Haskell Graphic Engine

010111010... Perdón, ¡hola! Queremos implementar nuestro propio motor gráfico en Haskell. ¡Copate y danos una mano!

Ah, pero ¿qué es un motor gráfico? Simple: un software que nos sirve para hacer dibujos en la pantalla. Y para ello, utilizamos dos elementos fundamentales: puntos y polígonos.

Un polígono es una figura que tiene una cantidad arbitraria de puntos 2D y un número indicando su transparencia, llamado alfa. Alfa pude oscilar entre 0 (el polígono no se ve, es transparente) a 255 (es totalmente opaco).

Punto 1

Los polígonos por sí solos no tienen mucho sentido ahí flotando en su mundo; nos va a interesar poder aplicar transformaciones sobre ellos. Desarrollá:

A. **`desplazar`**: recibe un desplazamiento 2D indicando cuánto tiene que movilizarse en cada eje.
B. **`espejar`**: multiplica a los valores de cada punto por -1 para todos sus ejes.
C. **`aclararEn:`** disminuye el alfa del polígono en X unidades, respectivamente. Tener en cuenta el rango de valores posibles.
D. **`oscurecerEn:`** similar a la transformación anterior, pero aumenta el alfa en lugar de disminuirlo.
E. **`simplificar:`** le quita los primeros N puntos al polígono

Escribí además un sinónimo de tipo **`Transformacion`** y utilizalo

Punto 2

En ocasiones queremos modelar transformaciones más complejas, que se construyen combinando otras transformaciones. Desarrollá:

A. **`barrelRoll`**: toma una transformación y la repite una cantidad X de veces.
B. **`pipelineCaprichoso`**: toma varias transformaciones y aplica sucesivamente todas aquellas que no hagan que el polígono quede en el origen; decimos que un polígono queda en el origen cuando alguno de sus puntos está en el origen. Por ejemplo:

```
> pipelineCaprichoso [restar1AlZDel2doPunto,
                      restar1AlZDel2doPunto,
                      espejar] (Poligono [(0, 0, 4), (0, 0, 2), (0, 0, 4)] 15)
(Poligono [(0, 0, -4), (0, 0, -1), (0, 0, -4)] 15)
```

Explicación: **`pipelineCaprichoso`** va a primero restarle 1 al segundo punto, luego va a ignorar la segunda función (porque dejaría al segundo punto en el origen), y finalmente, espejar el polígono resultante

Punto 3

Queremos saber si una transformación es inútil: lo cual es cierto si al aplicarlo sobre un polígono, no produce cambios. Desarrollá:

A. **`esInutil`**: que toma una transformación y un polígono y dice si es inutil según el párrafo anterior.
B. **`transformacionesUtiles`**: que toma una lista de transformaciones y un polígono, y se queda sólo con las útiles.

Punto 4

Por último, queremos representar a las animaciones como secuencias infinitas de polígonos. Por ejemplo, si tenés la siguiente lista infinita de polígonos:

```
[cuadradoChico, cuadradoMediano, cuadradoGrande, cuadradoMasGrande, etc, ...
```

Representa la animación de un cuadrado que fue creciendo en tamaño. Sabiendo eso, desarrollá las siguientes animaciones simples:

A. **`pasearPorLasX`**: toma un polígono y lo desplaza de a una unidad por vez a través del eje X.
B. **`parpadear`**: toma un polígono y un nivel de parpadeo X, y lo aclara en X unidades, luego lo oscurece en X unidades, luego lo aclara en X unidades, luego lo oscurece en X unidades y asi.

Consideraciones:

- Escribir el tipo de **todas** las funciones.
- Emplear sinónimos de tipo cuando sea posible.
- Se puede usar recursividad máximo una vez
- Evitar la repetición de lógica.