Temă: metoda transformărilor iterate

1) Incercaţi să desenaţi, utilizând o clasă derivată din `ComplexForm`, următoarea curbă a lui Peano, a cărei construcţie poate fi dedusă din primele două etape indicate în figurile 1 şi 2.

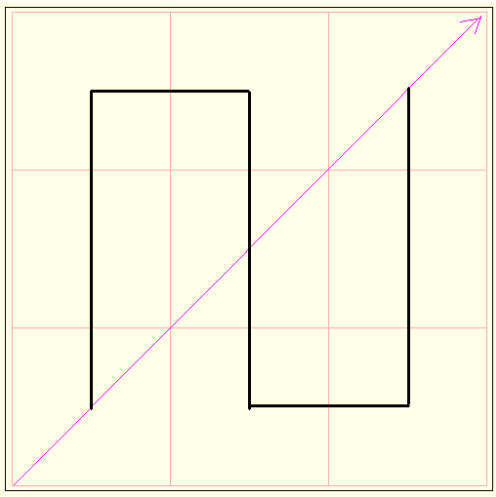


Figura 1.

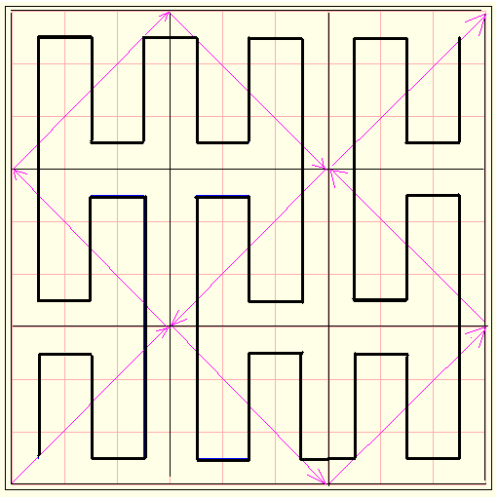


Figura 2.

Indicaţie: clasa următoare trasează desenul din Figura 3:

```
public class Peano_in_lucru : ComplexForm
 {
     static Complex i = new Complex(0.0, 1.0);

     static Complex c1 = (1 + 1 * i) / 6;
     static Complex c2 = (1 + 3 * i) / 6;
     static Complex c3 = (1 + 5 * i) / 6;
     static Complex c4 = (3 + 5 * i) / 6;
     static Complex c0 = (3 + 3 * i) / 6;
     static Complex c5 = (3 + 1 * i) / 6;
     static Complex c6 = (5 + 1 * i) / 6;
     static Complex c7 = (5 + 3 * i) / 6;
     static Complex c8 = (5 + 5 * i) / 6;

     Complex s1(Complex z)
     {
         return c1 + (z - c0) / 3;
     }
     Complex s2(Complex z)
     {
         return c2 + (z - c0) / 3;
     }
     Complex s3(Complex z)
     {
         return c3 + (z - c0) / 3;
     }
     Complex s4(Complex z)
     {
         return c4 + (z - c0) / 3; ;
     }
     Complex s0(Complex z)
     {
         return c0 + (z - c0) / 3;
     }
     Complex s5(Complex z)
     {
         return c5 + (z - c0) / 3; ;
     }
     Complex s6(Complex z)
     {
         return c6 + (z - c0) / 3;
     }

     Complex s7(Complex z)
     {
         return c7 + (z - c0) / 3; ;
     }

     Complex s8(Complex z)
     {
         return c8 + (z - c0) / 3;
     }
```

```
    void transforma(ref List<Complex> li)
    {
        List<Complex> rez = new List<Complex>();
        foreach (Complex z in li) rez.Add(s1(z));
        foreach (Complex z in li) rez.Add(s2(z));
        foreach (Complex z in li) rez.Add(s3(z));
        foreach (Complex z in li) rez.Add(s4(z));
        foreach (Complex z in li) rez.Add(s0(z));
        foreach (Complex z in li) rez.Add(s5(z));
        foreach (Complex z in li) rez.Add(s6(z));
        foreach (Complex z in li) rez.Add(s7(z));
        foreach (Complex z in li) rez.Add(s8(z));
        li = rez;
    }

    void traseaza(List<Complex> li)
    {
        initScreen();
        //trasam chenarul
        setLine(0.0, 1.0, PenColor);
        setLine(1.0, 1 + i, PenColor);
        setLine(1 + i, i, PenColor);
        setLine(i, 0.0, PenColor);
        //desenam curba curenta

        for (int n = 1; n < li.Count; n++)
        {
            Color col = Color.Blue;
            if (n % 9 == 0) col = Color.Red;
            setLine(li[n - 1], li[n], col);
        }
    }

    public override void makeImage()
    {
        setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1);
        ScreenColor = Color.White;
        PenColor = Color.Navy;
        List<Complex> fig = new List<Complex>() { c0 };
        for (int k = 0; k < 2; k++)
        {
            transforma(ref fig);
            traseaza(fig);
            if (!resetScreen()) return;
        }
    }
}
```

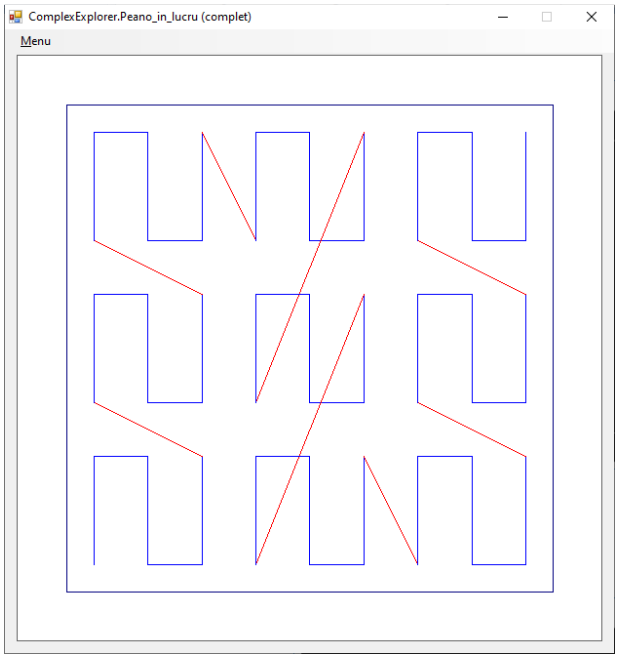


Figura 3

Mai trebuie doar corectate unele dintre transformările **`s0, s1, … ,s8`** astfel încât rezultatul să arate ca în Figura 4:

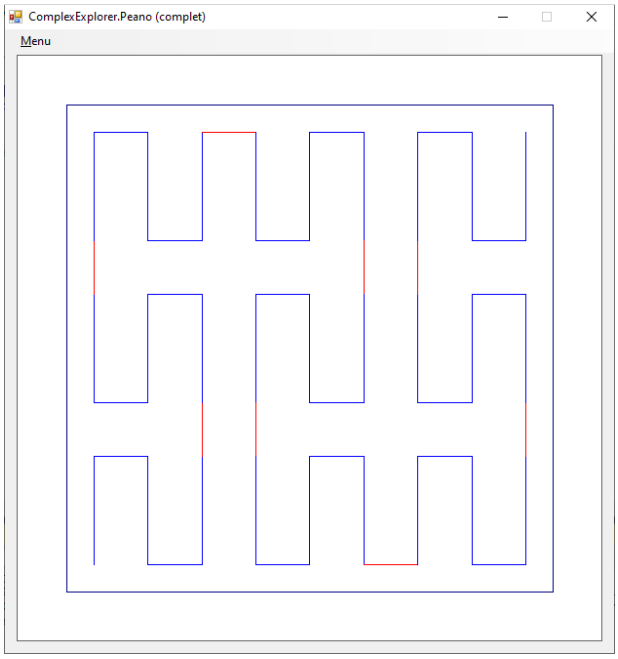
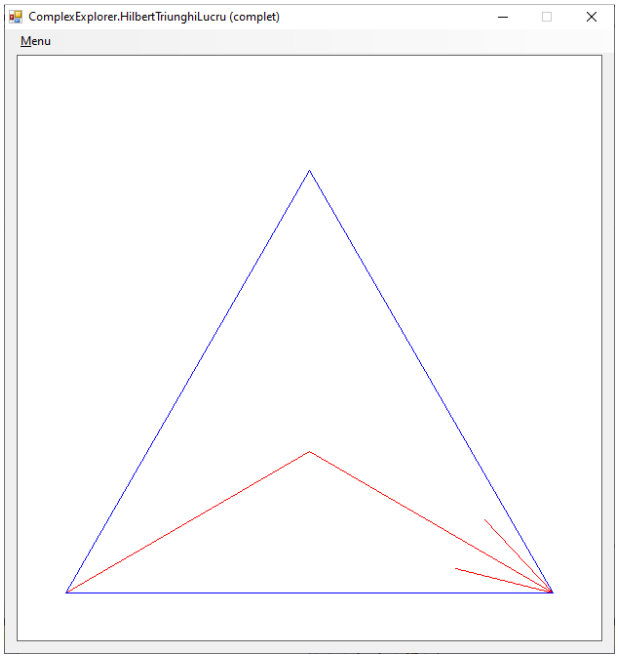


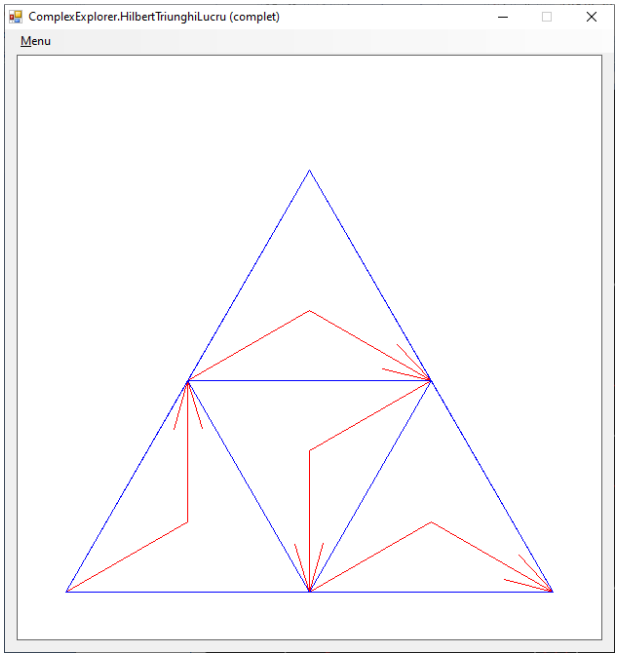
Figura 4.

2) Incercaţi să umpleţi un triunghi parcurgându-l în modul sugerat de următoarele primele trei etape:

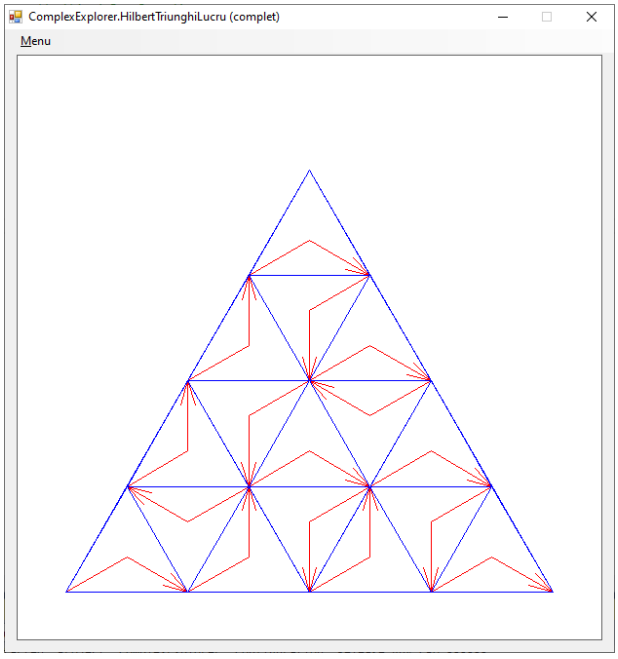
Etapa 1:



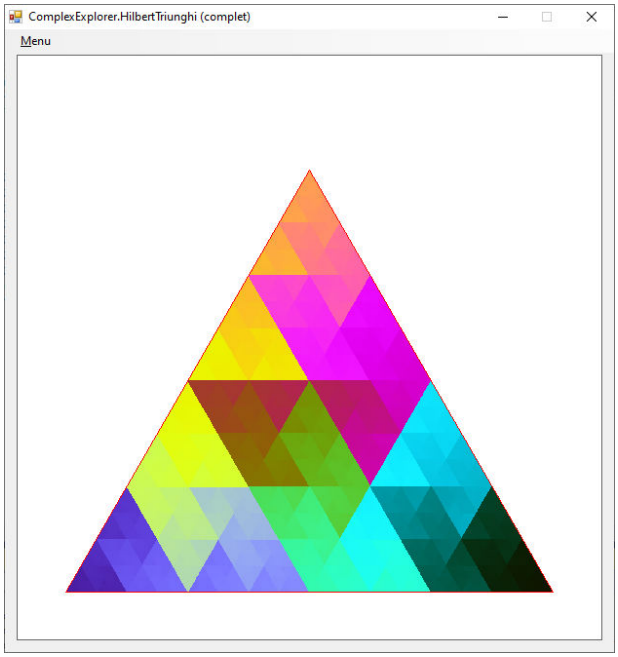
Etapa 2:



Etapa 3:



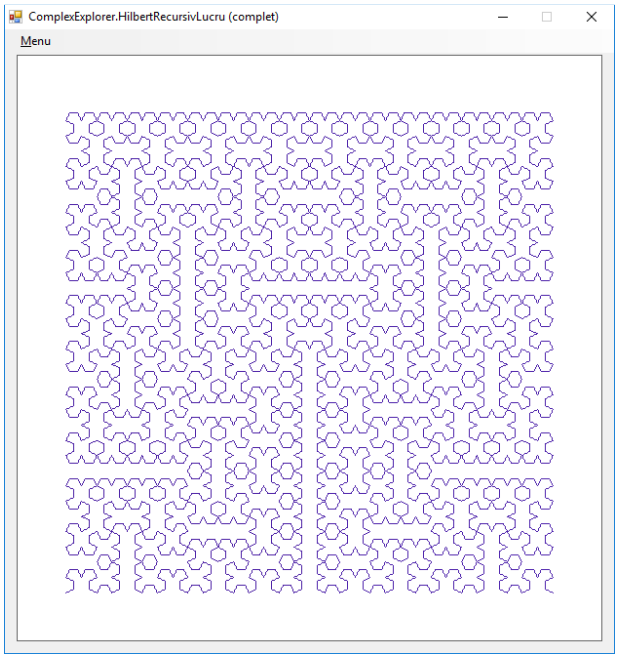
Rezultat, după ce s-a schimbat modul de colorare:



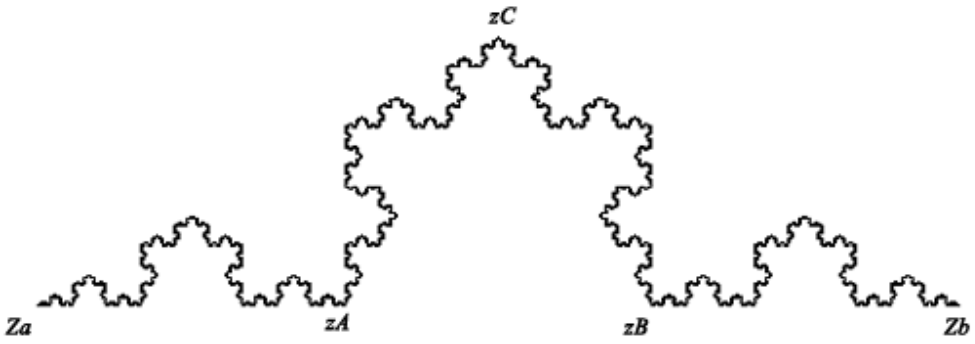
3) Următorul program trasează *Curba lui Hilbert* printr-o implementare cu funcţii recursive:

```
public class HilbertRecursivListe : ComplexForm
    {
        static Complex i = new Complex(0, 1);
        static List<Complex> lista = new List<Complex>();
        // traseu prin patrat:
        // z0 = coltul din stanga jos
        // z0+d1 = stanga sus
        // z0+d1+d2 = dreapta sus
        // z0+d2 = dreapta jos
        void genereaza(Complex z0, Complex d1, Complex d2, int niv) //Hilbert
        {
            d1 *= 0.5;
            d2 *= 0.5;
            if (--niv < 0)
            {
                lista.Add(z0 + d1 + d2);
            }
            else
            {
                genereaza(z0, d2, d1, niv);
                genereaza(z0 + d1, d1, d2, niv);
                genereaza(z0 + d1 + d2, d1, d2, niv);
                genereaza(z0 + d1 + d2 + d2, -d2, -d1, niv);
            }
        }
        void traseaza()
        {

            initScreen();
            for (int n = 1; n < lista.Count; n++)
            {

                setLine(lista[n - 1], lista[n], PenColor);
            }
            resetScreen();
        }
        public override void makeImage()
        {
            setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1);
            ScreenColor = Color.WhiteSmoke;
            PenColor = Color.Navy;
            Complex z0 = 0;
            Complex delta1 = i;
            Complex delta2 = 1;
            int nrEtape = 5;
            genereaza(z0, delta1, delta2, nrEtape);
            traseaza();
        }
    }
```

Modificaţi codul executat de metoda `genereaza()` la ieşirea din recursivitate (pe ramura *true* a if-ului), astfel încât după 5 etape să apară următorul rezultat:



4) Trasaţi curba lui Koch prin metoda transformărilor geometrice iterate, folosind asemănarea dintre curba construită pe segmentul de bază $z_a z_b$ şi cele 4 curbe Koch construite pe segmentele $z_a z_A$, $z_A z_C$, $z_C z_B$ şi $z_B z_b$ ale motivului.

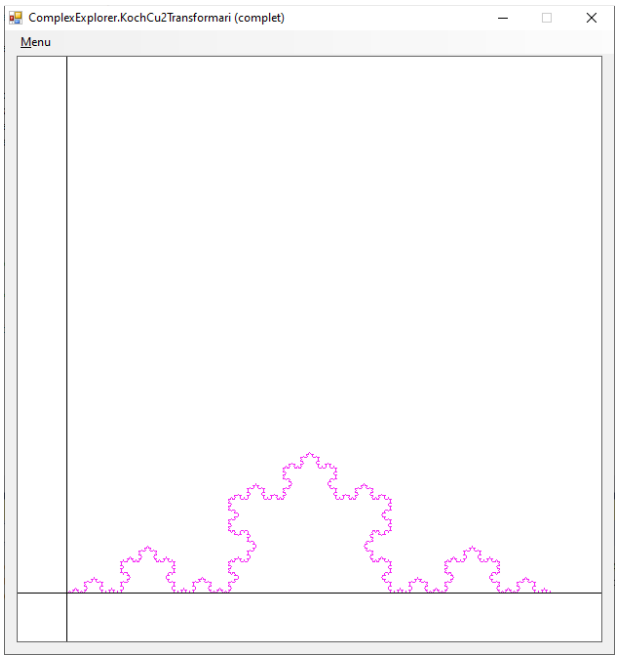


5) Următoarea clasă trasează curba lui Koch cu două transformărilor geometrice iterate, folosind asemănarea dintre curba construită pe stânga segmentului de bază $z_a z_b$ şi cele două curbe Koch construite pe dreapta segmentelor $z_a z_C$ şi $z_C z_b$. Justificaţi forma transformărilor folosite.

Indicaţie: triunghiul Δabc este *invers-asemenea* cu Δpqr dacă triunghiul Δabc este *direct-asemenea* cu triunghiul format de conjugatele numerelor complexe p, q şi r, în această ordine.

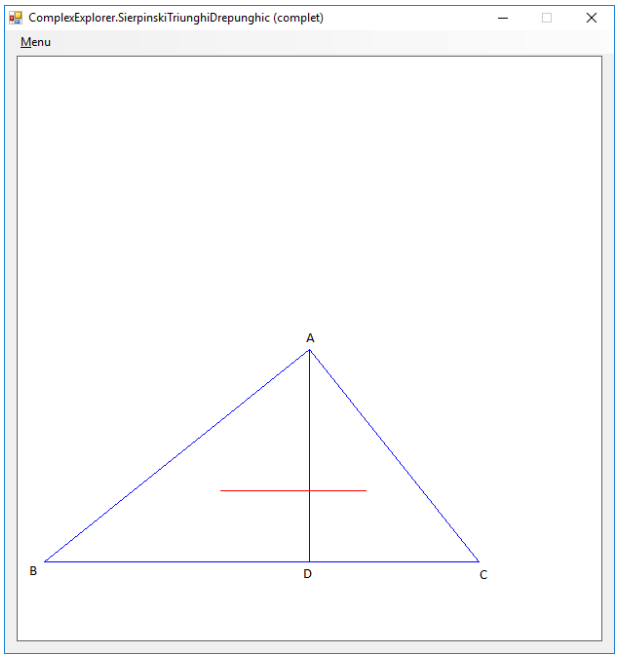
```
public class KochCu2Transformari : ComplexForm
{
    static Complex i = new Complex(0, 1);

    static double theta = Math.PI / 6;
    static double ro = 0.5 / Math.Cos(theta);
    static Complex w = Complex.setRoTheta(ro, theta);

    static Complex zA = 0, zB = 1;
    static Complex zC = zA + w * (zB - zA);

    static Complex omega1 = (zC - zA) / (zB - zA).conj;
    static Complex omega2 = (zC - zB) / (zA - zB).conj;


    Complex T1(Complex z)
    {
        return zA + omega1 * (z - zA).conj;
    }

    Complex T2(Complex z)
    {
        return zB + omega2 * (z - zB).conj;
    }

    void transforma(ref List<Complex> li)
    {
        List<Complex> rez = new List<Complex>();
        foreach (Complex z in li) rez.Add(T1(z));
        foreach (Complex z in li) rez.Add(T2(z));
        li = rez;
    }

    void traseaza(List<Complex> li, int etapa)
    {
        initScreen();
        //trasam chenarul
        Color col = getColor(300 + 10 * etapa);

        for (int n = 1; n < li.Count; n++)
        {

            setLine(li[n - 1], li[n], col);
        }
        setAxis();
    }
```

```
    public override void makeImage()
    {
        setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1);
        ScreenColor = Color.White;
        List<Complex> fig = new List<Complex>() { zA, zB };

        int nrEtape = 10;
        for (int k = 0; k < nrEtape; k++)
        {
            transforma(ref fig);
            traseaza(fig, k);
            if (!resetScreen()) return;
            delaySec(0.5);
        }
    }
}
```



6) Orice triunghi dreptunghic $\Delta ABC(A = 90°)$ este împărţit de înălţimea AD dusă din unghiul drept în două triunghiuri asemenea cu triunghiul iniţial.

Pornim cu un astfel de triunghi şi la fiecare etapă împărţim triunghiurile obţinute în două alte triunghiuri prin ducerea înălţimii. Linia poligonală formată de centrele de greutate ale triunghiurilor obţinute la etapa *n* reprezintă aproximarea de ordin *n* a *Curbei lui Sierpinski* într-un triunghi dreptunghic oarecare (cazul clasic este dat de triunghiul dreptunghic isoscel).



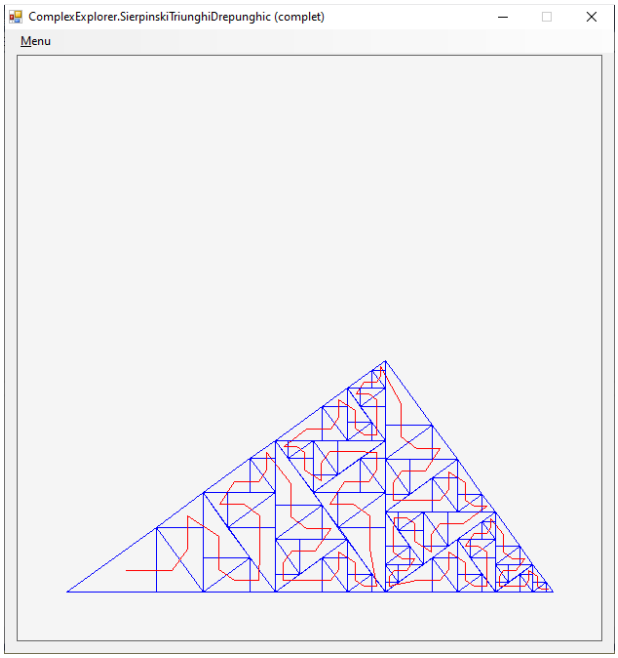
Clasa următoare trasează prin metoda transformărilor geometrice această curbă:

```
public class SierpinskiTriunghiDrepunghic : ComplexForm
  {
      static Complex i = new Complex(0.0, 1.0);
      static Complex zB = 0, zC = 1;
      static Complex zQ = (zB + zC) / 2;
      static Complex zA = zQ + Complex.setRoTheta((zC - zQ).Ro, 2 * Math.PI / 5);

      static Complex k1 = (zA - zC) / (zB - zC).conj;
      static Complex k2 = (zA - zB) / (zC - zB).conj;

      Complex s1(Complex z)
      {
          return zC + k1 * (z - zC).conj;
      }
      Complex s2(Complex z)
      {
          return zB + k2 * (z - zB).conj;
      }
```

```
    void transforma(ref List<Complex> li)
    {
        List<Complex> rez = new List<Complex>();
        foreach (Complex z in li) rez.Add(s1(z));
        foreach (Complex z in li) rez.Add(s2(z));
        li = rez;
    }



    void traseaza(List<Complex> li)
    {
        initScreen();
        Complex z1, z2, z3, zA, zB;

        z1 = li[0];
        z2 = li[1];
        z3 = li[2];
        zA = (z1 + z2 + z3) / 3;
        setLine(z1, z2, Color.Blue);
        setLine(z2, z3, Color.Blue);
        setLine(z3, z1, Color.Blue);
        if (li.Count == 3) return;

        for (int n = 5; n < li.Count; n += 3)
        {
            z1 = li[n - 2];
            z2 = li[n - 1];
            z3 = li[n];
            zB = (z1 + z2 + z3) / 3;
            setLine(z1, z2, Color.Blue);
            setLine(z2, z3, Color.Blue);
            setLine(z3, z1, Color.Blue);
            setLine(zA, zB, Color.Red);
            zA = zB;
        }
    }
    public override void makeImage()
    {
        setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1);
        ScreenColor = Color.WhiteSmoke;
        PenColor = Color.Navy;
        List<Complex> fig = new List<Complex>() { zB, zA, zC };

        //traseaza(fig);
        for (int k = 0; k < 7; k++)
        {
            transforma(ref fig);
            traseaza(fig);
            if (!resetScreen()) return;
        }
    }
}
```

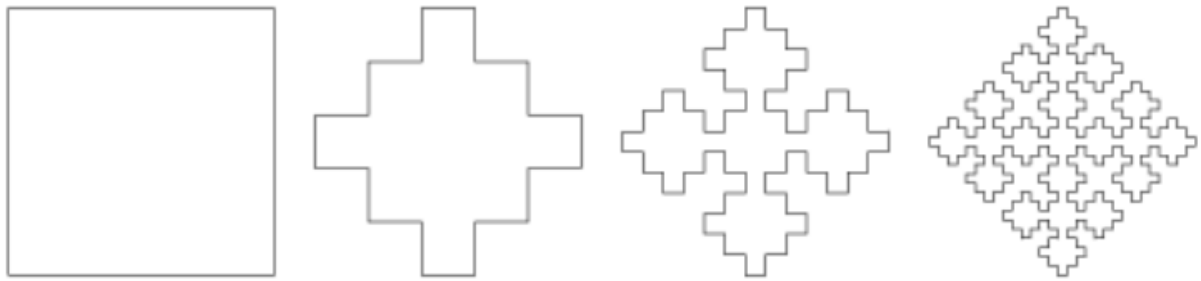
Aici transformarea **s1()** duce triunghiul ΔABC peste triunghiul ΔBDA, iar **s2()** duce pe ΔABC peste ΔADC.

Incercaţi să obţineţi acelaşi rezultat prin metoda motivelor iterate, păstrând la fiecare etapă în lista **fig** triunghiurile obţinute.

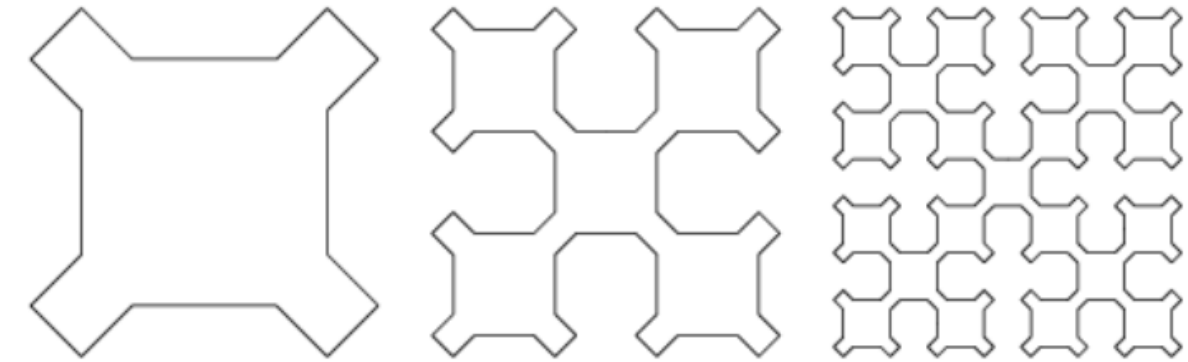


7) Incercaţi să generaţi următoarele c*urbe ale lui Sierpinski*, sugerate de primele etape:

a)



b)



c)