# Programación Funcional - Parcial Animaciones

En esta evaluación se busca modelar animaciones de un juego. Para eso se trabajará sobre una representación en el tipo de datos **Animacion**, dado por las siguientes declaraciones:

```
data Accion a = Paso a | SaltoArriba a | SaltoAdelante a | Girar a
type Tiempo = Int     -- el instante en el que sucede un movimiento
type Duracion = Int   -- cantidad de tiempos que dura el movimiento
data Animacion a =
    Espera Duracion                  -- durante la duración dada no hay acciones
  | Mov Duracion (Accion a)          -- un cierto movimiento con una duración dada
  | Sec (Animacion a) (Animacion a)  -- secuencia (la 2da empieza al terminar la 1era)
  | Par (Animacion a) (Animacion a)  -- paralelo (arrancan juntas y dura lo que la más larga)
```

Por ejemplo, la animación en paralelo de un personaje que va caminando y vuelve saltando, y otro que va saltando y vuelve caminando y luego esperan 1 segundo, se expresaría de la siguiente forma:

```
ej = Sec (Par (Sec (Sec (Espera 1) (Mov 3 (Paso "Bob")))
                   (Sec (Mov 1 (Girar "Bob")) (Mov 2 (SaltoAdelante "Bob"))))
              (Sec (Mov 2 (SaltoAdelante "Ana"))
                   (Sec (Mov 1 (Girar "Ana")) (Sec (Mov 3 (Paso "Ana")) (Espera 1)))))
         (Espera 1)
```

Además se utilizarán dos tipos adicionales para realizar conversiones de composiciones a otros formatos. Estos tipos son (en ambos casos, los tiempos se cuentan desde 1):

```
type Frame a = [Accion a]     -- acciones simultáneas en un tiempo específico
type Simulador a = Tiempo -> Frame a
-- función que da las acciones que ocurren en un tiempo dado
```

Por ejemplo, la animación del ejemplo, expresada como lista de acciones sería

```
fsEJ =
 [
        [SaltoAdelante "Ana"]   , [Paso "Bob", SaltoAdelante "Ana"] ,
         [Paso "Bob", Paso "Ana"] , [Giro "Ana", Paso "Bob"] ,
         [Giro "Bob", Paso "Ana"] , [SaltoAdelante "Bob", Paso "Ana"] ,
         [SaltoAdelante "Bob"] , []
 ]
```

y llamando **`simEj`** al simulador que expresa la misma animación del ejemplo repitiéndose una y otra vez, valen las siguientes equivalencias:

```
simEj 1 = [SaltoAdelante "Ana"]
simEj 2 = [Paso "Bob", SaltoAdelante "Ana"]
simEj 4 = [Paso "Bob", Paso "Ana"]
simEj 6 = []
simEj 7 = [SaltoAdelante "Bob"]
simEj 8 = [SaltoAdelante "Bob", Paso "Ana"]
simEj 9 = [SaltoAdelante "Ana"]
simEj 12 = [Paso "Bob", Paso "Ana"]
simEj 16 = []
```

# Ejercicios

1. Definir la siguiente función de forma tal que cada elemento de cada lista sea considerado exactamente una vez (o sea, debe realizar UN único recorrido sobre cada lista).
   a. **combinarSinDuplicados :: [Int] -> [Int] -> [Int]**, que dadas dos listas de números sin repetidos, ordenadas en forma creciente, describe una lista ordenada y sin repetidos que contiene los elementos de ambas listas.
   **AYUDA:** es **IMPRESCINDIBLE** tener en cuenta el orden de los elementos de las listas para poder cumplir con la exigencia de un único recorrido.
2. Definir las siguientes funciones utilizando recursión explícita sobre **Animacion**
   a. **duracion :: Animacion a -> Int**, que describe la duración total de la animación (considerando que las acciones en paralelo terminan cuando termina la más larga).
   b. **alargar :: Int -> Animacion a -> Animacion a**, que describe una animación donde la duración de cada acción está multiplicada por un factor con respecto a la duración original.
   c. **simular :: Animacion a -> [Frame a]**, que describe una lista con un frame por cada tiempo de la duración de la animación. Cada frame es una lista con las acciones que ocurren simultáneamente en el tiempo indicado (una lista vacía indica que el personaje está quieto).
   **AYUDA:** puede usarse sin definir la función **replicate :: Int -> a -> [a]**, que dado un número n y un elemento, devuelve una lista con n copias de ese elemento.
   d. **tiemposDeEspera :: Animacion a -> [Tiempo]**, que describe una lista de los tiempos de la animación donde no hay ninguna acción.
   **ATENCIÓN:** DEBE ser realizada por recursión explícita.
   **AYUDA 1:** puede servir utilizar la función del ejercicio 1, y también definir la función auxiliar **contarHasta :: Int -> [Int]**, que describe la lista de números desde 1 hasta el dado.
   **AYUDA 2:** al armar los tiempos de la secuencia, tener en cuenta que los tiempos de la 2da componente de la secuencia van DESPUÉS de los de la primera…
3. Demostrar la siguiente propiedad para las funciones del ejercicio 1:
   **para todo k >= 0.** ***duracion . (alargar k) = (k*) . duracion***
4. Dar los tipos y escribir los esquemas de recursión estructural y primitiva para **Animacion**.
5. Escribir versiones de **todas** las funciones recursivas del ejercicio 2 utilizando esquemas, y sin utilizar recursión explícita en **ninguna** función diferente de los esquemas (excepto la del ejercicio 1).
6. Escribir las siguientes funciones utilizando esquemas, y sin utilizar recursión explícita en **ninguna** función diferente de los esquemas. Pueden utilizarse las funciones del ejercicio 4.
   a. **ciclar :: Animacion a -> Simulador a**, que dada una animación describe un simulador en el que la animación se repite infinitamente.
   b. **combinar :: [Animacion a] -> [Animacion a] -> Animacion a**, que dadas dos listas con la misma longitud, describe una Animacion que consiste en la secuencia tal que la animación de cada posición de una lista ocurre en simultáneo con la correspondiente en la otra lista.
   c. **mezclar :: [Simulador a] -> Duracion -> [Frame a]**, que describe una lista de

Frames con la duración dada, donde en cada tiempo se observa la superposición de los simuladores en la lista de entrada.