# Trabajo Práctico # 2

Estructuras de Datos, Universidad Nacional de Quilmes

9 de septiembre de 2017

## 1. Tipos de datos definidos por el usuario

1. Definir el tipo de dato Dir, con las alternativas Norte, Sur, Este y Oeste. Luego implementar las siguientes funciones:

   - `opuesto :: Dir -> Dir`
     Dada una dirección devuelve su opuesta
   - `siguiente :: Dir -> Dir`
     Dada una dirección devuelve su siguiente, en sentido horario.

2. Definir el tipo de dato Persona, como un nombre la edad de la persona. Realizar las siguientes funciones:

   - `nombre :: Persona -> String`
     Devuelve el nombre de una persona
   - `edad :: Persona -> Int`
     Devuelve la edad de una persona
   - `crecer :: Persona -> Persona`
     Dada una persona la devuelve con su edad aumentada en 1.
   - `cambioDeNombre :: String -> Persona -> Persona`
     Dados un nombre y una persona, reemplaza el nombre de la persona por este otro.
   - `esMenorQueLaOtra :: Persona -> Persona -> Bool`
     Dadas dos personas indica si la primera es más joven que la segunda.
   - `mayoresA :: Int -> [Persona] -> [Persona]`
     Dados una edad y una lista de personas devuelve todas las personas que son mayores a esa edad.
   - `promedioEdad :: [Persona] -> Int`
     Dada una lista de personas devuelve el promedio de edad entre esas personas. La lista al menos posee una persona.
   - `elMasViejo :: [Persona] -> Persona`
     Dada una lista de personas devuelve la persona más vieja de la lista. La lista al menos posee una persona.

3. Definir los tipos de datos *Pokemon*, como un *TipoDePokemon* (agua, fuego o planta) y un porcentaje de energía; y *Entrenador*, como un nombre y una lista de Pokemon. Luego definir las siguientes funciones:

- `leGana :: TipoDePokemon -> TipoDePokemon -> Bool`
  Dado un *TipoDePokemon* y otro, devuelve True si el primero le gana al segundo. Agua gana a fuego, fuego a planta y planta a agua.
- `leGanaP :: Pokemon -> Pokemon -> Bool`
  Dados dos pokemon indica si el primero le puede ganar al segundo. Se considera que gana si su elemento es opuesto al del otro pokemon. Si poseen el mismo elemento se considera que no gana.
- `capturarPokemon :: Pokemon -> Entrenador -> Entrenador`
  Agrega un pokemon a la lista de pokemon del entrenador.
- `cantidadDePokemons :: Entrenador -> Int`
  Devuelve la cantidad de pokemons que posee el entrenador.
- `cantidadDePokemonsDeTipo :: TipoDePokemon -> Entrenador -> Int`
  Devuelve la cantidad de pokemons de determinado tipo que posee el entrenador.
- `lePuedeGanar :: Entrenador -> Pokemon -> Bool`
  Dados un entrenador y un pokemon devuelve *True* si el entrenador posee un pokemon que le gane ese pokemon.
- `esExperto :: Entrenador -> Bool`
  Dado un entrenador devuelve True si ese entrenador posee al menos un pokemon de cada tipo posible.
- `fiestaPokemon :: [Entrenador] -> [Pokemon]`
  Dada una lista de entrenadores devuelve una lista con todos los pokemon de cada entrenador.

4. Tenemos los siguientes tipos de datos

```
data Pizza = Prepizza
   | Agregar Ingrediente Pizza

data Ingrediente = Salsa
                 | Queso
                 | Jamon
                 | AceitunasVerdes Int
```

Definir las siguientes funciones:

- `ingredientes :: Pizza -> [Ingrediente]`
  Dada una pizza devuelve la lista de ingredientes
- `tieneJamon :: Pizza -> Bool`
  Dice si una pizza tiene jamón
- `sacarJamon :: Pizza -> Pizza`
  Le saca los ingredientes que sean jamón a la pizza
- `armarPizza :: [Ingrediente] -> Pizza`
  Dada una lista de ingredientes construye una pizza
- `duplicarAceitunas :: Pizza -> Pizza`
  Recorre cada ingrediente y si es aceitunas duplica su cantidad

- `sacar :: [Ingrediente] -> Pizza -> Pizza`
  Saca los ingredientes de la pizza que se encuentren en la lista
- `cantJamon :: [Pizza] -> [(Int, Pizza)]`
  Dada una lista de pizzas devuelve un par donde la primera componente es la cantidad de jamón de la pizza que es la segunda componente.
- `mayorNAceitunas :: Int -> [Pizza] -> [Pizza]`
  Devuelve las pizzas con más de "n" aceitunas.

5. Tenemos los siguientes tipos de datos

```
data Objeto = Cacharro | Tesoro
data Camino = Fin | Cofre [Objeto] Camino | Nada Camino
```

Definir las siguientes funciones:

- `hayTesoro :: Camino -> Bool`
  Indica si hay un cofre con un tesoro en el camino.
- `pasosHastaTesoro :: Camino -> Int`
  Indica la cantidad de pasos que hay que recorrer hasta llegar al primer cofre con un tesoro. Si un cofre con un tesoro está al principio del camino, la cantidad de pasos a recorrer es 0.
- `hayTesoroEn :: Int -> Camino -> Bool`
  Indica si hay un tesoro en una cierta cantidad exacta de pasos. Por ejemplo, si el número de pasos es 5, indica si hay un tesoro en 5 pasos.
- `alMenosNTesoros :: Int -> Camino -> Bool`
  Indica si hay al menos "n" tesoros en el camino.
- `cantTesorosEntre :: Int -> Int -> Camino -> Int`
  Dado un rango de pasos, indica la cantidad de tesoros que hay en ese rango. Por ejemplo, si el rango es 3 y 5, indica la cantidad de tesoros que hay entre hacer 3 pasos y hacer 5. Están incluidos tanto 3 como 5 en el resultado.

6. Tenemos los siguientes tipos de datos

```
data ListaNoVacia a = Unit a | Cons a (ListaNoVacia a)
```

Definir las siguientes funciones:

- `length :: ListaNoVacia a -> Int`
  Retorna la longitud de la lista.
- `head :: ListaNoVacia a -> a`
  Devuelve el primer elemento de la lista. ¿Es una función parcial o total?
- `tail :: ListaNoVacia a -> ListaNoVacia a`
  Devuelve el resto de la lista sacando el primer elemento. Pensar bien qué pasa en caso de que la lista tenga un sólo elemento. ¿Es una función parcial o total?

- `minimo :: ListaNoVacia Int -> Int`
  Dada una lista retorna el mínimo de dicha lista. ¿Es una función parcial o total?

7. Tenemos los siguientes tipos de datos

```
data T a = A
         | B a
         | C a a
         | D (T a)
         | E a (T a)
```

Responder los siguientes items:

- ¿Cuáles son los casos base? ¿Cuáles los recursivos?
- Dar un ejemplo de un valor de tipo `T Int`.
- ¿Cuántas veces pueden aparecer `A`, `B` o `C`?

Definir las siguientes funciones:

- `size :: T a -> Int`
  Retorna la cantidad de elementos de una estructura T.
- `sum :: T Int -> Int`
  Retorna la suma de todos los elementos.
- `hayD :: T a -> Bool`
  Indica si existe al menos una aparición de `D` en la estructura.
- `cantE :: T a -> Int`
  Retorna la cantidad de `D` que existen en la estructura.
- `recolectarC :: T a -> (a, a)`
  Retorna los valores del constructor `C`. Notar que ese constructor puede no darse en la estructura.
- `toList :: T a -> [a]`
  Convierte la estructura a una lista, conservando el orden de los elementos.