

Parcial 1 - Algoritmos I Taller: Tema B

Ejercicio 1

Van a representar canciones que se pueden escuchar en un servicio de *streaming* en modalidad gratuita como *Spotify*, *Youtube*, *SoundCloud*, etc. Para ello deben

a) Definir los tipos `Titulo` y `Artista` como sinónimos del tipo `String` y el tipo `Duracion` como sinónimo del tipo `Int`. Además se debe definir el tipo `Genero`, con constructores `Rock`, `Blues`, `Pop`, `Jazz` (todos sin parámetros).

El tipo `Genero` **no debe estar** en la clase `Eq`.

Por último deben definir el tipo `Cancion` que tiene constructores:

- Constructor `Tema` con parámetros:
 - El primero de tipo `Titulo`
 - El segundo del tipo `Artista`
 - El tercero del tipo `Genero`
 - El cuarto del tipo `Duracion` (la cantidad de segundos que dura la canción)
- Constructor `Publicidad` que tiene un único parámetro `Duracion` (cantidad de segundos que dura la molesta publicidad)

b) Definir mediante *pattern matching* la función

```
mismo_genero :: Genero -> Genero -> Bool
```

que dados dos valores `g1` y `g2` del tipo `Genero`, debe devolver `True` cuando `g1` y `g2` correspondan al mismo género musical (se construyen con el mismo constructor) y `False` en caso contrario. **Si se usan más de cinco casos, este apartado sumará menos puntaje.**

c) Definir la función

```
duracion_de :: Cancion -> Duracion
```

que dada una canción `c` devuelve la cantidad de segundos que dura su reproducción (ya sea un tema musical o una publicidad).

d) Incluir el tipo `Cancion` en la clase `Ord` de manera tal que una canción `c1` sea menor o igual que otra canción `c2` si la duración de `c1` es menor o igual que la duración de `c2`.

Ejercicio 2

Definir usando recursión y *pattern matching*:

```
solo_genero :: [Cancion] -> Genero -> [Titulo]
```

que dada una lista de canciones `cs` y un género `gi` devuelve los títulos de las canciones en `cs` que son temas musicales con género `gi`

IMPORTANTE: No se puede utilizar el operador `==` para hacer la comparación entre valores del tipo `Genero` puesto que el tipo no está en la clase `Eq`

Ejercicio 3

Basados en el tipo `ListaAsoc` del *Proyecto 2*, programar la función:

```
la_suma_mayores :: ListaAsoc a b -> b -> b
```

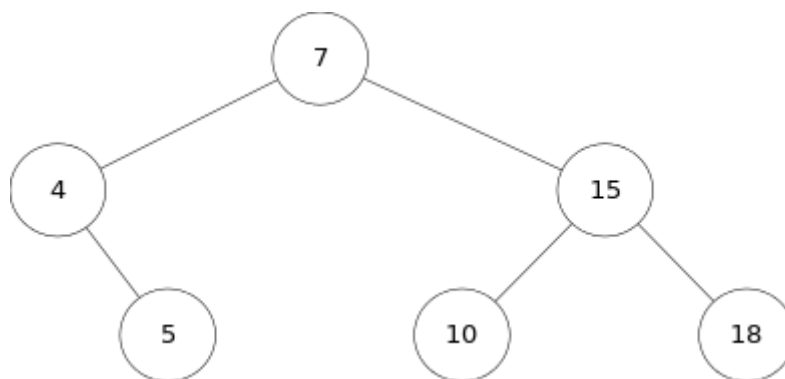
que dada una lista de asociaciones `la` y un dato `x` devuelve la suma de los datos de `la` que son mayores a `x`. Completar el tipado de la función para incluir los *type classes* necesarios para programarla.

Ejercicio 4*

a) Programar la función

```
a_listar :: Arbol a -> [a]
```

que dado un árbol `as` devuelve una lista con los elementos de `as`. En la lista resultante el elemento del padre siempre debe estar antes que los elementos de sus hijos, Por ejemplo



la lista debe ser `[7, 4, 5, 15, 10, 18]`

b) Inventar un ejemplo de uso de la función creando un árbol con al menos 3 elementos

c) Escribir el resultado de la función aplicada al ejemplo del inciso b)

Ejercicio 1

```
-- Ejercicio 1)
-- a)
type Titulo = String
type Artista = String
type Duracion = Int

data Genero = Rock | Blues | Pop | Jazz
data Cancion = Tema Titulo Artista Genero Duracion | Publicidad Duracion

-- b)
mismo_genero :: Genero -> Genero -> Bool
mismo_genero Rock Rock = True
mismo_genero Blues Blues = True
mismo_genero Pop Pop = True
mismo_genero Jazz Jazz = True
mismo_genero a b = False

-- c)
duracion_de :: Cancion -> Duracion
duracion_de (Publicidad c) = c
duracion_de (Tema a b c d) = d

-- d)
-- Necesito instanciar Eq para poder hacer comparacion utilizando el operador "<=".
instance Eq Cancion where
    c1 == c2 = duracion_de c1 == duracion_de c2
instance Ord Cancion where
    c1 <= c2 = duracion_de c1 <= duracion_de c2
```

Ejercicio 2

[illegible]

Ejercicio 3

```
-- Ejercicio 3)
data ListaAsoc a b = Vacia | Nodo a b (ListaAsoc a b)

-- b debe ser de tipo Num y Ord para poder utilizar los operadores "+" y ">".
la_suma_mayores :: (Num b, Ord b) => ListaAsoc a b -> b -> b
la_suma_mayores Vacia x = 0
la_suma_mayores (Nodo a b c) x | b > x = b + la_suma_mayores c x
                                | otherwise = la_suma_mayores c x
```

Muy bien!

Ejercicio 4

```
-- Ejercicio 4)
data Arbol a = Hoja | Rama (Arbol a) a (Arbol a)

a_listar :: Arbol a -> [a]
a_listar Hoja = []
a_listar (Rama a b c) = b:(a_listar a ++ a_listar c)

-- b) a_listar (Rama (Rama Hoja 5 Hoja) 8 (Rama Hoja 7 (Rama Hoja 9 Hoja)))
-- c) [8,5,7,9]
```

Muy bien!