



Objetivos

- Practicar las técnicas básicas de definición de funciones en un lenguaje funcional.
- Conocer las funciones de orden superior básicas para tratamiento de listas.
- Diseñar funciones de forma recursiva para el tratamiento de listas.

Tarea

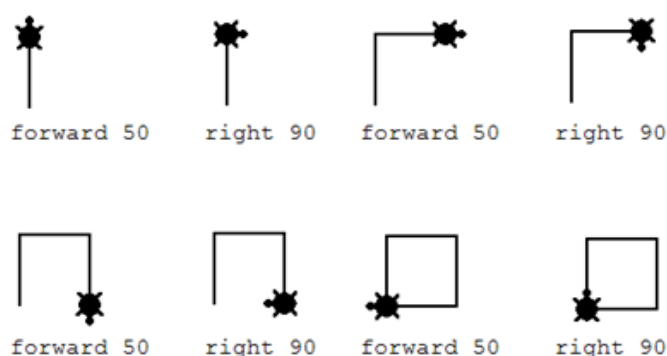
Diseña e implementa en Haskell las funciones que se piden a lo largo de la práctica.

1. Gráficos de Tortuga

El sistema denominado de Gráficos de Tortuga es un entorno programable de dibujo vectorial, popularizado por el lenguaje de programación Logo, que permite generar gráficos 2D basados en líneas poligonales mediante el desplazamiento de un cursor (la tortuga de Logo), que tiene una posición y una orientación en el plano. La tortuga puede moverse para trazar líneas, controlado por ordenes sencillas:

- moverse hacia adelante, en la dirección que indica la orientación de la tortuga, una distancia determinada, trazando una línea.
- girar la orientación de la tortuga hacia la derecha o la izquierda

Por ejemplo, para dibujar un cuadrado el proceso sería el siguiente:



En el lenguaje Logo el dibujo se hace directamente sobre una ventana, escribiendo el programa que controla la tortuga mediante órdenes del lenguaje. En esta práctica, para simplificar, generaremos una línea poligonal que se puede escribir en un fichero con formato SVG, que puedes visualizar abriéndolo con tu navegador. Y nuestras órdenes serán simplemente caracteres en una cadena, que generan nuevas posiciones de la tortuga:

- ">": avanzar la tortuga hacia delante una distancia predefinida
- "+": girar la tortuga hacia la derecha un ángulo predefinido
- "-": girar la tortuga hacia la izquierda un ángulo predefinido

En el módulo `Turtle.hs` puedes ver las definiciones de los tipos de datos y funciones que te damos para manejar la tortuga. Una tortuga (`Turtle`) se define mediante una tupla que contiene la información tanto de la distancia de avance y el ángulo de giro (en grados) predefinidos, como de la posición y orientación actuales. Puedes escribir una lista de datos de tipo `Position` en un SVG mediante la función `'saveSvg "nombre" puntos'`, definida en el módulo `SVG.hs`.

De esa forma, para dibujar un cuadrado, la información a utilizar sería la siguiente:

- Definición inicial de la tortuga: `(1, 90, (0,0), 0)`
- Secuencia de comandos: `">+>+>+>"`

Tarea

Implementa una función Haskell que genere la lista de puntos que corresponde al gráfico generado por una secuencia de comandos almacenados en una cadena:

```
tplot :: Turtle -> String -> [Position]
```

Úsala para generar distintas figuras geométricas (triángulo, cuadrado, círculo) y grábalas en formato SVG mediante las funciones dadas:

```
figura = tplot (1,90,(0,0),0) ">+>+>+>"
saveSvg "cuadrado" figura
```

2. Sistemas de Lindenmayer

Los **Sistemas de Lindenmayer** o **L-Systems** son sistemas de reglas de re-escritura que se utilizan para dibujar elementos naturales, como plantas o copos de nieve, mediante un sistema de gráficos de tortuga.

Un L-System añade al alfabeto de símbolos de la tortuga ("`>`", "`+`", "`-`"), otros símbolos nuevos que pueden ser letras mayúsculas o minúsculas. El sistema parte de un cadena inicial

(*axioma*), que puede re-escribirse reemplazando cada una de las letras por una nueva cadena de símbolos. La cadena que sustituye a cada símbolo se define mediante una serie de reglas de re-escritura. Ese proceso se repite un número determinado de veces, generando una cadena final, que se interpreta para generar un dibujo.

Una regla de re-escritura especifica como se sustituye cada símbolo, por ejemplo:

$$F \rightarrow F+F+F+F$$

Los símbolos para los que no exista regla se quedan como están. Normalmente los símbolos originales de control de la tortuga no se re-escriben. La interpretación final de los símbolos es la siguiente:

- ">", "+", "-": igual que antes
- letra mayúscula: avanza la tortuga, similar a ">"
- letra minúscula: se ignora a la hora de dibujar

A partir de un axioma inicial y una serie de reglas podemos obtener la definición de nuestro gráfico repitiendo iterativamente la aplicación de las reglas. En cada paso se aplican todas las reglas en el orden en que estén definidos los símbolos:

Triángulo de Sierpinsky

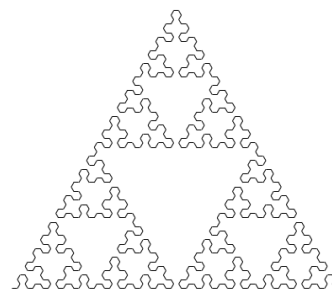
Reglas: $F \rightarrow G-F-G$

$G \rightarrow F+G+F$

Axioma: F

Distancia: 1

Angulo: 60 grados



Tarea

Utilizando el lenguaje Haskell, y suponiendo que existe una función

```
rules :: Char -> String
```

implementa la función **lsystem**, que a partir de la función anterior que define las reglas de re-escritura, el axioma inicial y el número de veces que tenemos que aplicarlas, devuelve la cadena resultante:

```
lsystem :: (Char -> String) -> String -> Int -> String
```

Por ejemplo:

```
rules :: Char -> String
-- Triángulo de Sierpinsky
-- 'F' -> "G-F-G"
-- 'G' -> "F+G+F"
-- otros ?

> lsystem rules "F" 0
"F"
> lsystem rules "F" 1
"G-F-G"
> lsystem rules "F" 2
"F+G+F-G-F-G-F+G+F"
> lsystem rules "F" 3
"G-F-G+F+G+F+G-F-G-F+G+F-G-F-G-F+G+F-G-F-G+F+G+F+G-F-G"
```

Tarea

Define diversas versiones de la función `rules`, que definan distintos L-Systems. En el Apéndice tienes la definición de algunos sistemas conocidos, y puedes encontrar muchas otras en Internet.

Entrega

Deberás entregar todos los ficheros de código fuente que hayas necesitado para resolver el problema, incluyendo el programa principal `main.hs`.

La solución deberá ser compilable y ejecutable con los ficheros que has entregado mediante los siguientes comandos:

```
ghc --make main.hs
./main
```

En caso de no compilar siguiendo estas instrucciones, el resultado de la evaluación de la práctica será de 0. No se deben utilizar paquetes ni librerías ni ninguna infraestructura adicional fuera de las librerías estándar del propio lenguaje.

Todos los archivos de código fuente solicitados en este guión deberán ser comprimidos en un único archivo zip con el siguiente nombre:

- `practica5_<nip1>_<nip2>.zip` (donde `<nip1>` y `<nip2>` son los NIPs de 6 dígitos los estudiantes involucrados) si el trabajo ha sido realizado por parejas. En este caso sólo uno de los dos estudiantes deberá hacer la entrega.

- practica5_<nip>.zip (donde <nip> es el NIP de 6 dígitos del estudiante involucrado) si el trabajo ha sido realizado de forma individual.

El archivo comprimido a entregar no debe contener ningún fichero aparte de los fuentes que te pedimos: ningún fichero ejecutable o de objeto, ni ningún otro fichero adicional.

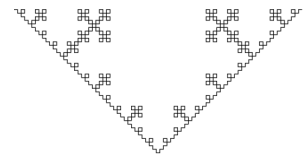
La entrega se hará en la tarea correspondiente a través de la plataforma Moodle:
<http://moodle.unizar.es>.

Apéndice

Definición de diversos L-Systems.

Curva de Koch cuadrada

Reglas: $F \rightarrow F+F-F-F+F$
 Axioma: F
 Distancia: 1
 Angulo: 90 grados



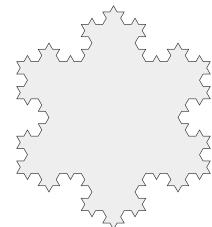
Curva de Koch triangular

Reglas: $F \rightarrow F+F--F+F$
 Axioma: F
 Distancia: 1
 Angulo: 60 grados



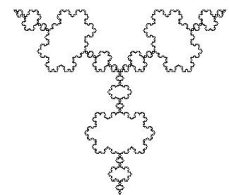
Koch Snowflake

Reglas: $F \rightarrow F-F++F-F$
 Axioma: $F++F++F$
 Distancia: 1
 Angulo: 60 grados



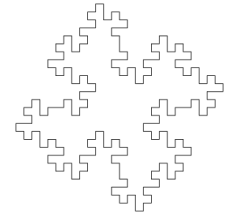
Koch Anti-Snowflake

Reglas: $F \rightarrow F+F--F+F$
 Axioma: $F++F++F$
 Distancia: 1
 Angulo: 60 grados



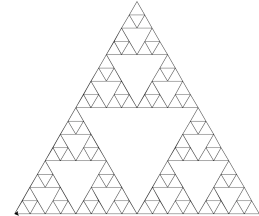
Isla de Minkowski

Reglas: $F \rightarrow F+F-F-F+F+F-F$
Axioma: $F+F+F+F$
Distancia: 1
Angulo: 90 grados



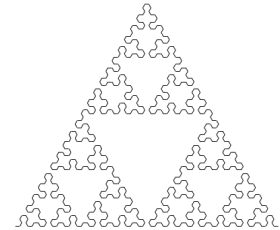
Triángulo de Sierpinsky

Reglas: $F \rightarrow F-G+F+G-F$
 $G \rightarrow GG$
Axioma: $F-G-G$
Distancia: 1
Angulo: 120 grados



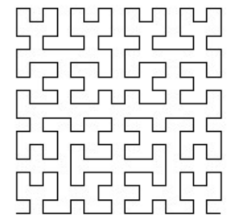
Sierpinsky Arrowhead

Reglas: $F \rightarrow G-F-G$
 $G \rightarrow F+G+F$
Axioma: F
Distancia: 1
Angulo: 60 grados



Curva de Hilbert

Reglas: $f \rightarrow -g>+f>f>+g-$
 $g \rightarrow +f>-g>g->f+$
Axioma: f
Distancia: 1
Angulo: 90 grados



Curva de Gosper

Reglas: $F \rightarrow F-G--G+F++FF+G-$
 $G \rightarrow +F-GG--G-F++F+G$
Axioma: F
Distancia: 1
Angulo: 60 grados

