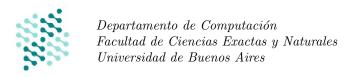
Introducción a la Programación

Guía Práctica 5++ Renombre de Tipos



La primera parte de esta guía usa los ejercicios de la guía 3, pero con renombre de tipos. La segunda parte deben realizar los ejercicios usando los renombre sugeridos para listas y tuplas.

 $\label{thm:com/making-our-own-types-and-type} Ver \ \textit{Type synonyms} \ en \ \texttt{https://wiki.haskell.org/Type\#Type_and_newtype} \\ \text{haskell.org/Type\#Type_and_newtype}$

Ejemplo, renombro las apariciones del tipo entero en edad: type Edad = Int

1. Renombre de tipos básicos

Ejercicio 1. Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números.

- a) Reimplementar ejercicio 4a prodInt: calcula el producto interno entre dos tuplas $\mathbb{R} \times \mathbb{R}$. Usar type Punto2D = (Float, Float)
- b) Reimplementar ejercicio 4b de la guía 3 todoMenor: dadas dos tuplas $\mathbb{R} \times \mathbb{R}$, decide si es cierto que cada coordenada de la primera tupla es menor a la coordenada correspondiente de la segunda tupla. Usar type Punto2D = (Float, Float)
- c) Reimplementar ejercicio 4c de la guía 3: distanciaPuntos: calcula la distancia entre dos puntos de \mathbb{R}^2 . Usar type Punto2D = (Float, Float)
- d) Reimplementar función del ejercicio 4g, de la guía 3: crearPar :: Float ->Float ->coordenada. Usar type Coordenada
 = (Float, Float)

```
type Año = Integer  
type EsBisiesto = Bool  
bisiesto :: Año ->EsBisiesto  
problema bisiesto (año: \mathbb{Z}) : Bool {  
    requiere: \{True\}  
    asegura: \{res = false \leftrightarrow \text{año no es múltiplo de 4 o año es múltiplo de 100 pero no de 400} \} 
Por ejemplo: bisiesto 1901 \leadsto False,  
bisiesto 1904 \leadsto True,  
bisiesto 1900 \leadsto False,  
bisiesto 2000 \leadsto True.
```

```
Ejercicio 3. Reimplementar el ejercicio 7 de la guía 3 usando: type Coordenada3d = (Float, Float) distanciaManhattan :: Coordenada3d ->Coordenada3d ->Float
```

```
problema distanciaManhattan (p: \mathbb{R} \times \mathbb{R} \times \mathbb{R}, q: \mathbb{R} \times \mathbb{R} \times \mathbb{R}): \mathbb{R} { requiere: \{True\} asegura: \{res = \sum_{i=0}^2 |p_i - q_i|\} } Por\ ejemplo: distanciaManhattan (2, 3, 4) (7, 3, 8) \leadsto 9 distanciaManhattan ((-1), 0, (-8.5)) (3.3, 4, (-4)) \leadsto 12.8
```

Ejercicio 2. Reimplementar la función del ejercicio 6 de la guía 3 usando:

2. Renombre de secuencias

Ejercicio 4. En este ejercicio trabajaremos con la lista de contactos del teléfono.

- a) Implementar una función que me diga si una persona aparece en mi lista de contactos del teléfono: enLosContactos :: Nombre ->ContactosTel ->Bool
- b) Implementar una función que agregue una nueva persona a mis contactos, si esa persona está ya en mis contactos entonces actualiza el teléfono. agregarContacto :: Contacto ->ContactosTel ->ContactosTel

c) Implementar una función que dado un nombre, elimine un contacto de mis contactos. Si esa persona no está no hace nada. eliminarContacto :: Nombre ->ContactosTel ->ContactosTel

Para esto definiremos los siguientes tipos:

```
type Texto = [Char]
type Nombre = Texto
type Telefono = Texto
type Contacto = (Nombre, Telefono)
type ContactosTel = [Contacto]
```

Sugerencia: Implementar las funciones auxiliares el Nombre y el Telefono para que dado un contacto devuelva el dato del nombre y el teléfono respectivamente.

Ejercicio 5. En este ejercicio trabajaremos con lockers de una facultad.

- 1. Implementar existeElLocker :: Identificacion ->MapaDeLockers ->Bool, una función para saber si un locker existe en la facultad.
- 2. Implementar ubicacionDelLocker :: Identificacion ->MapaDeLockers ->Ubicacion, una función que dado un locker que existe en la facultad, me dice la ubicación del mismo.
- 3. Implementar estaDisponibleElLocker :: Identificacion ->MapaDeLockers ->Bool, una función que dado un locker que existe en la facultad, me devuelve Verdadero si esta libre.
- 4. Implementar ocuparLocker :: Identificación ->MapaDeLockers ->MapaDeLockers, una función que dado un locker que existe en la facultad, y está libre, lo ocupa.

Para resolverlo usaremos un tipo MapaDelockers que será una secuencia de locker.

Cada locker es una tupla con la primera componente correspondiente al número de identificación, y la segunda componente el estado.

El estado es a su vez una tupla cuya primera componente dice si esta ocupado o libre, y la segunda componente es un texto con el código de ubicación del locker.

```
type Identificacion = Integer
type Ubicacion = Texto
type Estado = (Disponibilidad, Ubicacion)
type Locker = (Identificacion, Estado)
type MapaDeLockers = [Locker]
data Disponibilidad = Libre | Ocupado deriving (Eq, Show)
```

Opcional: Pueden usar el tipo enumerado data Disponibilidad = Libre | Ocupado Por ejemplo, un posible mapa de lockers puede ser:

```
lockers =
    [
    (100,(Ocupado,"ZD39I")),
    (101,(Libre,"JAH3I")),
    (103,(Libre,"IQSA9")),
    (105,(Libre,"QOTSA")),
    (109,(Ocupado,"893JJ")),
    (110,(Ocupado,"99292"))
]
```