Programación Funcional

Ejercicios de Práctica Nro.5

Tipos algebraicos

Aclaraciones:

- Los ejercicios siguen un orden de complejidad creciente, y cada uno puede servir a los siguientes. No se recomienda saltear ejercicios sin consultar antes a un docente.
- Recordar que se pueden aprovechar en todo momento las funciones ya definidas, tanto las de esta misma práctica como las de prácticas anteriores.
- Probar todas las implementaciones, al menos en una consola interactiva.
- Es sumamente aconsejable resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en clase, dado que los exámenes de la materia evalúan principalmente este aspecto. Para utilizando formas alternativas al resolver los ejercicios consultar a los docentes.

Ejercicio 1) Dadas las siguientes definiciones

```
data Gusto = Chocolate | DulceDeLeche | Frutilla | Sambayon
data Helado = Vasito Gusto
            | Cucurucho Gusto Gusto
            | Pote Gusto Gusto Gusto
chocoHelate consH = consH Chocolate
```

determinar el tipo de las siguientes expresiones:

a. `Vasito`
b. `Chocolate`
c. `Cucurucho`
d. `Sambayon`
e. `Pote`
f. `chocoHelate`
g. `chocoHelate Vasito`
h. `chocoHelate Cucurucho`
i. `chocoHelate (Cucurucho Sambayon)`
j. `chocoHelate (chocoHelate Cucurucho)`
k. `chocoHelate (Vasito DulceDeLeche)`
l. `chocoHelate Pote`
m. `chocoHelate (chocoHelate (Pote Frutilla))`

Ejercicio 2) Dado el siguiente tipo que pretende representar dígitos binarios

```
data DigBin = O | I
```

definir las siguientes funciones:

a. `dbAsInt :: DigBin -> Int`, que dado un símbolo que representa un dígito binario lo transforma en su significado como número.

b. **`dbAsBool :: DigBin -> Bool`**, que dado un símbolo que representa un dígito binario lo transforma en su significado como booleano.
c. **`dbOfBool :: Bool -> DigBin`**, que dado un booleano lo transforma en el símbolo que representa a ese booleano.
d. **`negDB :: DigBin -> DigBin`**, que dado un dígito binario lo transforma en el otro.

Ejercicio 3) Dado el siguiente tipo que pretende representar dígitos decimales

```
data DigDec = D0 | D1 | D2 | D3 | D4
            | D5 | D6 | D7 | D8 | D9
```

definir las siguientes funciones:

a. **`ddAsInt :: DigDec -> Int`**, que dado un símbolo que representa un dígito decimal lo transforma en su significado como número.
b. **`ddOfInt :: Int -> DigDec`**, que dado un número entre 0 y 9 lo transforma en el símbolo que representa a ese dígito.
c. **`nextDD :: DigDec -> DigDec`**, que dado un dígito decimal lo transforma en el siguiente según el orden circular dado en la definición.
d. **`prevDD :: DigDec -> DigDec`**, que dado un dígito decimal lo transforma en el anterior según el orden circular dado en la definición.

Ejercicio 4) Dado el siguiente tipo que representa medidas en un software de dibujo como LibreOffice Draw.

```
data Medida = Mm   Float | Cm   Float
            | Inch Float | Foot Float
```

y la siguiente tabla de conversión

	Mm	Cm	Inch	Foot
Mm	1	0,1	0,039	0,003
Cm	10	1	0,394	0,033
Inch	25,4	2,54	1	0,083
Foot	304,8	30,48	12	1

escribir las siguientes funciones:

a. **`asMm :: Medida -> Medida`**, que dada una medida cualquiera la transforma en una medida en milímetros que aproxima la dada según la conversión establecida.
b. **`asCm :: Medida -> Medida`**, que dada una medida cualquiera la transforma en una medida en centímetros que aproxima la dada según la conversión establecida.
c. **`asInch :: Medida -> Medida`**, que dada una medida cualquiera la transforma en una medida en pulgadas que aproxima la dada según la conversión establecida.

d. **`asFoot :: Medida -> Medida`**, que dada una medida cualquiera la transforma en una medida en pies que aproxima la dada según la conversión establecida.

Ejercicio 5) Dadas las siguientes definiciones:

```
data Shape = Circle Float | Rect Float Float
construyeShNormal :: (Float -> Shape) -> Shape
construyeShNormal c = c 1.0
```

Determinar el tipo de las siguientes expresiones:

a. **`uncurry Rect`**
b. **`construyeShNormal (flip Rect 5.0)`**
c. **`compose (uncurry Rect) swap`**
d. **`uncurry Cucurucho`**
e. **`uncurry Rect swap`**
f. **`compose uncurry Pote`**
g. **`compose Just`**
h. **`compose uncurry (Pote Chocolate)`**

Ejercicio 6) Para cada una de las expresiones del ejercicio anterior que denoten funciones, construir una expresión aplicándola.

Ejercicio 7) Dado el tipo **`Set`** definido en la clase teórica

```
data Set a = S (a -> Bool)
```

definir las siguientes funciones sobre él:

a. **`belongs :: Set a -> a -> Bool`**, que dado un conjunto, describe la función que indica si un elemento dado pertenece a ese conjunto.
b. **`empty :: Set a`**, que describe el conjunto vacío.
c. **`singleton :: a -> Set a`**, que dado un elemento describe un conjunto que contiene a ese único elemento.
d. **`union :: Set a -> Set a -> Set a`**, que dados dos conjuntos, describe al conjunto que resulta de la unión de ambos.
e. **`intersection :: Set a -> Set a -> Set a`**, que dados dos conjuntos, describe al conjunto que resulta de la intersección de ambos.

Ejercicio 8) Dados los siguientes tipos que modelan los posibles resultados de computar con excepciones

```
data MayFail a = Raise Exception | Ok a
data Exception = DivByZero | NotFound | NullPointer
               | Other String
type ExHandler a = Exception -> a
```

definir la función

```
tryCatch :: MayFail a -> (a -> b) -> ExHandler b -> b
```

que dada una computación que puede fallar, una función que indica cómo continuar si no falla, y un manejador de los casos de falla, expresa la computación completa.

Un ejemplo que utiliza esta función sería el siguiente:

```
sueldoGUIE :: Nombre -> [Empleado] -> GUI Int
sueldoGUIE nombre empleados =
    tryCatch (lookupE nombre empleados)
             mostrarInt
             (\e -> case e of
                      NotFound -> ventanaError msgNotEmployee
                      _        -> error msgUnexpected)
    where msgNotEmployee = "No es empleado de la empresa"
          msgUnexpected  = "Error inesperado"
```

sabiendo que

```
mostrarInt :: Int -> GUI Int
ventanaError :: String -> GUI a
lookupE :: Nombre -> [Empleado] -> MayFail Int
```