

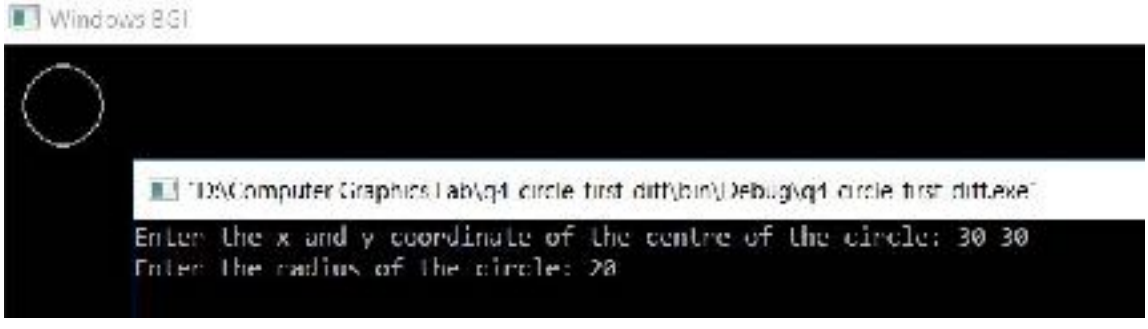
OUTPUT:



## 4. Circle drawing using first order differential approach (mid point approach)

```
#include <iostream>
#include <graphics.h>
using namespace std;
//for plotting 8 different points of circle using 8-
symmetry void pixel(int xc,int yc,int x,int y, int color) {

   putpixel(xc + x, yc + y, color);
   putpixel(xc + y, yc + x, color);
   putpixel(xc - y, yc + x, color);
   putpixel(xc - x, yc + y, color);
   putpixel(xc - x, yc - y, color);
   putpixel(xc - y, yc - x, color);
   putpixel(xc + y, yc - x, color);
   putpixel(xc + x, yc - y, color);
}
void circle_mpt(int xc, int yc, int r, int color = WHITE)
{
   int x = 0, y = r, d = 1 - r;
   pixel(xc, yc, x, y, color);
   while (x < y)
   {if (d < 0)
      {x++;
         d += (2 * x) + 3;
      }
      else
      {x++; y--;
               d += 2 * (x - y) + 5;
      }
      pixel(xc, yc, x, y, color);
   }}
int main()
{
   initwindow(800, 500);
   int cx, cy, r;
   cout << "Enter the x and y coordinate of the centre of the circle:
   "; cin >> cx >> cy;
  cout << "Enter the radius of the circle: ";
   cin >> r;
   circle_mpt(cx, cy, r);
   getch();
   return 0;
}
```

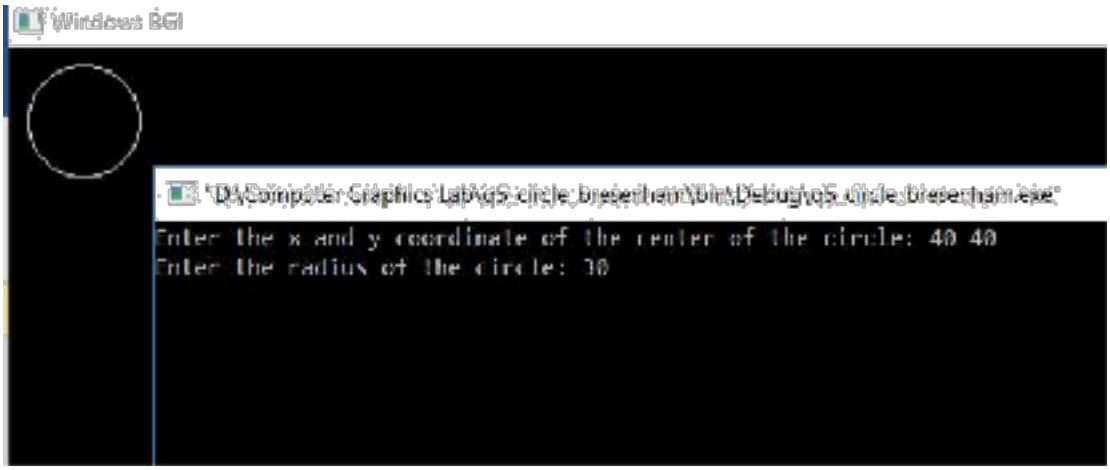
OUTPUT:



## 5. Circle drawing using Bresenham approach

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

using namespace std;

void drawCircle(int cx, int cy, int x, int y)
{
   putpixel(cx + x, cy + y, WHITE);
   putpixel(cx - x, cy + y, WHITE);
   putpixel(cx + x, cy - y, WHITE);
   putpixel(cx - x, cy - y, WHITE);
   putpixel(cx + y, cy + x, WHITE);
   putpixel(cx + y, cy - x, WHITE);
   putpixel(cx - y, cy - x, WHITE);
   putpixel(cx - y, cy + x, WHITE);
}

void mycircle(int cx, int cy, int r)
{
   int x = 0, y = r, d = 3 - 2 * r;
   drawCircle(cx, cy, x, y);

   while (x <= y) {
      if (d <= 0) {
         d += (4 * x + 6);
      }
      else {
         d += (4 * (x - y) + 10);
         y--;
      }
      x++;
      drawCircle(cx, cy, x, y);
   }
}

int main()
{
   initwindow(800, 500);

   int cx, cy, r;
   cout << "Enter the x and y coordinate of the center of the circle:
   "; cin >> cx >> cy;

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

   mycircle(cx, cy, r);

   getch();
   return 0;
}
```

OUTPUT:


Windbgi
"D:\Computer Graphics Lab\q5 circle bresenham\Bin\Debug\q5 circle bresenham.exe"
Enter the x and y coordinate of the center of the circle: 40 40
Enter the radius of the circle: 30

## 6. Pattern

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

void putting_pixel(int a0, int a1, int c1, int x_pivot, int y_pivot, int angle)
{
   int x_shifted = a0 - x_pivot;
   int y_shifted = a1 - y_pivot;
   a0 = x_pivot + (x_shifted*cos(angle) - y_shifted*sin(angle));
   a1 = y_pivot + (x_shifted*sin(angle) +
   y_shifted*cos(angle)); putpixel(a0, a1, c1);
}

void DrawCircle ( int cen_x , int cen_y , int Rad , int clr , float ang , int part)
{
    int X , Y , r ,d ;

  r = Rad;

  X = 0 ;
  Y = r ;
  d = 1 - r;

  ang = ang * (3.14/180) ;

  putting_pixel(X + 320 , Y + 240 , WHITE,320,240,ang);

  while ( X <= Y )
  {
     if ( d < 0 )
     {
        d += 2*X + 3;
     }
     else
     {
        d.+= 2*(X-Y)
        +5; Y--;
     }
     X++;

     //  putpixel( x+320+cen_x , -y+240+cen_y , clr ) ;
     //  putpixel( y+320+cen_x , -x+240+cen_y , clr ) ;
     //  putpixel( y+320+cen_x , x+240+cen_y , clr ) ;
     //  putpixel( x+320+cen_x , y+240+cen_y , clr ) ;
     //  putpixel( -x+320+cen_x , y+240+cen_y , clr ) ;
     //  putpixel( -y+320+cen_x , x+240+cen_y , clr ) ;
     //  putpixel( -y+320+cen_x , -x+240+cen_y , clr ) ;
     //  putpixel( -x+320+cen_x , -y+240+cen_y , clr
     ) ; float x,y;
     //float x_,y_;
    // x_ = 320 + cen_x ;
    // y_ = +240 + cen_y ;// X*sin(ang) + Y*cos(ang);
```

```
    //x = X*cos(ang) + Y*sin(ang);
    //y = -X*sin(ang) + Y*cos(ang);

   x = X ;
   y = Y;
      if (part==1)
      {
         putting_pixel( x+320 + cen_x , -y+240 + cen_y , clr ,320 , 240 ,
         ang) ; putting_pixel( y+320 + cen_x , -x+240 + cen_y , clr ,320 ,
         240 ,ang) ; putting_pixel( y+320 + cen_x , x+240 + cen_y , clr, 320
         , 240 , ang ) ; putting_pixel( x+320 + cen_x , y+240 + cen_y , clr ,
         320 , 240 , ang) ;
      }
      else if(part==2)
      {
         putting_pixel( y+320 + cen_x , x+240 + cen_y , clr ,320 , 240 ,
         ang) ; putting_pixel( x+320 + cen_x , y+240 + cen_y , clr ,320 ,
         240 , ang) ; putting_pixel( -x+320 + cen_x , y+240 + cen_y , clr ,
         320 , 240 , ang) ; putting_pixel( -y+320 + cen_x , x+240 + cen_y ,
         clr ,320 , 240 , ang) ;

      }
      else if (part==3)
      {
         putting_pixel( -x+320 + cen_x , y+240 + cen_y , clr ,320 , 240 ,
         ang) ; putting_pixel( -y+320 + cen_x , x+240 + cen_y , clr ,320 ,
         240 , ang) ; putting_pixel( -y+320 + cen_x , -x+240 + cen_y , clr ,
         320 , 240 , ang) ; putting_pixel( -x+320 + cen_x , -y+240 + cen_y ,
         clr ,320 , 240 , ang) ;
      }
      else if (part==4)
      {
         putting_pixel( x+320 + cen_x , -y +240 + cen_y , clr ,320 , 240 ,
         ang) ; putting_pixel( y+320 + cen_x , -x+240 + cen_y , clr ,320 ,
         240 , ang) ; putting_pixel( -y+320 + cen_x , -x+240 + cen_y , clr ,
         320 , 240 , ang) ; putting_pixel( -x+320 + cen_x , -y+240 + cen_y ,
         clr ,320 , 240 , ang) ;
      }
  }
}
int main()
{
   cout << "Hello world!" << endl;

    int gd = DETECT,gm;//left=100,top=100,right=200,bottom=200,x= 300,y=150,radius=50;

    initgraph(&gd, &gm, "C:\\TC\\BGI");
    int angle = 0 ;

    DrawCircle(0,-40,40,0xffffff,angle,3);
    DrawCircle(0,-60,20,0xffffff,angle,1);
    DrawCircle(0,-20,20,0xffffff,angle,3);
    DrawCircle(0,40,40,0xffffff,angle,1);
    DrawCircle(0,60,20,0xffffff,angle,3);
    DrawCircle(0,20,20,0xffffff,angle,1);
    DrawCircle(-40,0,40,0xffffff,angle,2);
    DrawCircle(-60,0,20,0xffffff,angle,4);
```

```
DrawCircle(-20,0,20,0xffffff,angle,2);
DrawCircle(40,0,40,0xffffff,angle,4);
DrawCircle(60,0,20,0xffffff,angle,2);
DrawCircle(20,0,20,0xffffff,angle,4);
```

```
    //angle--;



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

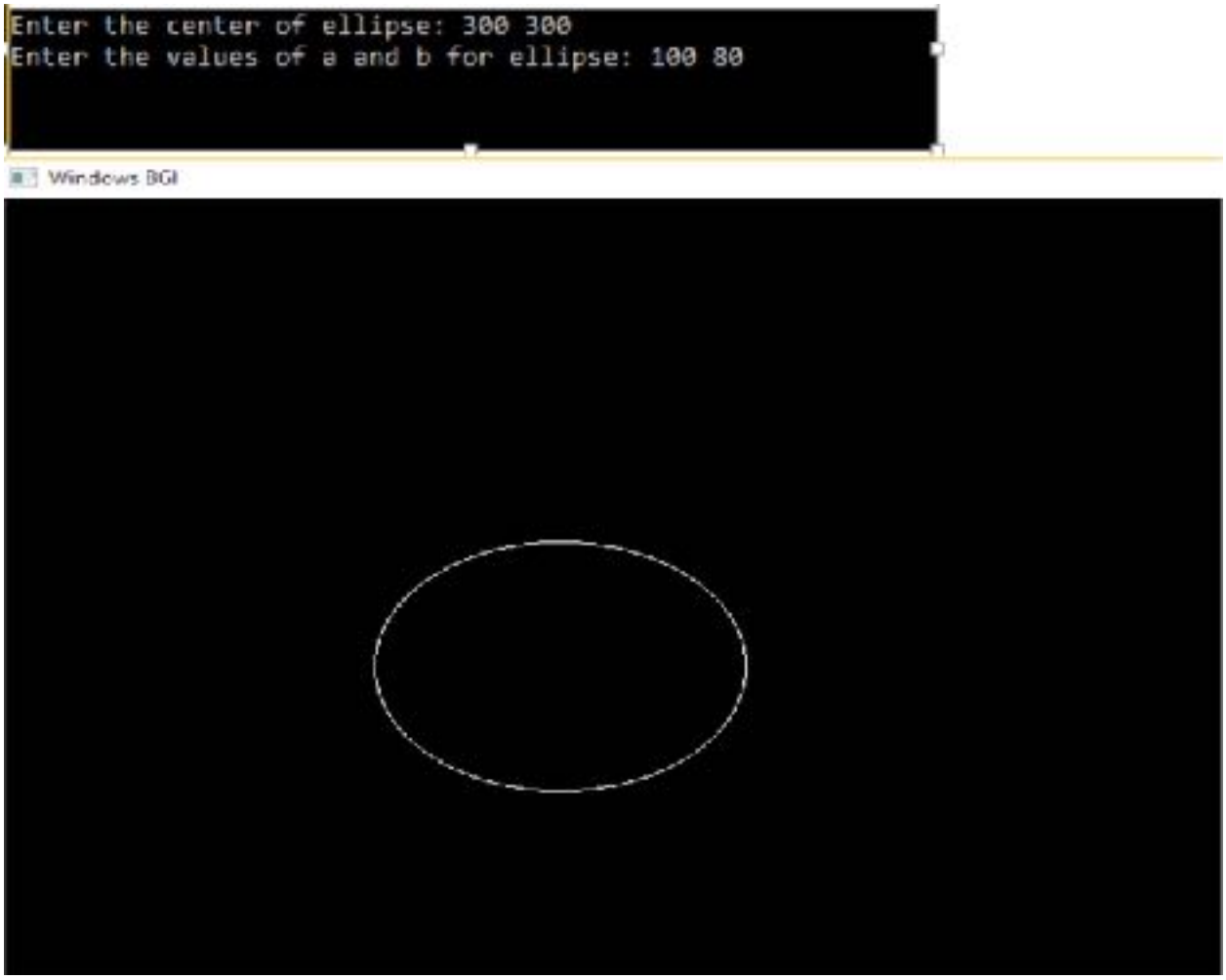
OUTPUT:

## 7. Ellipse drawing using Bresenham approach.

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

using namespace std;

void drawEllipse(int cx, int cy,int x, int y)
{
   putpixel(cx + x, cy - y, WHITE);
   putpixel(cx + x, cy + y, WHITE);
   putpixel(cx - x, cy - y, WHITE);
   putpixel(cx - x, cy + y, WHITE);
}

void myellipse(int cx, int cy, int a, int b)
{
   int x = 0, y = b;
   int d = 2 * b * b + a * a - 2 * a * a * b;
   drawEllipse(cx, cy, x, y);

   while (a * a * y > x * b * b) {
      if (d > 0) {
         d += (2 * b * b * (2 * x + 3) - 4 * a * a * (y - 1));
         y--;
      }
      else {
         d += (2 * b * b * (2 * x + 3));
      }
      x++;
      drawEllipse(cx, cy, x, y);

   }

   d = 2 * b * b * x * x + b * b + 2 * b * b * x + 2 * a * a * y * y + 2 * a * a - 4 * a * a * y - 2
* a * a * b * b;

  while (y >= 0) {
      if (d < 0) {
         d += (4 * b * b * (x + 1) - 2 * a * a * (2 * y - 3));
         x++;
      }
      else {
         d += 2 * a * a * (3 - 2 * y);
      }
      y--;
      drawEllipse(cx, cy, x, y);
   }
}

int main()
{
   initwindow(800, 500);

   int a, b, cx, cy;
   cout << "Enter the center of ellipse: ";
   cin >> cx >> cy;
```

```
    cout << "Enter the values of a and b for ellipse: ";
    cin >> a >> b;

    myellipse(cx, cy, a, b);

    getch();
    return 0;
}
```

OUTPUT:



## 8. Line clipping using Cyrus-Becker algorithm

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

using namespace std;

int main()
{
   int gdriver = DETECT, gmode, errorcode;
   initgraph(&gdriver, &gmode, "C:/TURBOC3/
   BGI"); errorcode = graphresult();

   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
      grapherrormsg(errorcode)); printf("Press any key to
      halt:"); getch();
      exit(1);
   }

   int wind[6][2]= {{30,150},{25,60},{80,70},{30,50},{100,40},
   {10,10}}; int point[2][2]= {{10,120},{80,-20}}; int inters[6]
   [2];
   int i;

   int edge[6][2];
   for(i=0; i<5; i++)
   {
      edge[i][0]=wind[i+1][0]-wind[i][0];
      edge[i][1]=wind[i+1][1]-wind[i][1];
   }
   edge[5][0]=wind[0][0]-wind[5][0];
   edge[5][1]=wind[0][1]-wind[5][1];

   int nor[6][2];
   for( i=0; i<6; i++)
   {
      nor[i][0]=-edge[i][1];
      nor[i][1]=edge[i][0];

   }
   float num[6],den[6],t[6];

   for( i=0; i<6; i++)
   {

      float numx=(point[0][0]-wind[i][0])*(nor[i][0]);
      float numy=(point[0][1]-wind[i][1])*(nor[i][1]);
      num[i]=numx+numy;
      float denx=((nor[i][0])*(point[1][0]-point[0][0]));
      float deny=((nor[i][1])*(point[1][1]-point[0][1]));
```

```
den[i]=-(denx+deny);
```

```
        t[i]=num[i]/den[i];
    }

    setcolor(RED);
    for(i=0; i<5; i++)
    {
        line(360+wind[i][0],240-wind[i][1],360+wind[i+1][0],240-wind[i+1][1]);
    }
    line(360+wind[5][0],240-wind[5][1],360+wind[0][0],240-wind[0][1]);

    for(i=0; i<6; i++)
    {
        inters[i][0]=point[0][0]+(point[1][0]-point[0][0])*t[i];
        inters[i][1]=point[0][1]+(point[1][1]-point[0][1])*t[i];
    }

    setcolor(YELLOW);
    line(360+inters[1][0],240-inters[1][1],360+inters[2][0],240-inters[2][1]);
    line(360+inters[3][0],240-inters[3][1],360+inters[4][0],240-inters[4][1]);
    line(360+inters[5][0],240-inters[5][1],360+inters[0][0],240-inters[0][1]);

    setcolor(CYAN);
    line(360+point[0][0],240-point[0][1],360+inters[5][0],240-inters[5][1]);
    line(360+inters[0][0],240-inters[0][1],360+inters[1][0],240-inters[1][1]);
    line(360+inters[2][0],240-inters[2][1],360+inters[3][0],240-inters[3][1]);
    line(360+point[1][0],240-point[1][1],360+inters[4][0],240-inters[4][1]);


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

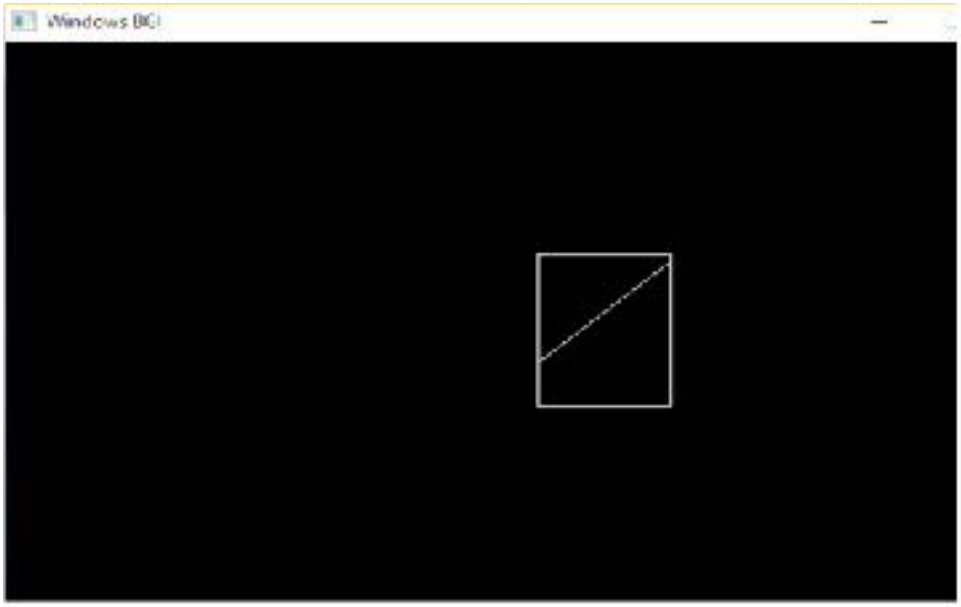
OUTPUT:

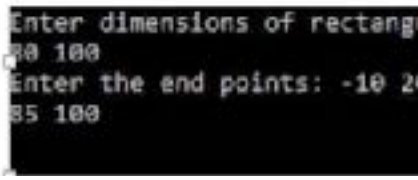
## 9. Line clipping using Cohen-Sutherland approach

```
#include <iostream>
#include <graphics.h>
#define LEFT 1
#define BOTTOM 2
#define RIGHT 4
#define TOP 8

using namespace std;

struct point
{
   float x,y;
};

float xmin,ymin,xmax,ymax;

int code(point a)
{
   int reg=0;

   if(a.x<xmin)
      reg=reg|LEFT;
   if(a.x>xmax)
      reg=reg|RIGHT;
   if(a.y<ymin)
      reg=reg|BOTTOM;
   if(a.y>ymax)
      reg=reg|TOP;
   return reg;
}

int main()
{
   cout << "Enter dimensions of rectangular window: ";
   cin >> xmin >> ymin >> xmax >> ymax;
   cout << "Enter the end points: ";
   point ini,fin;
   cin >> ini.x >> ini.y >> fin.x >> fin.y;
   float m=(fin.y-ini.y)/(fin.x-ini.x);
   int gd=DETECT,gm;
   initgraph(&gd,&gm,"C:/TC/BGI");
   setcolor(RED);
   line(xmin+320,240-ymin,320+xmax,240-ymin);
   line(xmax+320,240-ymin,320+xmax,240-ymax);
   line(320+xmax,240-ymax,320+xmin,240-ymax);
   line(320+xmin,240-ymax,320+xmin,240-ymin);
   setcolor(WHITE);
   while (1) {
      int r1=code(ini);
      int r2=code(fin);
      if ((r1&r2)!=0) {
         break;
      }
      if((r1|r2)==0) {   //visible
```

```
            line(ini.x+320,240-ini.y,320+fin.x,240-fin.y);
            break;
        }
        //partially
        if(r1==0) { //ensuring r1 always has non centre coordinate
            int temp=r1;
            r1=r2;
            r2=temp;
            point t=ini;
            ini=fin;
            fin=t;
        }
        if(r1&LEFT) {
            float ynew=m*(xmin-ini.x)+ini.y;
            ini.y=ynew;
            ini.x=xmin;
        }
        else if(r1&RIGHT) {
            float ynew=m*(xmax-ini.x)+ini.y;
            ini.y=ynew;
            ini.x=xmax;
        }
        else  if(r1&BOTTOM) {
            float xnew=(1/m)*(ymin-ini.y)+ini.x;
            ini.y=ymin;
            ini.x=xnew;
        }
         else  if(r1&TOP) {
            float xnew=(1/m)*(ymax-ini.y)+ini.x;
            ini.y=ymax;
            ini.x=xnew;
        }
    }
    getch();
    return 0;
}
```

OUTPUT:



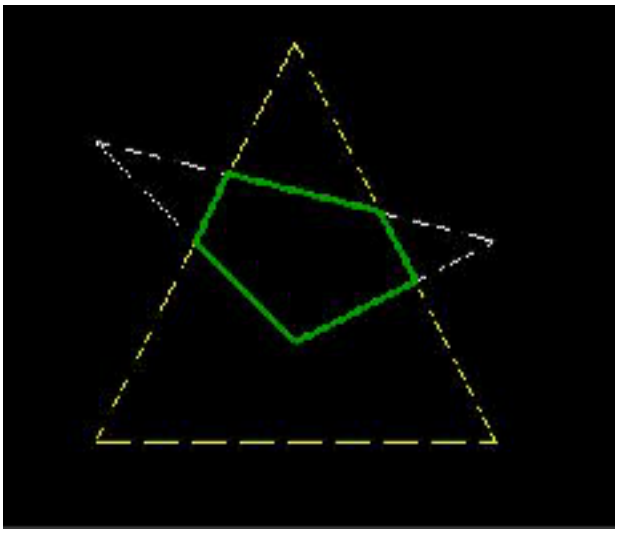
## 10. Polygon clipping via Sutherland-Hodgeman algorithm

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

using namespace std;

const int MAX_POINTS = 20;

int x_intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
   int num = (x1*y2 - y1*x2) * (x3-x4) - (x1-x2) * (x3*y4 -
   y3*x4); int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);

   return num/den;
}

int y_intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
   int num = (x1*y2 - y1*x2) * (y3-y4) - (y1-y2) * (x3*y4 -
   y3*x4); int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);

   return num/den;
}

void clip(int poly_points[][2], int &poly_size, int x1, int y1, int x2, int y2)
{
   int new_points[MAX_POINTS][2], new_poly_size = 0;

   for (int i = 0; i < poly_size; i++) {
      int k = (i+1) % poly_size;
      int ix = poly_points[i][0], iy = poly_points[i][1];
      int kx = poly_points[k][0], ky = poly_points[k][1];

      int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix-x1);
      int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);

      if (i_pos < 0 && k_pos < 0) {
         new_points[new_poly_size][0] = kx;
         new_points[new_poly_size][1] = ky;
         new_poly_size++;
      }
      else if (i_pos >= 0  && k_pos < 0) {
         new_points[new_poly_size][0] = x_intersect(x1, y1, x2, y2, ix, iy, kx,
         ky); new_points[new_poly_size][1] = y_intersect(x1, y1, x2, y2, ix,
         iy, kx, ky); new_poly_size++;
         new_points[new_poly_size][0] = kx;
         new_points[new_poly_size][1] = ky;
         new_poly_size++;
      }
      else if (i_pos < 0  && k_pos >= 0) {
         new_points[new_poly_size][0] = x_intersect(x1, y1, x2, y2, ix, iy, kx,
         ky); new_points[new_poly_size][1] = y_intersect(x1, y1, x2, y2, ix,
         iy, kx, ky); new_poly_size++;
```

}

```
        else {
            //No points are added
        }
    }

    poly_size = new_poly_size;
    for (int i = 0; i < poly_size; i++) {
        poly_points[i][0] = new_points[i][0];
        poly_points[i][1] = new_points[i][1];
    }
}

void suthHodgClip(int poly_points[][2], int poly_size, int clipper_points[][2], int
clipper_size) {
  for (int i=0; i<clipper_size; i++) {
       int k = (i+1) % clipper_size;
        clip(poly_points, poly_size, clipper_points[i][0], clipper_points[i][1], clipper_points[k]
[0], clipper_points[k][1]);
    }

    setlinestyle(0, 1, 3);
    setcolor(2);
    for (int i=0; i < poly_size; i++) {
        line(poly_points[i][0], poly_points[i][1],
poly_points[(i+1)%poly_size][0], poly_points[(i+1)%poly_size][1]);
    }
}

int main()
{
    initwindow(800, 500);
    int poly_size = 3;
    setlinestyle(3, 1, 1);
    int poly_points[20][2] = {{100,150}, {200,250},
    {300,200}}; for (int i=0; i < poly_size; i++) {
        line(poly_points[i][0], poly_points[i][1],
poly_points[(i+1)%poly_size][0], poly_points[(i+1)%poly_size][1]);
    }
     setcolor(14);
    int clipper_size = 3;
    int clipper_points[][2] = {{100,300}, {300,300},
    {200,100}}; for (int i=0; i < clipper_size; i++) {
        line(clipper_points[i][0], clipper_points[i][1], clipper_points[(i+1)%clipper_size]
[0], clipper_points[(i+1)%clipper_size][1]);
    }

    suthHodgClip(poly_points, poly_size, clipper_points, clipper_size);

    getch();
    return 0;
}
```

OUTPUT:



## 11. Polygon clipping via Weiler Atherton algorithm

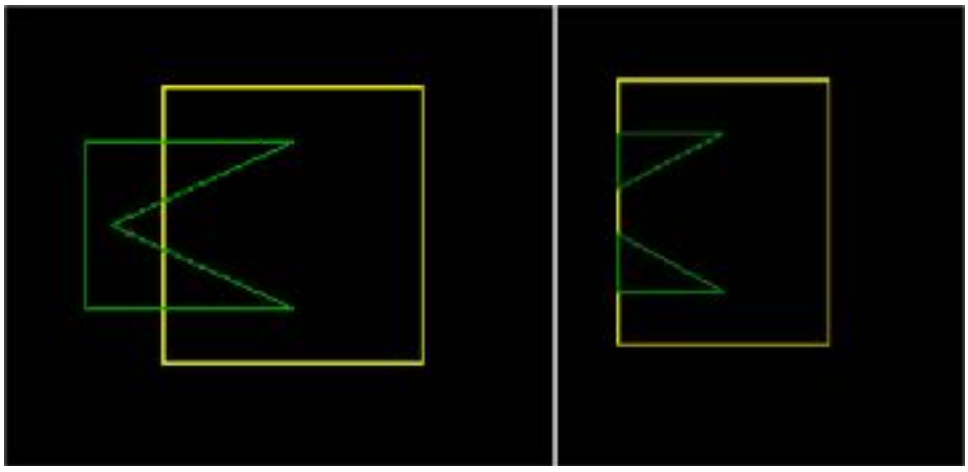
```
#include <bits/stdc++.h>
#include <graphics.h>

using namespace std;

float sdx[15],sdy[15];
int i,w=0,h;

void sort(float sdy[],int h)
{
   float temp;
   for(int j=0;j<=h-1;j++)
   {
      for(i=0;i<h-1-j;i++)
      {
         if(sdy[i]>sdy[i+1])
         {
            temp=sdy[i];
            sdy[i]=sdy[i+1];
            sdy[i+1]=temp;
         }
      }
   }
}

struct points
{
   float x;
   float y;
   float io;
   float vis;
};
struct points z[20];

int main()
{
   initwindow(640, 480);
   int n,m,s;
   float px[15]={0};
   float py[15]={0};
   float pdx[15],pdy[10];
   float outx[15]={0};
   float outy[15]={0};
   float xmin,ymin,xmax,ymax;

   cout<<"\nEnter xmin,ymin,xmax,ymax: ";
   cin>>xmin>>ymin>>xmax>>ymax;

   setcolor(YELLOW);
   rectangle(320+xmin,240-ymax,320+xmax,240-ymin);

   cout<<"\nEnter the no. of vertices (n): ";
   cin>>n;
   cout<<"\nEnter the x coordinate of all vertices: ";
```

```
for(m=0;m<n;m++)
{
   cin>>px[m];
}
cout<<"\nEnter the y coordinate of all vertices: ";
cout<<"\nEnter the y coordinate of all vertices: ";
for(m=0;m<n;m++)
{
   cin>>py[m];
}

setcolor(GREEN);
px[n]=px[0];py[n]=py[0];
for(s=0;s<n;s++)
{
   line(320+px[s],240-py[s],320+px[s+1],240-py[s+1]);
}

getch();
cleardevice();
getch();

px[n]=px[0];
py[n]=py[0]; int l=0;
for(m=0;m<n;m++)
{
   if(px[m]>=xmin && px[m+1]<=xmin)
   {
      pdx[m]=xmin;
      pdy[m]=py[m]+((py[m+1]-py[m])/(px[m+1]-px[m]))*(xmin-px[m]);
      outx[l]=pdx[m];outy[l]=pdy[m];
      z[l].io=1;
      l++;
   }
   if(px[m]>=xmin && px[m+1]>=xmin)
   {
      outx[l]=px[m+1];outy[l]=py[m+1];
      z[l].io=0;
      l++;
   }
   if(px[m]<=xmin && px[m+1]>=xmin)
   {
      pdx[m]=xmin;
      pdy[m]=py[m]+((py[m+1]-py[m])/(px[m+1]-px[m]))*(xmin-px[m]);
      outx[l]=pdx[m];outy[l]=pdy[m];
      z[l].io=0;
      l++;
      outx[l]=px[m+1];outy[l]=py[m+1];
      z[l].io=0;
      l++;
   }
}

outx[l]=outx[0];outy[l]=outy[0];
setcolor(YELLOW);
rectangle(320+xmin,240-ymax,320+xmax,240-ymin);
```

```
setcolor(GREEN);

for(i=0;i<l;i++)
{
   if(outx[i]==xmin)
   {
      sdx[w]=outx[i];
      sdy[w]=outy[i];
      w++;
   }
}

sort(sdy,w);
outx[l]=outx[0];outy[l]=outy[0];
for(i=0;i<=l;i++)
{
   z[i].x=outx[i];
   z[i].y=outy[i];
   z[i].vis=0;
}

s=0;
for(m=0;m<=l-1;m++)
{
   outx[l]=outx[0];outy[l]=outy[0];
   sdx[w+1]=sdx[0];sdy[w+1]=sdy[0];

   if(z[s].io==0)
   {
      line(320+outx[s],240-outy[s],320+outx[s+1],240-outy[s+1]);
      z[s].vis=1;
      z[s+l].vis=1;
   }
   else if(z[s].io==1)
   {
      for(i=0;i<=w;i++)
      {
         if(sdy[i]==outy[s])
         {
            line(320+sdx[i],240-sdy[i],320+sdx[i+1],240-sdy[i+1]);
            z[s].vis=1;
            z[s+l].vis=1;
            break;
         }
      }
      for(int j=0;j<l;j++)
      {
         if(sdy[i+1]==z[j].y)
         {
            s=j;
            line(320+outx[s],240-outy[s],320+outx[s+1],240-outy[s+1]);
            z[s].vis=1;
            z[s+l].vis=1;
            break;
         }
      }
   }
```

```
        if(s<=l-1)
        {
           s++;
        }
        else
        {
           s=0;
        }
        if(s==l)
        {
           s=0;
        }

        int p=s;
        while(z[s].vis == 1)
        {
           s++;
           if(s==p+l)
           {
              break;
           }
        }
    }

    getch();
    return 0;
}
```

OUTPUT:

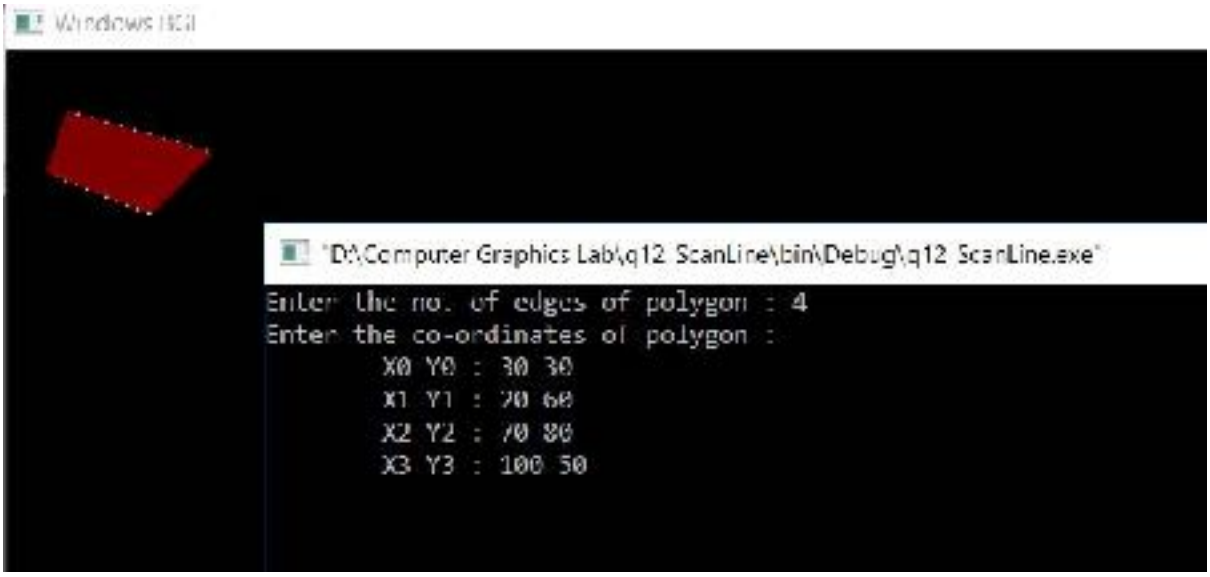


## 12. Polygon filling through Scanline approach

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

using namespace std;

int main()
{
   int n, i, j, k, dy, dx;
   int x, y, temp;
   int xv[20], yv[20], xi[20];
   float slope[20];

   cout << "Enter the no. of edges of polygon : ";
   cin >> n;
   cout << "Enter the co-ordinates of polygon :\n";
   for (i = 0; i < n; i++) {
      printf("\tX%d Y%d : ",i,i);
      scanf("%d %d",&xv[i],&yv[i]);
   }
   xv[n]=xv[0];
   yv[n]=yv[0];

   initwindow(800, 500);

   /* draw polygon */
   for (i = 0; i < n; i++) {
      line(xv[i],yv[i],xv[i+1],yv[i+1]);
      delay(50);
   }

   for (i = 0; i < n; i++) {
      dy = yv[i+1] - yv[i];
      dx = xv[i+1] - xv[i];
      if(dy==0)
         slope[i]=1.0;
      if(dx==0)
         slope[i]=0.0;
      if ((dy != 0) && (dx != 0)) {/*- calculate inverse
         slope -*/ slope[i] = (float) dx / dy;
      }
   }
   for(y = 0; y < 480; y++) {
      k = 0;
      for (i=0;i<n;i++) {
         if (((yv[i] <= y) && (yv[i+1] > y)) || ((yv[i] > y) && (yv[i+1]
            <= y))) { xi[k] = (int)(xv[i] + slope[i] * (y - yv[i]));
            k++;
         }
      }
      for(j = 0; j < k - 1; j++) {/*- Arrange x-intersections in order -*/
```

```
            for(i = 0; i < k - 1; i++) {
               if(xi[i] > xi[i+1]) {
                  temp  =xi[i];
                  xi[i] = xi[i+1];
                  xi[i+1] = temp;
               }
            }
            setcolor(4);
            for(i = 0; i < k; i += 2) {
               line(xi[i], y, xi[i+1] + 1, y);
               delay(10);
            }
         }
      }

      getch();
      return 0;
}
```

OUTPUT:

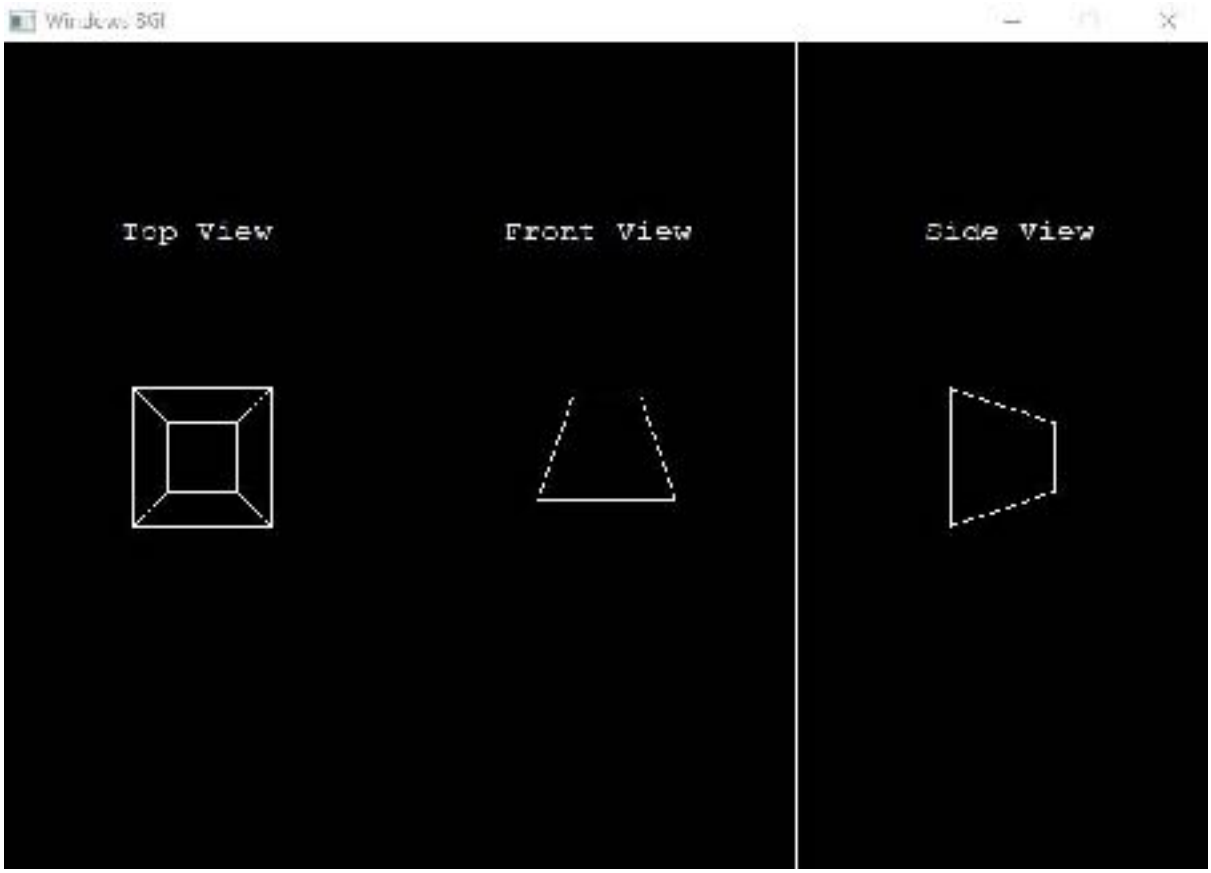


## 13. Demonstrating 3D transformations

```
#include<graphics.h>
#include<bits/stdc++.h>

using namespace std;

int main()
{
   initwindow(700, 480);

   int p[4][8] = {   {-40,40,40,-40,-20,20,20,-20},
                     {40,40,-40,-40,20,20,-20,-20},
                     {0,0,0,0,60,60,60,60},
                     {1,1,1,1,1,1,1,1} };

   float pr[4][8];
   int i, j, k, page = 0;
   float q = 0.0, sum = 0, d = 0;
   settextstyle(8, HORIZ_DIR, 1);

   while(1)
   {
      setactivepage(page); setvisualpage(1 - page); cleardevice();
      q=((d*(22/7.0))/180.0);
      float rotatez[4][4]={ {cos(q), -sin(q), 0, 0},
                            {sin(q), cos(q), 0, 0},


                     {0,                    0,                    1,    0},


                     {0,                    0,                    0,    1}  };


      for (i = 0; i <= 3; i++) {
         for (j = 0; j <= 8; j++) {
            sum = 0;
            for (k = 0; k <= 3; k++) {
               sum += rotatez[i][k] * p[k][j];
            }
            pr[i][j] = sum;
         }
      }

      //top view
      outtextxy(68, 100, "Top View");
      for(i=0;i<4;i++) {
         line(115+pr[0][i],240-pr[1][i],115+pr[0][(i+1)%4],240-pr[1][(i+1)%4]);
         line(115+pr[0][i+4],240-pr[1][i+4],115+pr[0][(i+1)%4+4],240-pr[1]
         [(i+1)%4+4]); line(115+pr[0][i],240-pr[1][i],115+pr[0][i+4],240-pr[1][i+4]);
      }
      line(230, 0, 230, getmaxy());

      //front view
```

```
        outtextxy(290, 100, "Front View");
        for(i=0;i<4;i++) {
           line(350+pr[0][i],265-pr[2][i],350+pr[0][(i+1)%4],265-pr[2][(i+1)%4]);
           line(350+pr[0][i+4],265-pr[2][i+4],350+pr[0][(i+1)%4+4],265-pr[2]
           [(i+1)%4+4]); line(350+pr[0][i],265-pr[2][i],350+pr[0][i+4],265-pr[2][i+4]);
        }
        line(460, 0, 460, getmaxy());
        //side view
        outtextxy(535, 100, "Side View");
        for(i=0;i<4;i++) {
           line(550+pr[2][i],240-pr[1][i],550+pr[2][(i+1)%4],240-pr[1][(i+1)%4]);
           line(550+pr[2][i+4],240-pr[1][i+4],550+pr[2][(i+1)%4+4],240-pr[1]
           [(i+1)%4+4]); line(550+pr[2][i],240-pr[1][i],550+pr[2][i+4],240-pr[1][i+4]);
        }

        page = 1 - page,  d++;
        delay(5);
     }

   getch();
   return 0;
}
```

OUTPUT:

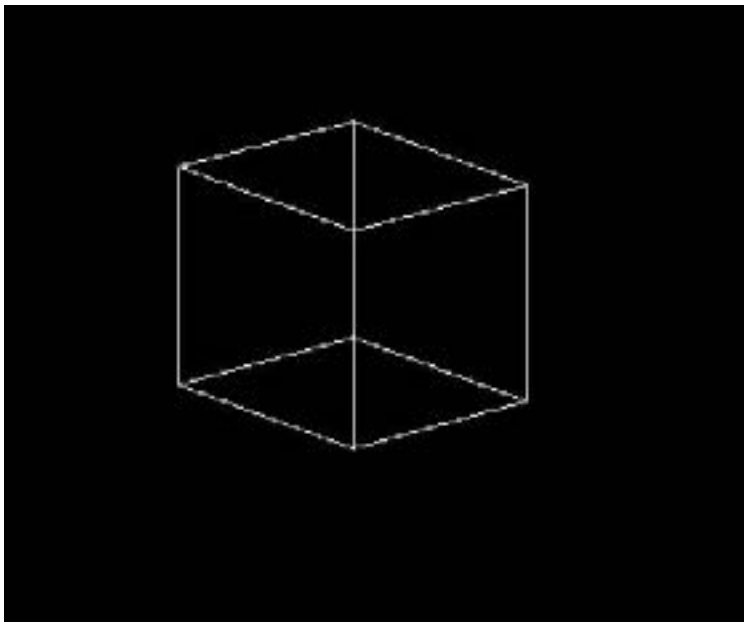


## 14. Demonstrating the isometric view of a cube

```
#include<iostream>
#include<graphics.h>
#include<conio.h>
#include<windows.h>
#include<cmath>

using namespace std;

void makecubeFront(float obj[8][4])
{
   for(int i=0;i<4;i++){
      line(210+obj[i][0],240-obj[i][1],210+obj[(i+1)%4][0],240-obj[(i+1)%4][1]);
      line(210+obj[i+4][0],240-obj[i+4][1],210+obj[((i+1)%4)+4][0],240-obj[((i+1)%4)+4][1]);
      line(210+obj[i][0],240-obj[i][1],210+obj[i+4][0],240-obj[i+4][1]);
   }
}

void multiply(float obj[][4],float trans[4][4], int rows)
{
      float mul[rows][4];
      for(int i=0;i<rows;i++){
         for(int j=0;j<4;j++){
            float sum=0;
            for(int k=0;k<4;k++){
               sum=sum+(obj[i][k]*trans[k][j]);
            }
            mul[i][j]=sum;
         }
      }
      for(int i=0;i<rows;i++){
         for(int j=0;j<4;j++)
            obj[i][j]=mul[i][j];
      }
}

int main()
{
   float pi=3.14159265;
   float
obj[8]
[4]={0,120,0,0,120,120,0,0,120,0,0,0,0,0,0,0,120,120,0,120,120,120,0,120,0,120,0,0,0,120,
0};
   int gd = DETECT,gm;
   initgraph(&gd, &gm, "C:\\TC\\BGI");
   //initwindow(1720,720);
  // float isometric[4][4]={0.7071,0,-0.7071,0,-0.40825,0.8165,-
0.40825,0,0.7071,0.40825,0.8165,0,0,0,0,1}; wrong matrix (actually transpose)
   float isometric[4][4]={0.7071,-0.40825,0.7071,0,0,0.8165,0.40825,0,-0.7071,-
0.40825,0.8165,0,0,0,0,1};
   multiply(obj,isometric,8); ///If you want rotation wrt any axis, uncomment the code
below

   float       rotx[4][4]={1,0,0,0,0,cos(pi/180),-1*sin(pi/180),0,0,sin(pi/180),cos(pi/180),
   0,0,0,0,1};    float    roty[4][4]={cos(pi/180),0,sin(pi/180),0,0,1,0,0,-1*sin(pi/180),0,cos(pi/
```

```
    180),0,0,0,0,1};   float   rotz[4][4]={cos(pi/180),-1*sin(pi/180),0,0,sin(pi/180),cos(pi/180),
    0,0,0,0,1,0,0,0,0,1}; for(int theta=0;theta<90;theta++){
        multiply(obj,rotx,8);
        setcolor(WHITE);
        makecubeFront(obj);
        Sleep(100);
        setcolor(BLACK);
        makecubeFront(obj);
    }
    for(int theta=0;theta<90;theta++){
        multiply(obj,roty,8);
        setcolor(WHITE);
        makecubeFront(obj);
        Sleep(100);
        setcolor(BLACK);
        makecubeFront(obj);
    }
    for(int theta=0;theta<90;theta++){
        multiply(obj,rotz,8);
        setcolor(WHITE);
        makecubeFront(obj);
        Sleep(100);
        setcolor(BLACK);
        makecubeFront(obj);
    }
    setcolor(WHITE);
    makecubeFront(obj);
    getch();
    closegraph();
    return 0;
}
```

OUTPUT:



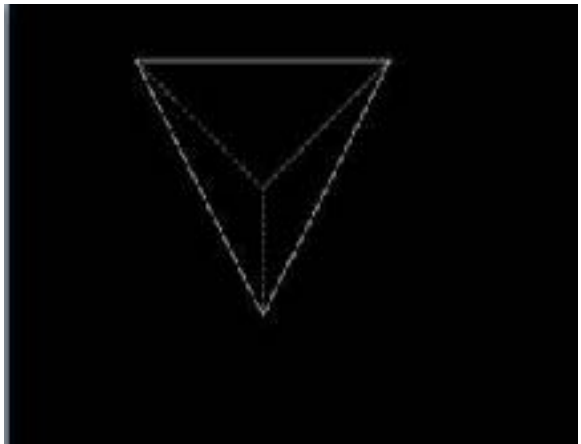
## 15. Hidden surface elimination using back face detection.

```
#include<math.h>
#include<graphics.h>
#define pi 3.14

float prism[4][8]= {{0, 80, 80, 0, 20, 60, 60, 20},
                    {80, 80, 0, 0, 60, 60, 20, 20},
                    {0, 0, 0, 0, -60, -60, -60, -60},
                    {1, 1, 1, 1, 1, 1, 1, 1} };

float dot(float n1[],float n2[])
{
   int i;
   float ans=0;
   for(i=0; i<3; i++) {
      ans+=(n1[i]*n2[i]);
   }
   return ans;
}

float mag(float n[])
{
   return(sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]));
}

void normal(float n[],int a,int b,int c,int s1,int s2)
{
   int i;
   float n1[3],n2[3],n3[3],v[3],cos;
   for(i=0; i<3; i++) {
      n1[i]=prism[i][a];
      n2[i]=prism[i][b];
      n3[i]=prism[i][c];
      v[i]=prism[i][s1]-prism[i][s2];
   }
   n[0]=((n1[1]-n2[1])*(n2[2]-n3[2]))-((n1[2]-n2[2])*(n2[1]-n3[1]));
   n[1]=((n1[2]-n2[2])*(n2[0]-n3[0]))-((n1[0]-n2[0])*(n2[2]-n3[2]));
   n[2]=((n1[0]-n2[0])*(n2[1]-n3[1]))-((n1[1]-n2[1])*(n2[0]-n3[0]));
   cos=dot(n,v)/(mag(n)*mag(v));
   if(cos>0)
      for(i=0; i<3; i++)
         n[i]=n[i]*(-1);
}

void mp(float n[],int a,int b)
{
   int i;
   for(i=0; i<3; i++)
      n[i]=(prism[i][a]+prism[i][b])/2;
   n[2]-=32768;
}

void fline(int x0,int y0,int x1,int y1,int x2,int y2,int c1=15,int style=0)
```

```
{
   setcolor(c1);
   setlinestyle(style,1,1);
   line(x0+x1,y0-y1,x0+x2,y0-y2);
   setcolor(15);
   setlinestyle(0,1,1);
}

void surface(int x0,int y0,int t,float prism[][8],int color[])
{
   int i,j,style=0;
   float n[3],n2[3];
   if(t==0) {    //ABCD
      normal(n,0,1,2,4,0);
      mp(n2,0,2);
      if(dot(n,n2)>0)
         style=3;
      for(i=0; i<4; i++) {
         j=i+1;
         if(j==4)
            j=0;
         fline(x0,y0,prism[0][i],prism[1][i],prism[0][j],prism[1][j],color[t],style);
      }
      setfillstyle(SOLID_FILL,color[t]);
   }
   else if(t==1) {  //EFGH
      normal(n,4,5,6,0,4);
      mp(n2,4,6);
      if(dot(n,n2)>0)
         style=3;
      for(i=4; i<8; i++) {
         j=i+1;
         if(j==8)
            j=4;
         fline(x0,y0,prism[0][i],prism[1][i],prism[0][j],prism[1][j],color[t],style);
      }
      setfillstyle(SOLID_FILL,color[t]);
   }
   else if(t==2) {  //GFBC
      normal(n,6,5,1,4,5);
      mp(n2,6,1);
      if(dot(n,n2)>0)
         style=3;
      fline(x0,y0,prism[0][6],prism[1][6],prism[0][5],prism[1][5],color[t],style);//
      GF       fline(x0,y0,prism[0][5],prism[1][5],prism[0][1],prism[1]
      [1],color[t],style);      //FB      fline(x0,y0,prism[0][1],prism[1][1],prism[0]
      [2],prism[1][2],color[t],style);      //BC      fline(x0,y0,prism[0][2],prism[1]
      [2],prism[0][6],prism[1][6],color[t],style);       //CG
      setfillstyle(SOLID_FILL,color[t]);
   }
   else if(t==3) {//GCDH
      normal(n,6,2,3,5,6);
      mp(n2,6,3);
      if(dot(n,n2)>0)
         style=3;
      fline(x0,y0,prism[0][6],prism[1][6],prism[0][2],prism[1][2],color[t],style);//GC
      fline(x0,y0,prism[0][2],prism[1][2],prism[0][3],prism[1][3],color[t],style);//CD
```

```
        fline(x0,y0,prism[0][3],prism[1][3],prism[0][7],prism[1][7],color[t],style);//DH
        fline(x0,y0,prism[0][7],prism[1][7],prism[0][6],prism[1][6],color[t],style);//HG
        setfillstyle(SOLID_FILL,color[t]);
    }
    else if(t==4) {  //AEFB
        normal(n,4,5,1,6,5);
        mp(n2,1,5);
        if(dot(n,n2)>0)
            style=3;
        fline(x0,y0,prism[0][0],prism[1][0],prism[0][4],prism[1][4],color[t],style);//AE
        fline(x0,y0,prism[0][4],prism[1][4],prism[0][5],prism[1][5],color[t],style);//EF
        fline(x0,y0,prism[0][5],prism[1][5],prism[0][1],prism[1][1],color[t],style);//FB        fline(x0,y0,prism[0][1],prism[1][1],prism[0][0],prism[1][0],color[t],style);//BA setfillstyle(SOLID_FILL,color[t]);
    }
    else if(t==5) {  //EADH
        normal(n,4,0,3,5,4);
        mp(n2,4,3);
        if(dot(n,n2)>0)
            style=3;
        fline(x0,y0,prism[0][4],prism[1][4],prism[0][0],prism[1][0],color[t],style);//EA
        fline(x0,y0,prism[0][0],prism[1][0],prism[0][3],prism[1][3],color[t],style);//AD
        fline(x0,y0,prism[0][3],prism[1][3],prism[0][7],prism[1][7],color[t],style);//DH
        fline(x0,y0,prism[0][7],prism[1][7],prism[0][4],prism[1][4],color[t],style);//HE
        setfillstyle(SOLID_FILL,color[t]);
    }
}
void front(float prism[][8],int x0=320,int y0=240)
{
    int i;
    int color[6]= {15,15,15,15,15,15};
    for(i=0; i<6; i++)
        surface(x0,y0,i,prism,color);
}

int main()
{
    initwindow(640,480);
    front(prism);

    getch();
    return 0;
}
```

OUTPUT:

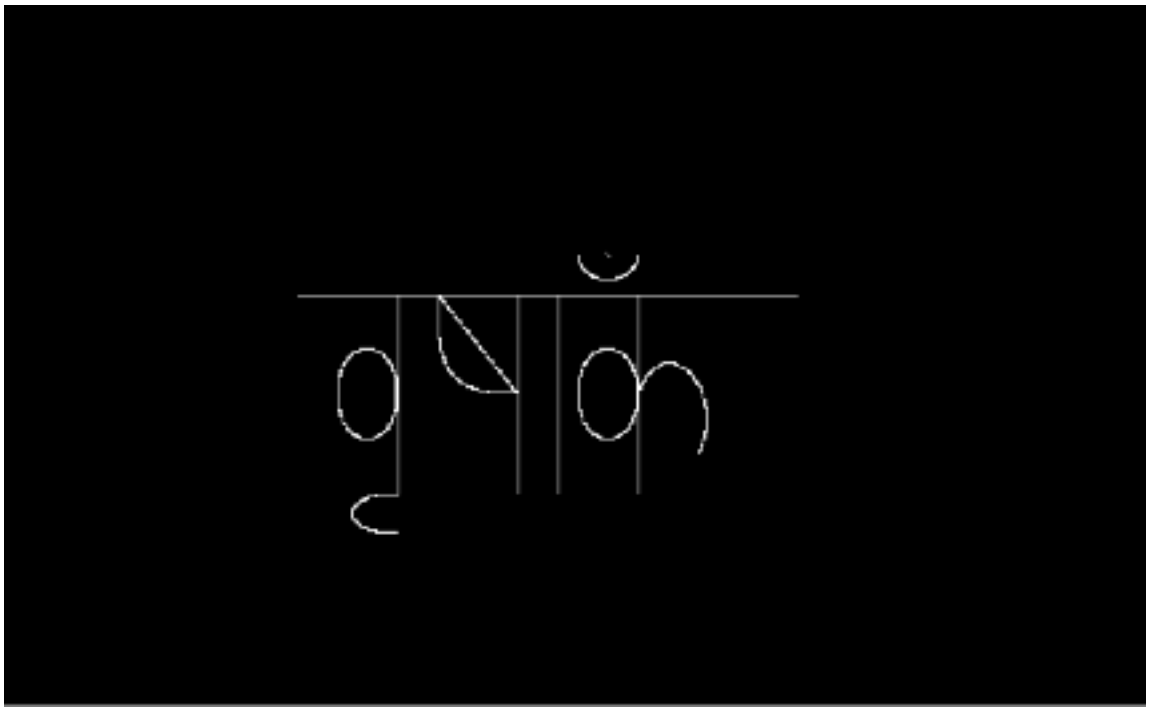


## 16. Drawing our name using hermite curve

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;

void hermite(double x1,double y1,double x2,double y2,double r1,double r2,double r3,double
r4, double cx, double cy, double i)
{
   double x,y,t,X,Y;

   for(t=0.0; t<=1.0; t+=.001)
   {
      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)*r1+(t*t*t-t*t)*r3;
      y=(2*t*t*t-3*t*t+1)*y1+(-2*t*t*t+3*t*t)*y2+(t*t*t-2*t*t+t)*r2+(t*t*t-t*t)*r4;

      x -= cx;
      y -= cy;

      X = x*cos(i) - y*sin(i);
      Y = x*sin(i) + y*cos(i);

      X += cx;
      Y += cy;
      putpixel(getmaxx()/2+X,getmaxy()/2-Y,WHITE);
   }
}


void mult(double a[][3],double b[][3],double res[][3])
{
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
      {
         res[i][j] =0;

         for (int k = 0; k < 3; k++)
         {
            res[i][j] += (a[i][k] * b[k][j]);
            cout << res[i][j] << " ";
         }
      }

}

void mult1by3(double a[][3],double b[][3],double res[][3])
{
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
      {
         res[i][j] =0;

         for (int k = 0; k < 1; k++)
         {
            res[i][j] += (a[i][k] * b[k][j]);
            cout << res[i][j] << " ";
         }
      }

}
```

```
int main()
{
   initwindow(800,800);

   hermite(-250,250,0,250,0,0,0,0,0,0,0);

   hermite(-200,250,-200,150,0,0,0,0,0,0,0);
   hermite(-200,200,-230,200,0,90,0,-90,0,0,0);
   hermite(-230,200,-200,200,0,-90,0,90,0,0,0);
   hermite(-200,150,-200,130,-90,0,90,0,0,0,0);


  hermite(-140,250,-140,150,0,0,0,0,0,0,0);
   hermite(-180,250,-140,200,0,-110,110,0,0,0,0);
   hermite(-180,250,-140,200,0,0,0,0,0,0,0);


   hermite(-120,250,-120,150,0,0,0,0,0,0,0);

   hermite(-110,270,-80,270,0,-50,0,50,0,0,0);
   hermite(-95,270,-96,271,0,0,0,0,0,0,0);

   hermite(-80,250,-80,150,0,0,0,0,0,0,0);
   hermite(-80,200,-110,200,0,90,0,-90,0,0,0);
   hermite(-110,200,-80,200,0,-90,0,90,0,0,0);
   hermite(-80,200,-50,170,40,110,-50,-90,0,0,0);

   getchar();

}return 0;
}
```

OUTPUT:



## 17. Demonstrating the isometric view of a cube

```
 #include<iostream>
#include<bits/stdc++.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include <stdlib.h>
#include <stdio.h>

using namespace std;
float ob[4][8]={
0 ,40,40,0 ,0 ,40,40,0,
0 ,0 ,40,40,0 ,0 ,40,40,
0 ,0 ,0 ,0 ,40,40,40,40,
1,1,1,1,1,1,1,1
};
float t,pi=3.14;
float ob1[4][8],ob2[4][8];

float roty[4][4]=
   {cos(pi/180),0,sin(pi/180),0,
   0,1,0,0,
   -sin(pi/180),0,cos(pi/180),0,
   0,0,0,1
   };
   float rotx[4][4]=
   {1 ,0,0,0,
   0,cos(pi/180),-sin(pi/180),0,
   0,sin(pi/180),cos(pi/180),0,
   0,0,0,1
   };
   float rotz[4][4]=
   {cos(pi/180),-sin(pi/180),0,0,
    sin(pi/180),cos(pi/180),0,0,
    0,0,1,0,
    0,0,0,1};
    float isometric[4]
[4]={0.7071,0,-0.7071,0,-0.40825,0.8165,-0.40825,0,0.7071
,0.40825,0.8165,0,0,0,0,1};

    //matmul(isometric,ob1);
void matmul(float mat1[4][4],float mat2[4][8])
{
   float res[4][8];
   int i,j,k;
   for(i=0;i<4;i++)
   {
      for(j=0;j<8;j++)
      {
         res[i][j]=0;
         for(k=0;k<4;k++)
         {
            res[i][j]+=mat1[i][k]*mat2[k][j];
         }
```

```
        }
    }
    for(int i=0;i<4;i++){
            for(int j=0;j<8;j++)
                mat2[i][j]=res[i][j];
        }
}

void diview(float ob[4][8])
{
     int i=0;
    for(i=0;i<4;i++)
        line(219+ob[0][i%4],300-ob[1][i%4],219+ob[0][(i+1)%4],300-ob[1]
    [(i+1)%4]); for(i=4;i<8;i++)
         line(219+ob[0][i%4+4],300-ob[1][i%4+4],219+ob[0][(i+1)%4+4],300-ob[1][(i+1)%4+4]);

    for(i=0;i<4;i++)
        line(219+ob[0][(i%8)],300-ob[1][i%8],219+ob[0][(i+4)%8],300-ob[1][(i+4)%8]);
}
//float dimetric[4][4];
void rot(float ob[4][8],float ob1[4][8],float ob2[4][8],float dimetric[4][4])
{
    float t;
     float obnew[4][8];
    int i,j,k;
    matmul(isometric,ob1);
    matmul(dimetric,ob2);
    for(t=0;t<=10;t+=0.01)
    {

    matmul(roty,ob);
    matmul(rotx,ob1);
    matmul(rotx,ob2);
        setcolor(WHITE);
        diview(ob2);
        delay(20);
        cleardevice();
    }

}
int main()
{
    int gdrive=DETECT,gmode;
    for(int i=0;i<4;i++)
{
    for(int j=0;j<8;j++)
    {
        ob1[i][j]=ob2[i][j]=ob[i][j];
    }
}
float th1,th2,k;

cout<<"enter k";
cin>>k;
//th1=asin(sqrt(k*k/2)),th2=asin(sqrt(k*k/(k*k+2)));
//th1=acos(sqrt(1/2)),th2=acos(sqrt(2*k*k/(k*k+2)));
//th1=asin(sqrt(1/2)),th2=asin(-sqrt((2-k*k)/(2*(1+k*k))));
```

```
th1=acos(sqrt(k*k/2)),th2=acos(sqrt(2/(k*k+2)));
float dimetric[4][4]={cos(th1),0,-sin(th1),0,-
sin(th1)*sin(th2),cos(th2),-sin(th2)*cos(th1),
0,cos(th2)*sin(th1),sin(th2),cos(th2)*cos(th1),0,0,0,0,1}; //float
x=x1,R=50,y=y1,d;
   initgraph(&gdrive,&gmode,"C    :\\TURBOC3\\BGI");
   /*matmul(dimetric,ob2);
   frontview(ob2);
   diview(ob2);
   matmul(isometric,ob1);
   isoview(ob1);*/
  rot(ob,ob1,ob2,dimetric);
 //isomet(ob1,isometric);
  getche();
    return 0;
}
```

OUTPUT: