Introducción a la Programación

Algoritmos y Estructuras de Datos I

Práctica 5: Recursión sobre listas

Primer cuatrimestre de 2025

Departamento de Computación - FCEyN - UBA

Práctica 5: Listas en Haskell

Ejercicio 2.5

Implementar `quitar :: (Eq t) => t -> [t] -> [t]`, que dados un entero x y una lista xs, elimina la primera aparición de x en la lista xs (de haberla).

Posible solución Ej2.5

```
quitar :: Int -> [Int] -> [Int]
quitar _ [] = []
quitar e (x:xs) | e == x = xs
                | otherwise = x: quitar e xs
```

Ejercicio 3.3

Definir las siguientes funciones sobre listas de enteros

`problema maximo` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ $\{$
  `requiere:` $\{ |s| > 0 \}$
  `asegura:` $\{ resultado \in s \land$ todo elemento de s es menor o igual a $resultado\}$
$\}$

Posible solución Ej3.3

```
maximo :: [Int] -> Int
maximo [x] = x
maximo (x:y:xs) | x > y = maximo (x:xs)
                | otherwise = maximo (y:xs)
```

Ejercicio 3.9

Definir las siguientes funciones sobre listas de enteros

`problema ordenar` (s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle ent\rangle$ {
  `requiere:` { True }
  `asegura:` { $resultado$ contiene los elementos de s ordenados de forma creciente}
}

Sugerencia: Hay muchas formas distintas de ordenar secuencias. Una opción puede ser utilizar la función `máximo` y la función `quitar` que ya implementamos.

Posible solución Ej3.9

```
ordenar :: [Int] -> [Int]
ordenar [] = []
ordenar xs = (ordenar (quitar max xs)) ++ [max]
             where max = maximo xs
```

Ejercicio 6

Usando los siguientes tipos de datos:

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

Ejercicio 6

Usando los siguientes tipos de datos:

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

a) Implementar una función que me diga si una persona aparece en mi lista de contactos del teléfono: `enLosContactos :: Nombre -> ContactosTel -> Bool`

Ejercicio 6

Usando los siguientes tipos de datos:

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

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`

Posible solución Ej6

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

enLosContactos :: Nombre -> ContactosTel -> Bool
enLosContactos _ [] = False
enLosContactos nombre ((contactoNombre, _):contactos) = nombre == contactoNombre
                                    || enLosContactos nombre contactos

agregarContacto :: Contacto -> ContactosTel -> ContactosTel
agregarContacto c [] = [c]
agregarContacto (nombre, nuevoTel) ((contactoNombre, contactoTel):contactos) |
                contactoNombre == nombre = (nombre, nuevoTel):contactos
```