VNIVERSITAT DE VALÈNCIA

ESTRUCTURAS DE DATOS Y ALGORITMOS
***Grado en Ingeniería Multimedia (2º).* Curso 2022-23**

Práctica Nº 4: Algoritmos recursivos

Periodo de realización: Semana del **07/11/2022 a 11/11/2022**

Objetivos

- Diseñar algoritmos recursivos.
- Implementar funciones recursivas en C++.
- Medir el coste de algoritmos recursivos.

Introducción

Un fractal[1] es un objeto geométrico cuya estructura básica, fragmentada o aparentemente irregular, se repite a diferentes escalas[2]. Estas formas constan de fragmentos de orientación y tamaño variable, pero de aspecto similar. Si se amplían a diferentes escalas (niveles de detalle) se observa que la estructura se repite. En la Fig.1 se muestra un ejemplo:

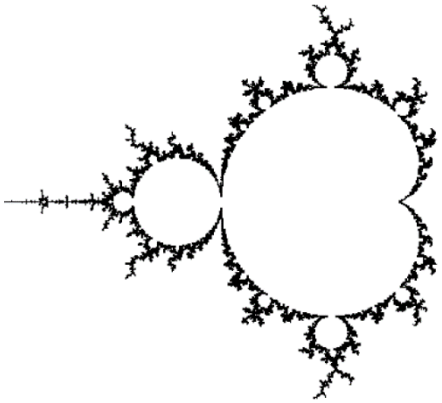


Fig. 1: Ejemplo de fractal tipo Madelbrot

La curva de Koch

La llamada curva de Koch es un tipo de curva fractal que se construye mediante segmentos de recta, que a cada nuevo nivel de escala se van sustituyendo por otros cuatro segmentos, cada uno de ellos con 1/3 de longitud respecto al anterior y con diferentes ángulos de inclinación. Las curvas, para el nivel 0 y nivel 1, se muestran en la parte superior e inferior de la Fig.2:

[1] https://es.wikipedia.org/wiki/Fractal
[2] Benoît Mandelbrot, La Geometría Fractal de la Naturaleza

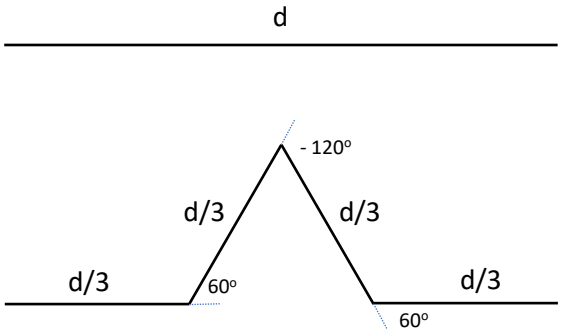


Fig. 2: Nivel 0 (superior) y nivel 1 (inferior)

Como se puede observar, el nivel 0 corresponde a un único segmento de recta de longitud *d*. Si "ampliamos" la escala (hacemos "zoom") pasamos al nivel 1 de detalle (parte inferior de Fig.2). En este segundo nivel el segmento original se divide en 4 segmentos más pequeños, todos de longitud *d/3*, donde el primero aparece con la misma orientación que el original, el segundo girado 60º (pi/3 radianes) respecto al primero, el tercero con un giro de -120º (-2pi/3 radianes) respecto al segundo y el cuarto con un giro de 60º (pi/3 radianes) respecto al tercero, volviendo así a la orientación original. Por lo tanto, la curva de Koch de nivel 1 se construye de la siguiente manera:

```
Curva de Koch (nivel=0, tamaño=d/3)
Girar (pi / 3)
Curva de Koch (nivel=0, tamaño=d/3)
Girar (- 2 * pi / 3)
Curva de Koch (nivel=0, tamaño=d/3)
Girar (pi / 3)
Curva de Koch (nivel=0, tamaño=d/3)
```

La misma estrategia se sigue para obtener la curva de Koch a cualquier nivel. Así, por ejemplo, la curva de Koch de nivel 2 estaría formada por 4 curvas de Koch de nivel 1 con las mismas orientaciones indicadas en el ejemplo anterior. Por lo tanto, se puede definir el siguiente algoritmo para generar la curva de Koch para un nivel de detalle (*nivel*) y tamaño (*distancia*) dados:

```
Algoritmo CurvaKoch(nivel: ℕ, distancia: ℝ⁺)
   si nivel = 0
       dibujar segmento de recta de longitud = distancia
   si no
       CurvaKoch (nivel – 1, distancia / 3)
       girar un ángulo pi/3
       CurvaKoch (nivel – 1, distancia / 3)
       girar un ángulo -2*pi/3
       CurvaKoch (nivel – 1, distancia / 3)
       girar un ángulo pi/3
       CurvaKoch (nivel – 1, distancia / 3)
  fsi
```

Algoritmo 1: Generar una curva de Koch de cualquier nivel y tamaño.

Evidentemente, como toda curva se define como la composición de 4 curvas de nivel inferior[3], el algoritmo 1 es recursivo de tipo múltiple.

La figura de Koch

Además de *curvas* ("abiertas"), también se pueden definir *figuras* de Koch ("cerradas") a partir de un polígono cuyos lados son curvas de Koch. El llamado "*copo de nieve*" de Koch es una figura que se compone a partir de un triángulo inicial cuyos lados son tres curvas de Koch, tal y como se indica en la Fig. 3.

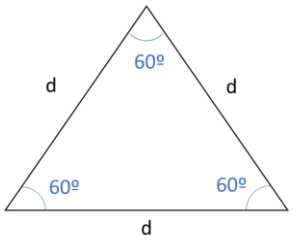


Fig. 3: Triangulo para generar la figura de Koch (nivel 0)

Esta figura cerrada se puede "observar" a diferentes escalas o niveles. En cada uno de estos niveles cada lado del triángulo de la figura es sustituido por la curva de Koch del mismo nivel, dando lugar a las figuras mostradas en la Fig.4:

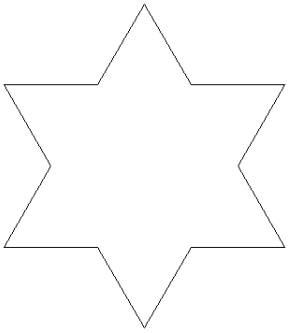


Figura de nivel 1 (*12* segmentos)

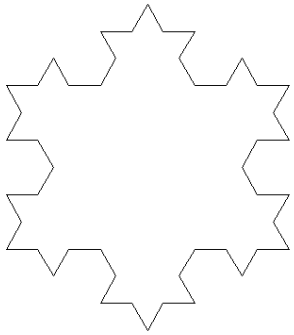


Figura de nivel 2 (*48* segmentos)

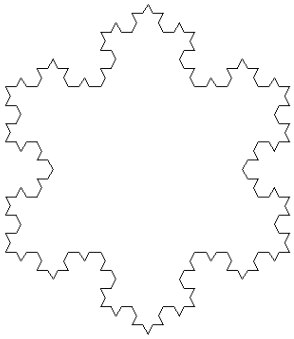


Figura de nivel 3 (*192* segmentos)

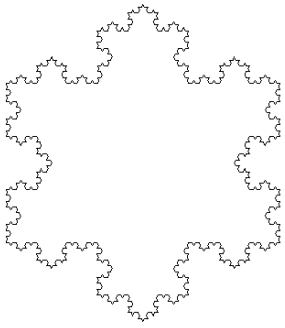


Figura de nivel 4 (*768* segmentos)

Fig. 4: Figura ("copo de nieve") de Koch para diferentes niveles y el número de segmentos que la componen.

[3] Excepto la curva de nivel 0 que se define como una recta (caso base).

El algoritmo para generar figuras cerradas de Koch se puede definir a partir del algoritmo 1 de generación de curvas de la siguiente manera, en función del nivel y tamaño:

```
Algoritmo CopoKoch(nivel: ℕ, distancia: ℝ⁺)
   CurvaKoch (nivel, distancia)
   girar un ángulo -2*pi/3
   CurvaKoch (nivel, distancia)
   girar un ángulo -2*pi/3
   CurvaKoch (nivel, distancia)
```

Algoritmo 2: Generar una figura de Kooch de cualquier nivel y tamaño.

De igual manera, se puede definir una variante de esta figura denominado "anticopo de nieve" de Koch. En el "anticopo de nieve" las curvas de Koch correspondientes a cada lado del triángulo se construyen orientadas hacia el centro del triángulo. Esto se consigue sin más que construir las curvas de Koch con los ángulos cambiados de signo, como se indica en la figura 5.

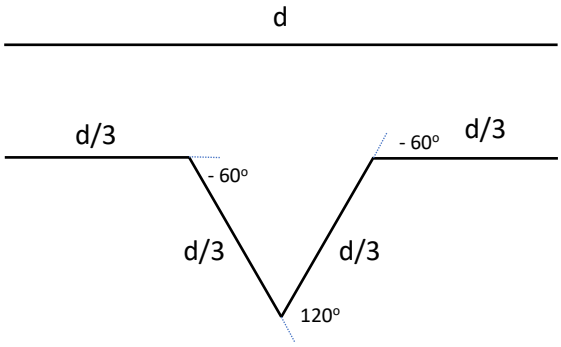


Fig. 5: "Anticurvas" de Koch de nivel 0 (superior) y nivel 1 (inferior)

Para esta variante de la curva original, en el segundo nivel el segmento original se divide en 4 más pequeños, todos de longitud *d/3*, donde el primero aparece con la misma orientación que el original, el segundo girado -60º (-pi/3 radianes) respecto al primero, el tercero con un giro de 120º (2pi/3 radianes) respecto al segundo y el cuarto con un giro de -60º (-pi/3 radianes) respecto al tercero. Por lo tanto, la "anticurva" de Koch de nivel 1 se construye de la siguiente manera:

```
Anticurva de Koch (nivel=0, tamaño=d/3)
Girar (-pi / 3)
Anticurva de Koch (nivel=0, tamaño=d/3)
Girar (2 * pi / 3)
Anticurva de Koch (nivel=0, tamaño=d/3)
Girar (-pi / 3)
Anticurva de Koch (nivel=0, tamaño=d/3)
```

De manera completamente análoga a la curva básica se puede definir el siguiente algoritmo para generar la *anticurva* de Koch para un nivel de detalle (`nivel`) y tamaño (`distancia`) dados:

```
Algoritmo AntiCurvaKoch(nivel: ℕ, distancia: ℝ⁺)
   si nivel = 0
        dibujar segmento de recta de longitud = distancia
   si no
        AntiCurvaKoch (nivel – 1, distancia / 3)
        girar un ángulo -pi/3
        AntiCurvaKoch (nivel – 1, distancia / 3)
        girar un ángulo 2*pi/3
        AntiCurvaKoch (nivel – 1, distancia / 3)
        girar un ángulo -pi/3
        AntiCurvaKoch (nivel – 1, distancia / 3)
  fsi
```

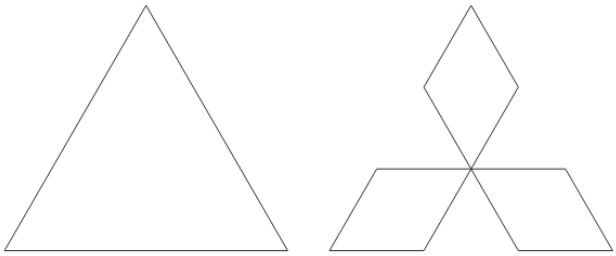
Algoritmo 3: Generar una "anticurva" de Koch de cualquier nivel y tamaño.

De manera equivalente, se puede definir el algoritmo para generar el "anticopo de nieve" de Koch (figura cerrada) de cualquier nivel y tamaño:

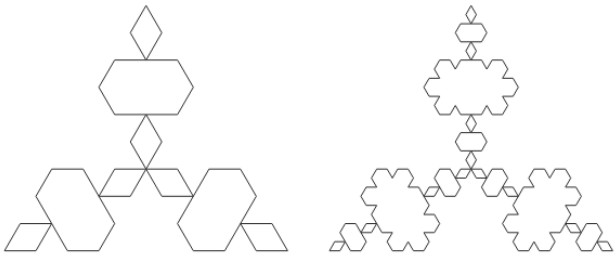
```
Algoritmo AnticopoKoch(nivel: ℕ, distancia: ℝ⁺)
   AntiCurvaKoch (nivel, distancia)
   girar un ángulo -2*pi/3
   AntiCurvaKoch (nivel, distancia)
   girar un ángulo -2*pi/3
   AntiCurvaKoch (nivel, distancia)
```

Algoritmo 2: Generar una figura de Kooch de cualquier nivel y tamaño.

La generación del anticopo de nieve de Koch para diferentes niveles da lugar a las figuras mostradas en la Fig.6:



Anticopo de Nivel 0 Anticopo de Nivel 1



Anticopo de Nivel 2 Anticopo de Nivel 3

Fig. 6: "Anticopo de nieve" de Koch de niveles 0 a 3.

En cualquier caso, tanto para el "copo de nieve" como para el "anticopo de nieve", el coste del algoritmo para generar la figura se puede calcular en función del **nivel de detalle**[4], que determina el número de segmentos que se dibujan en la imagen. Esta es la operación crítica del algoritmo y cuantos más segmentos formen la figura más costará generarla.

El coste de generar cualquier figura de Koch, atendiendo a la sentencia crítica indicada, se puede calcular de forma exacta y es exponencial con el nivel. Una figura, de cualquiera de los dos tipos, requiere generar 3 curvas y cada una de estas curvas requiere generar 4 de nivel inferior, por tanto, el coste del algoritmo es:

$$T(nivel) = 3*4^{nivel}$$

Pasar de un nivel al siguiente implica multiplicar por 4 el número de segmentos generados y, por tanto, el algoritmo tiene un coste muy elevado. Así, por ejemplo, como se ha visto en Fig. 3 y Fig.4, la figura de nivel 0 tiene 3 segmentos, la de nivel 4, pasa a tener 768, la de nivel 8 tendrá 196.608, la de nivel 12, tendrá 50.331.648[5] y la de nivel 20, 3,29853E+12[6].

Objetivo de la práctica

En esta práctica se trata de generar figuras de Koch de diferentes niveles y representarlas gráficamente con *gnuplot*. También, se medirá el número de segmentos que forman cada una de estas figuras para verificar el comportamiento teórico de los algoritmos.

Trabajo a realizar

`Git`: Debes trabajar en la carpeta "Pr4" del repositorio que ya tienes creado para la asignatura. Descarga los archivos disponibles en Aula Virtual y realiza un primer *commit* ("`Pr.4: Versión inicial`") y *push* con ese material inicial. A partir de aquí, ya sabes que debes registrar periódicamente los cambios realizados. La política de gestión de los repositorios es responsabilidad de los estudiantes y se valorará en el apartado "Organización" de los criterios de evaluación de las prácticas.

Se recomienda identificar todos los *commits* como "`Pr.4: …`" para facilitar la identificación de cambios en el repositorio.

Tarea 0: Clase Estado

La representación gráfica de las figuras de Koch no se va a realizar desde el programa en C++ a desarrollar durante la práctica. Este programa deberá generar todos los datos que permitirán a una aplicación externa de representación gráfica, como *gnuplot*, mostrar las figuras por pantalla. Por lo tanto, el programa se limitará a calcular los puntos donde se deben pintar cada uno de los segmentos que forman las figuras (y las curvas) y proporcionarlos en un formato que *gnuplot* pueda representar adecuadamente. Para ello, se hará uso de la capacidad de representar segmentos en gnuplot, que requiere indicar para cada segmento la siguiente información (4 valores): coordenada x de inicio,

[4] Es la talla de este problema.
[5] Más de 50 millones de segmentos (!).
[6] Más de 3 billones de segmentos (!!).

coordenada y de inicio, desplazamiento en x del final y desplazamiento en y del final. Así, por ejemplo, los datos: `1, 0, 7, 3` representan un segmento de recta que comienza en la posición `(1,0)` y cuyo final estará en `(1+7,0+3) = (8,3)`.

Lo que deberá hacer nuestro programa es generar archivos de datos con los valores de los segmentos que forman las diferentes figuras de acuerdo con el formato indicado (1 segmento = 4 valores). Con este objetivo, se proporciona como material inicial una clase, llamada *Estado*, que representa puntos en el espacio 2D, con coordenadas x e y, más un `ángulo` que indica la orientación en la que se debe realizar el siguiente desplazamiento. Esta clase, tal como indica su documentación, además de disponer de las operaciones típicas para establecer y consultar sus datos (coordenadas y ángulo), dispone de una operación *`Desplazar(d)`*, que desplaza el estado actual a una nueva posición a distancia *d* del anterior, y una operación *`Girar(a)`*, que permite girar la orientación del estado actual un ángulo *a*. La orientación es un dato muy importante para poder calcular los desplazamientos. La clase *Estado* se encuentra completamente implementada y solo se debe utilizar en el programa a desarrollar[7], pero no debe ser modificada.

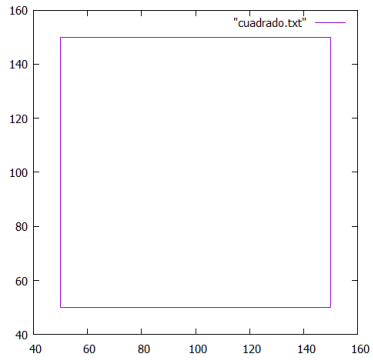
Para probar y comprender mejor el funcionamiento de la clase *Estado*, se proporciona el programa "`testEjemploEstado.cpp`". Este programa utiliza objetos de la clase *Estado* para generar los puntos que representan los 4 segmentos de un cuadrado ubicado en la posición (50,50) (esquina inferior izquierda) y con lados de tamaño 100. Se debe compilar y ejecutar este programa y verificar que se genera el archivo "`data/cuadrado.txt`" con la siguiente información:

```
50    50    6.12303e-15  100
50    150   100   0
150   150   6.12303e-15  -100
150   50    -100   -1.22461e-14
```

Ahora, se debe abrir el programa *gnuplot* y modificar el directorio de trabajo al directorio "*`data`*" de la práctica Pr4. Desde ahí, ejecuta las siguientes órdenes de gnuplot:

```
gnuplot> reset session
gnuplot> set size ratio -1
gnuplot> plot "cuadrado.txt" with vectors nohead
```

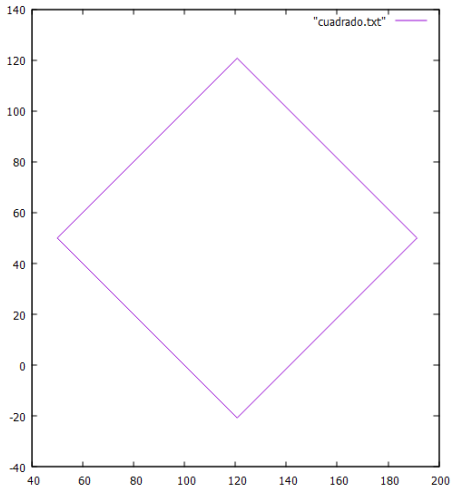
La primera orden reinicia la configuración de *gnuplot*, la segunda establece la misma escala en *x* e *y*, y la tercera es la que representa el archivo de datos como vectores (segmentos) y no como puntos. El resultado debe ser el siguiente:



[7] Es muy importante leer la documentación del código para interpretar correctamente el significado de cada operación.

Lo más importante en esta tarea es interpretar el ejemplo para entender el uso de los objetos de la clase *Estado* y la forma de generar los segmentos en el archivo de salida.

Modifica el programa para que genere un cuadrado que esté girado pi/4 respecto a los ejes. Esto solo requiere modificar el estado inicial antes de generar los 4 segmentos del cuadrado. Comprueba que al representar el resultado con *gnuplot* se obtiene la rotación esperada:



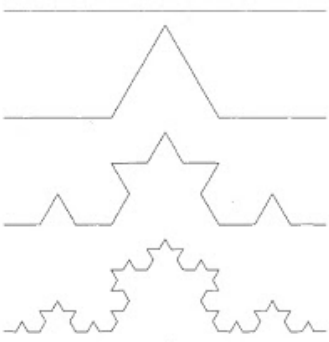
Tarea 1: Generar curvas de Koch

En esta práctica se debe escribir un nuevo programa (*main.cpp*) que llegue a generar figuras de *Koch*. Este programa utilizará la clase *Estado* como base para la representación de la información espacial, de forma similar a como se usado en el programa de test de la tarea 0. Pero antes de generar figuras (cerradas) se requiere implementar la creación de curvas de Koch (abiertas), puesto que las figuras están compuestas por 3 curvas de este tipo. Por tanto, comienza el programa incluyendo una función que permita generar una curva (abierta) de Koch de cualquier nivel y tamaño, de acuerdo con el algoritmo 1 especificado en la Introducción de este guion y con la siguiente especificación:

```
/**
 * @brief Funcion que genera la curva de Koch de un tamaño y nivel
 *
 * @param nivel [in] escala de detalle
 * @param d [in] tamaño de la forma
 * @param e [in] estado espacial
 * @param f [out] archivo de salida
 * @return número de segmentos generados por la curva
 */
size_t GeneraCurvaKoch(unsigned int nivel, double d, Estado& e, ofstream& f)
```

El significado de los argumentos y el valor de retorno está indicado en el comentario previo. El estado *e* indica el punto de inicio de la curva (posición y orientación) y el archivo *f* es donde se deben escribir los datos de los segmentos que constituyen la curva.

Comprueba el correcto funcionamiento de esta función generando desde el programa principal curvas de Koch de diferentes niveles, por ejemplo, de 0 a 3. Representa las curvas obtenidas en *gnuplot*, tal como se ha indicado en la tarea 1 y comprueba que el resultado es el esperado:



Verifica también que el número de segmentos que devuelve la función es correcto:

- Nivel 0: 1 segmento
- Nivel 1: 4 segmentos
- Nivel 2: 16 segmentos
- Nivel 3: 64 segmentos

Tarea 2: Generar el copo de nieve de Koch

Una vez resuelto el problema de generar curvas de Koch, ya es posible generar el *copo de nieve* de Koch (figuras cerradas). Para ello, se debe incluir en el programa una función que realice esta tarea, de acuerdo con el algoritmo 2 de la Introducción y la siguiente especificación:

```
/**
 * @brief Genera el copo de nieve de Koch formado por 3 curvas de Koch
 *
 * @param nivel [in] escala de detalle
 * @param d [in] tamaño de la forma
 * @param e [in] estado espacial
 * @param f [out] archivo de salida
 * @return número de segmentos generados por la figura
 */
size_t GeneraCopoKoch(unsigned int nivel, double d, Estado& e, ofstream& f)
```

Verifica el funcionamiento de esta función generando, al igual que en la tarea 2, figuras de diferentes niveles y representándolas con *gnuplot* (no excedas el nivel 5). Comprueba que el número de segmentos también es correcto (ver Fig.4).

Tarea 3: Generar múltiples copos de nieve de Koch

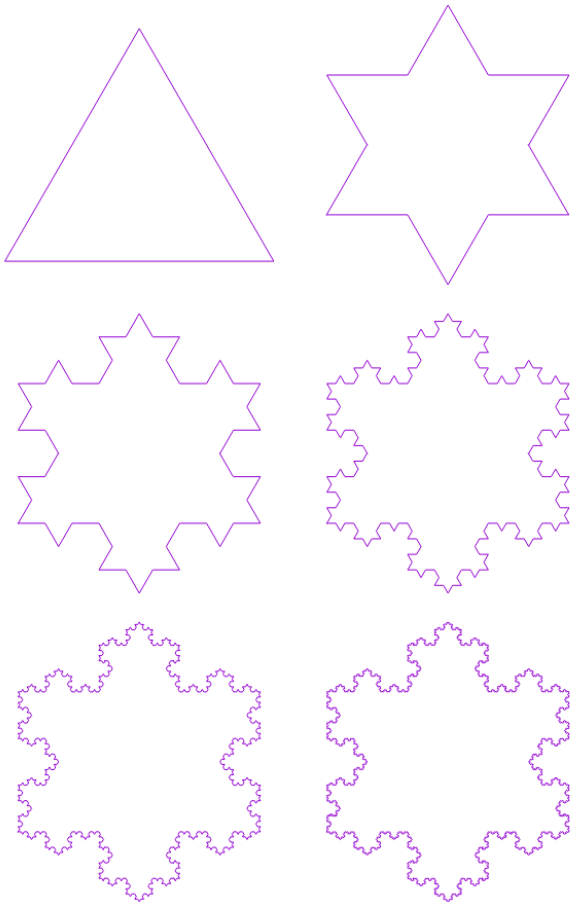
Teniendo ya la función que genera cualquier copo de nieve de Koch, se debe hacer que el programa genere en la misma ejecución los copos de nieve de Koch de niveles 0 a 5, de acuerdo con los siguientes requisitos:

- La posición/estado inicial de las figuras será `(75,100)` y ángulo `pi/3`.
- Los datos de cada figura se deben almacenar en un archivo de texto diferente (en la carpeta "`data`") con los nombres: `koch0.txt`, `koch1.txt`, …, `koch5.txt`.
- Se mostrará el nivel de cada figura generada y el número de segmentos que la componen.

Tras la ejecución del programa se deben representar las figuras. Para ello, se debe usar la siguiente secuencia de órdenes de *gnuplot*[8]:

```
reset session
set term qt size 600, 900
set size ratio -1
unset border
unset xtics
unset ytics
set multiplot layout 3,2
plot "koch0.txt" with vectors nohead notitle
plot "koch1.txt" with vectors nohead notitle
plot "koch2.txt" with vectors nohead notitle
plot "koch3.txt" with vectors nohead notitle
plot "koch4.txt" with vectors nohead notitle
plot "koch5.txt" with vectors nohead notitle
unset multiplot
```

De esta manera, se mostrarán conjuntamente los 6 primeros copos de nieve de Koch:



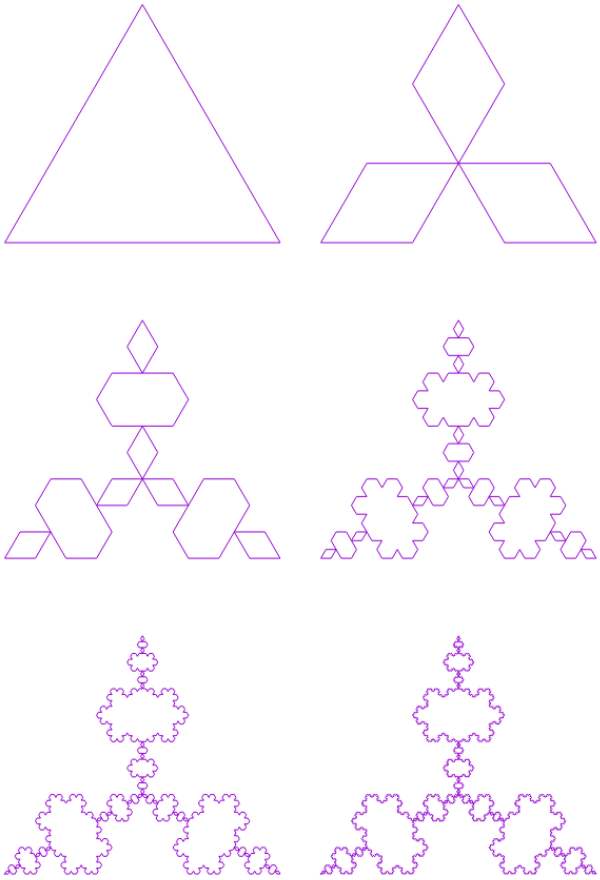
[8] Se recomienda guardarlas en un archivo de texto para facilitar su ejecución.

Tarea 4: Generar anticopos de nieve de Koch

Se debe hacer que el programa genere también en la misma ejecución los anticopos de nieve de Koch de niveles 0 a 5, de acuerdo con los siguientes requisitos:

- La posición/estado inicial de las figuras será `(75,100)` y ángulo `pi/3`.
- Los datos de cada figura se deben almacenar en un archivo de texto diferente (en la carpeta "`data`") con los nombres: antikoch`0.txt`, antikoch`1.txt`, …, antikoch`5.txt`.

Tras la ejecución del programa se deben representar las figuras. Para ello, se debe usar el mismo esquema de representación indicado en la tarea anterior, pero con los nuevos archivos de datos. En este caso, las figuras obtenidas serán:



Tarea 5: Generación de documentación

El código del programa deberá estar correctamente documentado con Doxygen, de acuerdo con la guía de estilo. Una vez finalizado el programa, se deberá ejecutar Doxygen para verificar la correcta generación de la documentación del programa.