# Parcial 1 - Algoritmos I Taller: Tema A

## Ejercicio 1

En las siguientes preguntas marque la respuesta correcta.

a) Si tengo una función con la siguiente declaración de tipos de la función `f`

`f :: Eq a => [a] -> (a -> b) -> b` puedo decir que:

1. Es una función polimórfica Paramétrica
2. Es una función polimórfica Ad hoc
3. Es una función recursiva.
4. Es un constructor.
5. Ninguna de las anteriores.

b) Si tengo una función con la siguiente declaración

```
f :: Eq a => a -> a -> a
f x y | x<y = x
      | otherwise = y
```

Puedo decir que:

1. Es una función polimórfica paramétrica
2. La declaración de tipos está mal, porque debería incluir otra clase de tipos.
3. La definición de la función es incorrecta, debería utilizar pattern matching.
4. Es un constructor.
5. Ninguna de las anteriores.

c) Dada la siguiente declaración de función en Haskell:

```
incrementar :: Int -> Maybe Int
incrementar maxBound = Nothing
incrementar x = Just (x + 1)
```

¿Cuál es el propósito de la función `incrementar` y cómo maneja el valor `maxBound`?

1) La función incrementar lanza un error cuando se le pasa Nothing.
2) La función incrementar siempre devuelve Just 1 independientemente de su entrada.
3) La función incrementar solo funciona con valores Nothing y no con Just.
4) Devuelve Just del valor que toma incrementado en 1 o Nothing si se le pasa `maxBound`.
5) Ninguna de las anteriores.

d) Dada la siguiente declaración de tipos en Haskell:

```
data Quizas a = Nada | Algo a
```

Puedo afirmar que :

1) Ese tipo está mal definido, debería haber utilizado el comando `type`.
2) El tipo está mal definido porque ambos constructores no toman parámetros.
3) El tipo `Quizas` tiene dos constructores, uno sin parámetros y el otro constructor con un parámetro.
4) No se puede definir un tipo de esa manera.
5) Ninguna de las anteriores.

# Ejercicio 2

Se va a representar el stock de un corralón de materiales de construcción, usando tipos en Haskell. Los materiales que tenemos en cuenta son: `Ladrillos, Viguetas, Cemento`. La idea es poder detallar para cada tipo de material, las características más importantes. En tal sentido identificamos las siguientes características de cada uno de los materiales a tener en cuenta:

`Ladrillo`

- `TipoLadrillo`, que es un tipo enumerado con las siguientes opciones: `Ceramico`, `Hormigon`, `Tradicional`
- `UsoDeLadrillo`, que es un tipo enumerado con las siguientes opciones: `Pared`, `Techo`
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Vigueta`

- `Largo`, que es un sinónimo de `Float` indicando el largo de la vigueta
- `MaterialViga`, que es un tipo enumerado con las siguientes opciones: `Hierro`, `Madera`.
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Cemento`

- `MarcaCemento`, que es un tipo enumerado con las siguientes opciones: `Minetti`, `LomaNegra`.
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

**a)** Definir el tipo `MaterialesConstruccion` que consta de los constructores `Ladrillo`, `Vigueta` y `Cemento`, constructores con parámetros descriptos arriba (Se deben definir también los tipos enumerados `TipoLadrillo`,`UsoDeLadrillo`, `MaterialViga`, `MarcaCemento`, y el sinónimo de tipos `Precio`). **Los tipos** `MaterialesConstruccion` **y** `MaterialViga` **<u>no deben</u> estar en la clase** `Eq`, ni en la clase `Ord`. Agregue la clase Show en los tipos que necesite.

**b)** Definir la función `ladrillosDeMenorPrecio` de la siguiente manera:

```
ladrillosDeMenorPrecio :: [MaterialesConstruccion] -> Int ->
[MaterialesConstruccion]
```

que dada una lista de `MaterialesConstruccion lm` y un valor `n` de `Precio`, devuelve la lista de `MaterialesConstruccion que son Ladrillo` en `lm y que tienen un precio menor o igual a n`.

**NOTA:** Dejar como comentario un ejemplo donde hayas probado la función `ladrillosDeMenorPrecio` con una lista con al menos 3 `MaterialesConstruccion`.

**c)** Definir igualdad para el tipo de `MaterialesConstruccion` : de tal manera que, dos valores de tipo `Ladrillo` son iguales sólo si tienen el mismo **tipo de ladrillo** y el mismo **precio**, dos `Vigueta` son iguales solo si tienen el mismo **largo** y el mismo **materialViga**, mientras que dos Cemento son iguales si tienen la misma **marca** . Como es de suponer `Ladrillo`, `Vigueta` y `Cemento` son distintos entre sí. Puede crear funciones auxiliares si necesita.

**NOTA:** Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

# Ejercicio 3

Queremos hacer un programa, para que el dueño de un vivero de árboles nativos pueda saber cuales de sus árboles están en condiciones de ser vendidos.

a) Definir un tipo recursivo `ArbolesNativos`, que permite guardar las características de cada árbol en esta época. El tipo `ArbolesNativos`, tendrá dos constructores:

1) `EvolucionDelArbol`, que tiene los siguientes parámetros:

- `String`, para el nombre del árbol
- `Estado` (tipo enumerado con el `estado` del actual del del arbol, `Seco`, `EnFlor`, `Verde`, `ConFrutos`)
- `Int` ( el tamaño de alto, entre 1 y 10)
- `Int` ( el tamaño de diametro, entre 1 y 10)
- `Int` ( apreciación general, entre 1 y 10)
- `ArbolesNativos`, recursión con el resto de los árboles.

2) `NoHayMasArboles`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar que se terminaron los árboles.

Las condiciones del arbol para poder ser vendido se describen a continuación según las diferentes características:

- Si el árbol tiene Estado ConFrutos puede ser vendido.
- `Si el Estado es EnFlor` debe tener más de 7 en el diámetro o en el alto, y tener en la apreciación general al menos un 8.
- Si el árbol tiene Estado Verde, debe tener al menos un 9 en diámetro, 9 en altura y 9 en apreciación general.

b) Programar la función `paraVender`, que toma como primer parámetro `árboles` del tipo **`ArbolesNativos,`** y como segundo parámetro el `nombre` del árbol de tipo **`String`** y retorna un valor de tipo **`Bool,`** indicando si el árbol con ese `nombre` **puede ser vendido.**.

```
paraVender :: ArbolesNativos -> String -> Bool
```

**NOTA:** Dejar como comentario un ejemplo donde hayas probado `paraVender` con un parámetro de tipo **`ArbolesNativos`** que tenga al menos 3 árboles.

c) Programar la función `devolverAltura` con la siguiente declaración:

```
devolverAltura :: ArbolesNativos -> String -> Maybe Int
```

que toma una variable `a` de tipo **`ArbolesNativos`**, y como segundo argumento un **`nombre`**, que identifica al árbol, y en caso que el árbol esté en `a` (con una altura **`h`**), retorna Just h y `Nothing` en caso contrario.

**NOTA:** Dejar como comentario un ejemplo donde hayas probado la función.