"Only Knowledge can provide salvation"

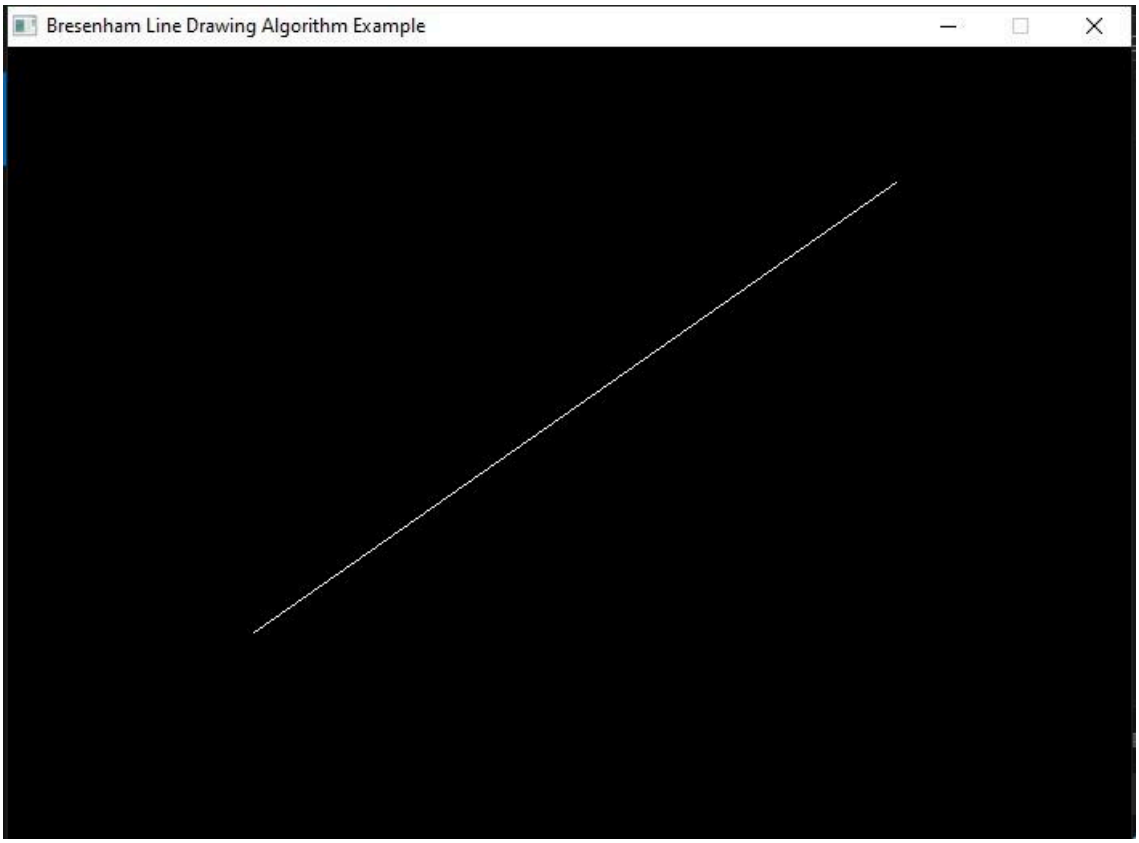
Computer Graphics
Practical File

Name: Ryan Jose
Roll No.: 10872
Course: BSc. Hons Computer Science
Section: B

Q1. Write a program to implement Bresenham's line drawing algorithm.

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

using namespace std;

void bresenhamLine(int x0, int y0, int x1, int y1, int val)
{
 if (x0 == x1 && y0 == y1)
 {
  putpixel(x1, y1, val);
 }
 else
 {
  int dx = x1 - x0;
  int dy = y1 - y0;

  float m = float(dy) / (float)(dx);

  if (m >= 1 || m <= 0)
  {
   cout << "ERROR: Slope must be between 0 and 1." << endl;
   exit(1);
  }

  int d = 2 * dy - dx;
  int del_E = 2 * dy;
  int del_NE = 2 * (dy - dx);

  int x = x0;
  int y = y0;
  putpixel(x, y, val);

  while (x < x1)
  {
   if (d <= 0)
   {
    d += del_E;
    x += 1;
   }
```

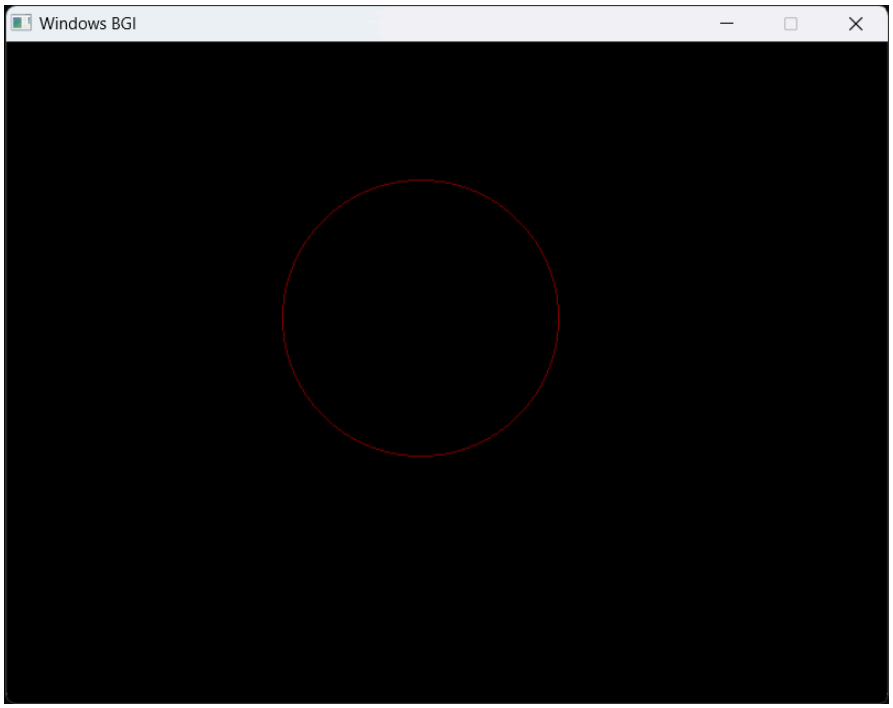
```
      else
      {
        d += del_NE;
        x += 1;
        y += 1;
      }

      putpixel(x, y, val);
    }
  }

  return;
}

int main(void)
{
  int x0, y0, x1, y1;
  cout << "Enter Left Endpoint (x0 y0): ";
  cin >> x0 >> y0;
  cout << "Enter Right Endpoint (x1 y1): ";
  cin >> x1 >> y1;

  cout << "Drawing Line..." << endl;

  int gd = DETECT, gm;
  initgraph(&gd, &gm, NULL);
  bresenhamLine(x0, y0, x1, y1, WHITE);
  delay(5e3);
  closegraph();

  cout << "Finished..." << endl;

  return 0;
}
```



Q2. Write a program to implement mid-point circle drawing algorithm.

```
#include <iostream>
#include <graphics.h>
using namespace std;
int main(){
	int c,r,xc,yc;
	cout<<"Enter the centre coordinates of the circle =  "<<endl;
	cin>>xc>>yc;
	cout<<"Enter radius of the circle =  "<<endl;
	cin>>r;
	int x = 0;
	int y = r;
	int p = 1-r;
	int gd = DETECT, gMode;
	initgraph(&gd,&gMode, NULL);

	do{
		putpixel(x+xc, y+yc,4);
```

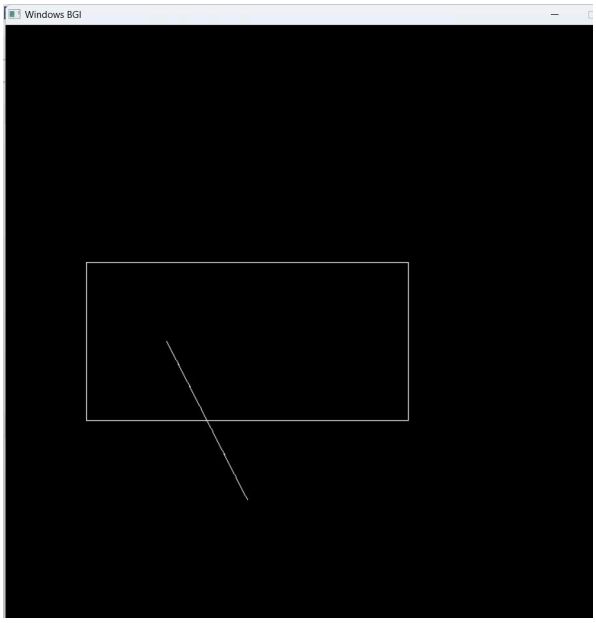
```
            putpixel(xc+x, yc-y,4);
            putpixel(xc-x, yc-y,4);
            putpixel(xc+y, yc+x,4);
            putpixel(xc+y, yc-x,4);
            putpixel(xc-x, yc+y,4);
            putpixel(xc-y, yc+x,4);
            putpixel(xc-y, yc-x,4);

            if(p<0){
                  x =x+1;
                  p = p+2*x+1;
                  putpixel(x+xc, y+yc,4);


            }
            else{
                  x = x+1;
                  y = y-1;
                  p = p+2*x-2*y+1;
                  putpixel(x+xc, y+yc, 4);

            }
      }while(x<=y);
      delay(10000);
      return 0;
}
```

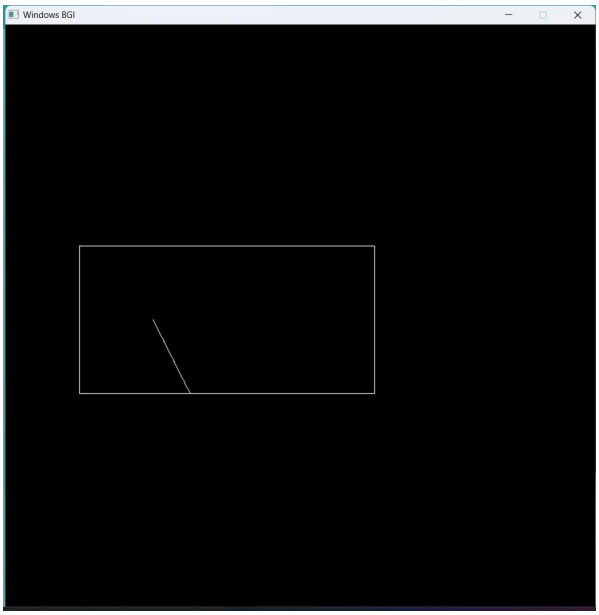


Q3. Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

```
#include <iostream>
#include <graphics.h>
using namespace std;
int xmin = 100, ymin = 300, xmax = 500, ymax = 500;
const int Left = 1;
const int Right = 2;
const int Top = 8;
const int Bottom = 4;

int computecode (int x, int y) {
   int code = 0;
   if (x < xmin) code |= Left;
   else if (y < ymin) code |= Bottom;
   if (x > xmax) code |= Right;
   else if (y > ymax) code |= Top;
   return code;
}

void clip (int x0, int x1, int y0, int y1) {
   int code1, code2;
   int accept, flag = 0;
   code1 = computecode(x0, y0);
   code2 = computecode(x1, y1);
   double m = (y1 - y0) / (x1 - x0);
   if ((code1 & code2) != 0) {
      accept = false;
   } else {
      do {
         if (code1 == 0 && code2 == 0) {
            accept = true;
            flag = 1;
         } else {
            int x, y, temp;
            if (code1 == 0) temp = code2;
            else temp=code1;

            if (temp & Top) {
               x= x0 + (1 / m) * (ymax - y0);
               y = ymax;
            } else if(temp & Bottom) {
```

```
            x = x0 + (1 / m) * (ymin - y0);
            y = ymin;
         } else if(temp & Left){
            y = y0 + m * (xmin - x0);
            x = xmin;
         } else if(temp & Right) {
            y = y0 + m * (xmax - x0);
            x = xmax;
         }
         if (temp == code1) {
            x0 = x;
            y0 = y;
            code1 = computecode(x0, y0);
         } else {
            x1 = x;
            y1 = y;
            code2 = computecode(x1, y1);
         }
      }
   }while(!flag);// do-while end
   }
   if (accept) {
      cleardevice();
      line(x0, y0, x1, y1);
      rectangle(xmin, ymin, xmax, ymax);
   }
}
int main(){
   int window1 = initwindow(800, 800);
   int x0, x1, y0, y1;
   cout << "Enter the co-ordinate of first point: ";
   cin >> x0 >> y0;
   cout << "Enter the co-ordinate of second point: ";
   cin >> x1 >> y1;
   line(x0, y0, x1, y1);
   rectangle(xmin, ymin, xmax, ymax);
   delay(7000);
   clip(x0, x1, y0, y1);
   system("pause");
   return 0;
}
```


Windows BGI


Windows BGI

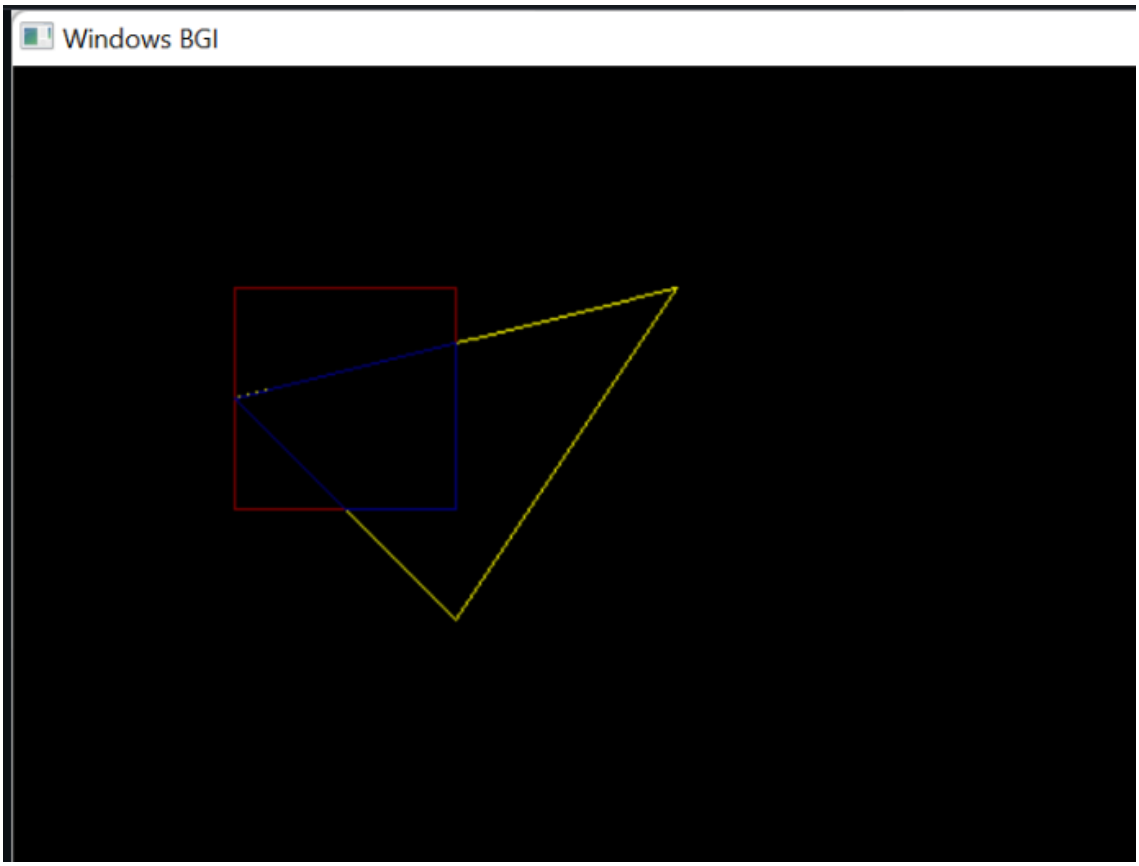
Q4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

```
#include <iostream>
#include <graphics.h>
#define round(a) ((int)(a+0.5))
using namespace std;
int xmin = 100, xmax = 500, ymin = 100, ymax = 500, arr[20], m;
int k;
void clipLeft(int x1, int y1, int x2, int y2) {
	if (x2 - x1) {
		m = (y2 - y1)/(x2 - x1);
	}
	else {
		m = 10000;
	}
	if (x1 >= xmin && x2 >= xmin) {
		arr[k] = x2;
		arr[k+1] = y2;
		k += 2;
	}
	if (x1 < xmin && x2 >= xmin) {
		arr[k] = xmin;
		arr[k+1] = y1 + m*(xmin - x1);
		arr[k+2] = x2;
		arr[k+3] = y2;
		k+=4;
	}
	if (x1 >= xmin && x2 < xmin) {
		arr[k] = xmin;
		arr[k+1] = y1 + m*(xmin - x1);
		k += 2;
	}
}

void clipTop(int x1, int y1, int x2, int y2) {
	if (y2 - y1) {
		m = (x2 - x1)/(y2 - y1);
	}
	else {
		m = 10000;
	}
	if (y1<=ymax && y2 <= ymax) {
		arr[k] = x2;
```

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

void clipRight(int x1, int y1, int x2, int y2){
        if(x2-x1){
                m = (y2-y1)/(x2 -x1);

        }
        else{
                 m = 10000;
        }
        if(x1<=xmax && x2<= xmax){
                arr[k] = x2;
                arr[k+1]= y2;
                k +=2;
        }

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

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

```
}
void clipBottom(int x1, int y1, int x2, int y2){
	if(y2-y1){
		m = (x2-x1)/(y2-y1);
	}
	else{
		m = 10000;
	}
	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 >= ymax && y2 < ymin) {
		arr[k] = x1 + m * (ymin - y1);
		arr[k+1] = ymin;
		k += 2;
	}
}
int main() {
	int poly[20];
	int window1 = initwindow(800, 800);
	int n, i;
	cout << "Enter the number of edges: " << endl;
	cin >> n;
	cout << "Enter the coordinates: " << endl;
	for (i = 0; i < 2 * n; i++)
	cin>>poly[i];
	poly[i] = poly[0];
	poly[i+1] = poly[1];
	rectangle(xmin, ymax, xmax, ymin);
	fillpoly(n , poly);
	delay(1000);
	cleardevice();
	k = 0;
```

```
        for(i =0; i<2*n; i +=2)
        clipLeft(poly[i], poly[i+1], poly[i+2], poly[i+3]);

        n = k/2;
        for(i = 0; i <k; i++)
        poly[i]= arr[i];
        poly[i]= poly[0];
        poly[i+1]= poly[1];

        k = 0;
        for(int i =0; i<2*n; i +=2)
        clipRight(poly[i], poly[i+1], poly[i+2], poly[i+3]);

        n = k/2;
        for(int i = 0; i <k; i++)
        poly[i]= arr[i];
        poly[i]= poly[0];
        poly[i+1]= poly[1];

        k = 0;
        for(int i =0; i<2*n; i +=2)
        clipBottom(poly[i], poly[i+1], poly[i+2], poly[i+3]);

        for(int i = 0; i <k; i++)
        poly[i]= arr[i];
        rectangle(xmin, ymax, xmax, ymin);


        if(k)
        fillpoly(k/2,poly);

        system("pause");
        return 0;
}
```

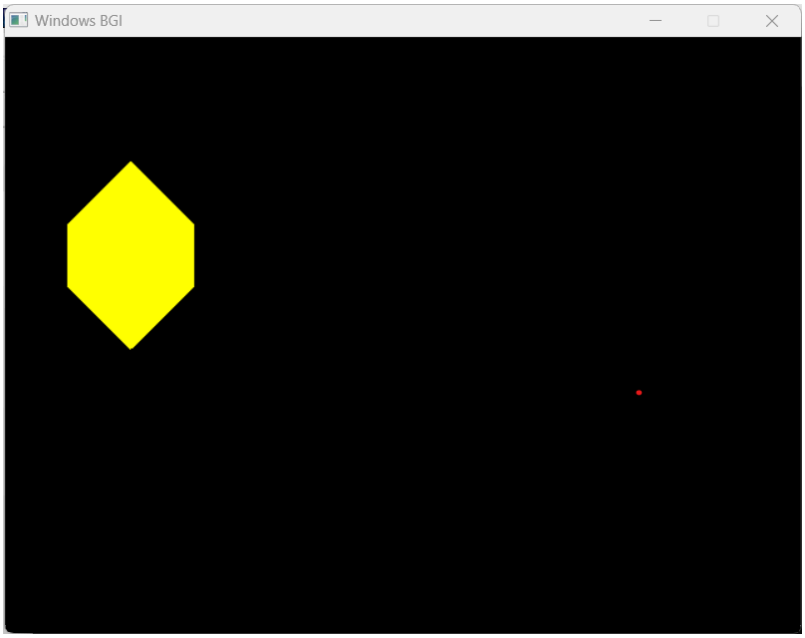


Q5. Write a program to fill a polygon using the Scan line fill algorithm.

```
#include <graphics.h>
#include <iostream>
using namespace std;
int main()
{
   int n, i, j, k, gd, gm, dy, dx;
   int x, y, temp;
   int a[20][2], xi[20];
   float slope[20];
   int temp1 = 0;
   cout << "\nEnter the number of edges ";
   cin >> n;
   for (i = 0; i < n; i++)
   {
      cout << "Enter the coordinate x" << i + 1 << " ";
      cin >> a[i][0];
      cout << "Enter the coordinate y" << i + 1 << " ";
      cin >> a[i][1];
   }
   a[n][0] = a[0][0];
```

```
a[n][1] = a[0][1];
initgraph(&gd, &gm, NULL);
setcolor(YELLOW);
for (i = 0; i < n; i++)
{
   line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);
}
for (i = 0; i < n; i++)
{
   dy = a[i + 1][1] - a[i][1];
   dx = a[i + 1][0] - a[i][0];
   if (dy == 0)
      slope[i] = 1.0;
   if (dx == 0)
      slope[i] = 0.0;
   if ((dy != 0) && (dx != 0))
   {
      slope[i] = (float)dx / dy;
   }
}
for (y = 0; y < 400; y++)
{
   k = 0;
   for (i = 0; i < n; i++)
   {
      if (((a[i][1] <= y) && (a[i + 1][1] > y)) || ((a[i][1] > y) && (a[i + 1][1] <= y)))
      {
         xi[k] = (int)(a[i][0] + slope[i] * (y - a[i][1]));
         k++;
      }
   }
   for (j = 0; j < k; j++)
      for (i = 0; i < k; i++)
      {
         if (xi[i] > xi[i + 1])
         {
            temp = xi[i];
            xi[i] = xi[i + 1];
            xi[i + 1] = temp;
         }
      }
   setcolor(YELLOW);
   for (i = 0; i < k; i += 2)
   {
```

```
            line(xi[i], y, xi[i + 1] + 1, y);
            temp1 = i;
        }
    }
    delay(7000);
    return 0;
}
```

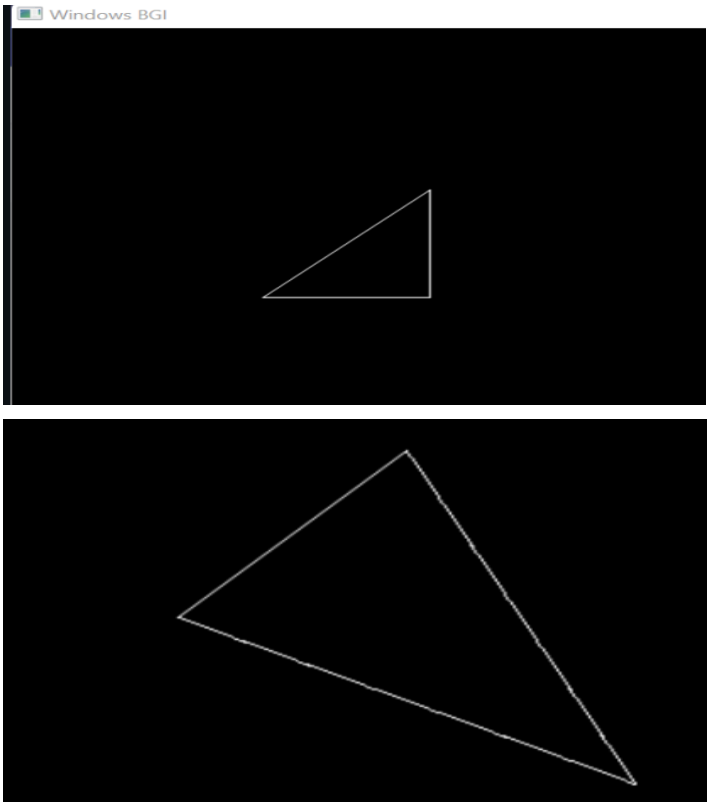


Q6. Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates).

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

int main()
{
    int gm;
    int gd=DETECT;
    int x1,x2,x3,y1,y2,y3,nx1,nx2,nx3,ny1,ny2,ny3,c;
    int sx,sy,xt,yt,r;
    float t;
    initgraph(&gd,&gm,"c:\\tc\\bg:");
    cout<<"\t Program for basic transactions :";
    cout<<"\n\t Enter the points of triangle :\n";
    cin>>x1>>y1>>x2>>y2>>x3>>y3;
    line(x1,y1,x2,y2);
```

```
   line(x2,y2,x3,y3);
   line(x3,y3,x1,y1);
   cout<<("\n 1. Translation\n 2. Rotation\n 3. Scaling\n 4. Reflection\n 5. Shearing\n 6.
Exit");
   cout<<"\nEnter your choice:";
   cin>>c;
   switch(c)
   {
      case 1:
         cout<<"\n Enter the translation factor: ";
         cin>>xt>>yt;
         nx1=x1+xt;
         ny1=y1+yt;
         nx2=x2+xt;
         ny2=y2+yt;
         nx3=x3+xt;
         ny3=y3+yt;
         line(nx1,ny1,nx2,ny2);
         line(nx2,ny2,nx3,ny3);
         line(nx3,ny3,nx1,ny1);
         break;

      case 2:
         cout<<"\n Enter the angle of rotation: ";
         cin>>r;
         t=(3.14*r)/180;
         nx1=x1*cos(t)-y1*sin(t);
         ny1=x1*sin(t)+y1*cos(t);
         nx2=x2*cos(t)-y2*sin(t);
         ny2=x2*sin(t)+y2*cos(t);
         nx3=x3*cos(t)-y3*sin(t);
         ny3=x3*sin(t)+y3*cos(t);
         line(nx1,ny1,nx2,ny2);
         line(nx2,ny2,nx3,ny3);
         line(nx3,ny3,nx1,ny1);
         break;

      case 3:
         cout<<"\n Enter the scaling factor: ";
         cin>>sx>>sy;
         nx1=x1*sx;
         ny1=y1*sy;
         nx2=x2*sx;
         ny2=y2*sy;
```

```
            nx3=x3*sx;
            ny3=y3*sy;
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            break;

        case 4:
            cout<<"\n Reflection about X-axis";
            nx1 = x1;
            ny1 = -y1;
            nx2 = x2;
            ny2 = -y2;
            nx3 = x3;
            ny3 = -y3;
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            break;

        case 5:
            cout<<"\n Shearing along X-axis: ";
            cout<<"\n Enter the shearing factor: ";
            cin>>sx;
            nx1 = x1 + sx * y1;
            ny1 = y1;
            nx2 = x2 + sx * y2;
            ny2 = y2;
            nx3 = x3 + sx * y3;
            ny3 = y3;
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            break;

        case 6:
            break;

        default:
            cout<<"Enter the correct choice";
    }
    getch();
    closegraph();
}
```



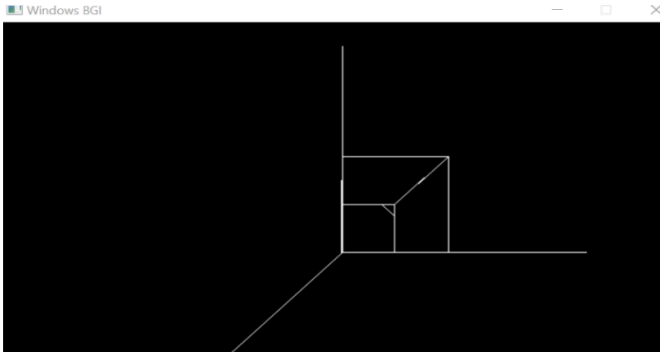
Q7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

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

int main()
{
   int gm;
   int gd=DETECT;
   int x1,x2,x3,y1,y2,y3,z1,z2,z3,nx1,nx2,nx3,ny1,ny2,ny3,nz1,nz2,nz3,c;
   int tx, ty, tz, sx, sy, sz, rx, ry, rz;
   float theta_x, theta_y, theta_z, shear_xy, shear_xz, shear_yx, shear_yz, shear_zx, shear_zy;

   initgraph(&gd,&gm,"c:\\tc\\bg:");
   cout<<"\t Program for basic 3D transformations :";
   cout<<"\n\t Enter the points of triangle (x, y, z):\n";
   cin>>x1>>y1>>z1>>x2>>y2>>z2>>x3>>y3>>z3;
```

```
   line(x1,y1,x2,y2);
   line(x2,y2,x3,y3);
   line(x3,y3,x1,y1);
   cout<<("\n 1. Translation\n 2. Rotation\n 3. Scaling\n 4. Reflection\n 5. Shearing\n 6.
Exit");
   cout<<"\nEnter your choice:";
   cin>>c;
   switch(c)
   {
      case 1:
         cout<<"\n Enter the translation factors (tx, ty, tz): ";
         cin>>tx>>ty>>tz;
         nx1=x1+tx;
         ny1=y1+ty;
         nz1=z1+tz;
         nx2=x2+tx;
         ny2=y2+ty;
         nz2=z2+tz;
         nx3=x3+tx;
         ny3=y3+ty;
         nz3=z3+tz;
         line(nx1,ny1,nx2,ny2);
         line(nx2,ny2,nx3,ny3);
         line(nx3,ny3,nx1,ny1);
         break;

      case 2:
         cout<<"\n Enter the rotation angles (theta_x, theta_y, theta_z): ";
         cin>>theta_x>>theta_y>>theta_z;
         theta_x = (theta_x * 3.14) / 180;
         theta_y = (theta_y * 3.14) / 180;
         theta_z = (theta_z * 3.14) / 180;

         nx1 = x1*cos(theta_y)*cos(theta_z) +
x2*(sin(theta_x)*sin(theta_y)*cos(theta_z)-cos(theta_x)*sin(theta_z)) +
x3*(cos(theta_x)*sin(theta_y)*cos(theta_z)+sin(theta_x)*sin(theta_z));
         ny1 = y1*cos(theta_y)*cos(theta_z) +
y2*(sin(theta_x)*sin(theta_y)*cos(theta_z)-cos(theta_x)*sin(theta_z)) +
y3*(cos(theta_x)*sin(theta_y)*cos(theta_z)+sin(theta_x)*sin(theta_z));
         nz1 = z1*cos(theta_y)*cos(theta_z) +
z2*(sin(theta_x)*sin(theta_y)*cos(theta_z)-cos(theta_x)*sin(theta_z)) +
z3*(cos(theta_x)*sin(theta_y)*cos(theta_z)+sin(theta_x)*sin(theta_z));
```

```
        nx2 = x1*cos(theta_y)*sin(theta_z) +
x2*(sin(theta_x)*sin(theta_y)*sin(theta_z)+cos(theta_x)*cos(theta_z)) +
x3*(cos(theta_x)*sin(theta_y)*sin(theta_z)-sin(theta_x)*cos(theta_z));
        ny2 = y1*cos(theta_y)*sin(theta_z) +
y2*(sin(theta_x)*sin(theta_y)*sin(theta_z)+cos(theta_x)*cos(theta_z)) +
y3*(cos(theta_x)*sin(theta_y)*sin(theta_z)-sin(theta_x)*cos(theta_z));
        nz2 = z1*cos(theta_y)*sin(theta_z) +
z2*(sin(theta_x)*sin(theta_y)*sin(theta_z)+cos(theta_x)*cos(theta_z)) +
z3*(cos(theta_x)*sin(theta_y)*sin(theta_z)-sin(theta_x)*cos(theta_z));

        nx3 = x1*-sin(theta_y) + x2*sin(theta_x)*cos(theta_y) +
x3*cos(theta_x)*cos(theta_y);
        ny3 = y1*-sin(theta_y) + y2*sin(theta_x)*cos(theta_y) +
y3*cos(theta_x)*cos(theta_y);
        nz3 = z1*-sin(theta_y) + z2*sin(theta_x)*cos(theta_y) +
z3*cos(theta_x)*cos(theta_y);

        line(nx1,ny1,nx2,ny2);
        line(nx2,ny2,nx3,ny3);
        line(nx3,ny3,nx1,ny1);
        break;

    case 3:
        cout<<"\n Enter the scaling factors (sx, sy, sz): ";
        cin>>sx>>sy>>sz;
        nx1=x1*sx;
        ny1=y1*sy;
        nz1=z1*sz;
        nx2=x2*sx;
        ny2=y2*sy;
        nz2=z2*sz;
        nx3=x3*sx;
        ny3=y3*sy;
        nz3=z3*sz;
        line(nx1,ny1,nx2,ny2);
        line(nx2,ny2,nx3,ny3);
        line(nx3,ny3,nx1,ny1);
        break;

    case 4:
        cout<<"\n Reflection about XY-plane";
        nx1 = x1;
        ny1 = y1;
        nz1 = -z1;
```

```
            nx2 = x2;
            ny2 = y2;
            nz2 = -z2;
            nx3 = x3;
            ny3 = y3;
            nz3 = -z3;
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            break;

        case 5:
            cout<<"\n Shearing along X-axis: ";
            cout<<"\n Enter the shearing factors (shear_xy, shear_xz): ";
            cin>>shear_xy>>shear_xz;
            nx1 = x1 + shear_xy * y1 + shear_xz * z1;
            ny1 = y1;
            nz1 = z1;
            nx2 = x2 + shear_xy * y2 + shear_xz * z2;
            ny2 = y2;
            nz2 = z2;
            nx3 = x3 + shear_xy * y3 + shear_xz * z3;
            ny3 = y3;
            nz3 = z3;
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            break;

        case 6:
            break;

        default:
            cout<<"Enter the correct choice";
    }
    getch();
    closegraph();
}
```

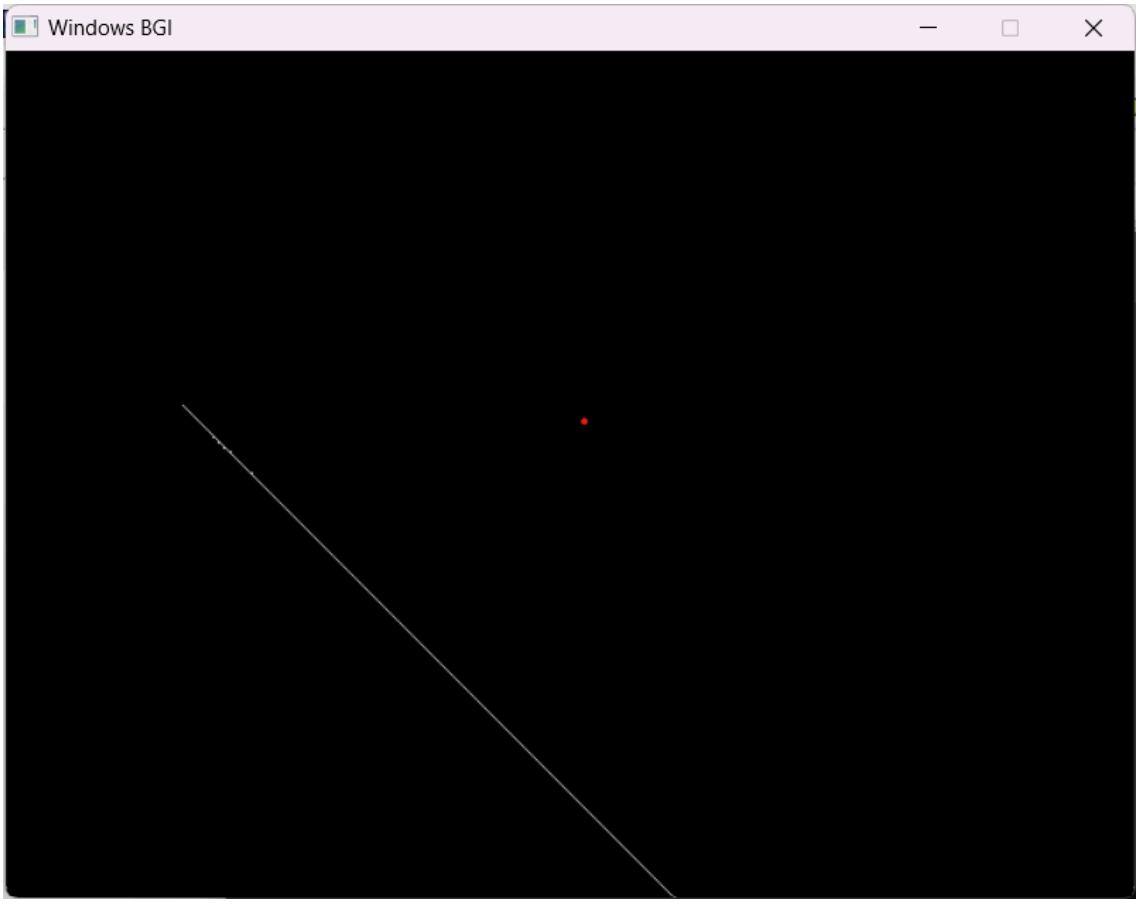


Q8. Write a program to draw the Hermite/Bezier curve.

Bezier Curve

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

int main(){
	int x[4], y[4], i, gd = DETECT, gm;
	float u, px, py;
	initgraph(&gd, &gm, "");
	for (i = 0; i < 4; i++) {
		cout << "Enter x and y coordinates: ";
		cin >> x[i] >> y[i];
		putpixel(x[i], y[i], 4);
	}
	for (u =0; u <= 1.0; u = u + 0.001) {
```

```
            px = pow(1 - u, 3) * x[0] + 3*u*pow(1-u, 2) * x[1] + 3* pow(u, 2) * (1 - u) *
                    x[2] + pow(u, 3) *   x[3];
            py = pow(1 - u, 3) * y[0] + 3*u*pow(1-u, 2) * y[1] + 3* pow(u, 2) * (1 - u) *
                     y[2] + pow(u, 3) * y[3];
            putpixel(px, py, 7);
        }
        getch();
        return 0;
}
```



Hermite Curve

```
#include <iostream>
#include <graphics.h>
#include <cmath>
using namespace std;
int main() {
        int x[4], y[4], i, gd = DETECT, gm;
        float t, px, py;
        initgraph(&gd, &gm, "");
```

```
        for (i = 0; i < 4; i++) {
                cout << "Enter x and y coordinates: ";
                cin >> x[i] >> y[i];
                putpixel(x[i], y[i], 4);
        }
        for (t =0; t <= 1.0; t = t + 0.001) {
                px = (2*pow(t,3)-3*pow(t,2) +1)*x[0] + (-2*pow(t,3)+3*pow(t,2))*x[1] +
                (pow(t,3) -2*pow(t,2) + t)*x[2]+(pow(t,3)-pow(t,2))*x[3];
                py = (2*pow(t,3)-3*pow(t,2) +1)*y[0] + (-2*pow(t,3)+3*pow(t,2))*y[1] +
                        (pow(t,3)-2*pow(t,2) + t)*y[2]+(pow(t,3)-pow(t,2))*y[3];
                putpixel(px, py, 7);
        }
        getch();
        return 0;

}
```