Práctico 3 de Programación Funcional.

UdelaR/FCien/CMat

xx al yy de septiembre de 2016.

1. ¿Cuáles son los valores de las siguientes expresiones?

   ```
   show (show 42)
   show 42 ++ show 42
   show (\n)
   ```

2. Supongamos que representamos fechas con ternas de enteros (d, m, a) donde d representa al día, m toma valores entre 1 y 12 y representa al mes y a representa al año. Declarar

   data `Date = MkDate Int Int Int`

   Declarar `Date` como instancia de la clase `Show`, de modo que la fecha asociada a la terna $(10, 9, 2013)$ se muestre como `10 de septiembre de 2013`.

   Declarar `Date` como instancia de la clase `Show`, para mostrar la fecha en inglés del siguiente modo:

   A la fecha asoc. con $(10, 9, 2013)$ `-->  10th september 2013`
   A la fecha asoc. con $(31, 12, 2013)$ `-->  31st december 2013`

   etcétera.

3. Definir una función `copy ::  Int -> String -> String` que tiene por argumentos un entero `n::Int` y un `xs::String` y, si `n > 0` devuelve el `String` que contiene `n` copias de `xs` concatenadas. ¿Qué hace la función definida en el caso en que `n` es negativo? Elabore diferentes versiones que traten este caso de otra forma.

4. Este ejercicio tiene por objeto programar una calculadora que realice operaciones combinadas de `+, -, *, /`. Dado que las operaciones ya están implementadas en Haskell, el ejercicio gira en torno al problema de transformar secuencias de caracteres a expresiones aritméticas y viceversa (i.e.: hacer un *parsing*). Evitaremos usar un orden de precedencia, poniendo en su lugar paréntesis explícitos en torno a una expresión que contenga un símbolo de operación en su interior. En todas las definiciones de funciones se reflexionará acerca de la forma más conveniente de tratar los argumentos incorrectos.

(a) Defina una función `stringToInt :: String -> Int` que lea una cadena de caracteres que representa un entero –en notación decimal– y devuelva el entero representado. Así:

`stringToInt ‘‘325’’` reduce a `325` de tipo `Int`

(b) Defina una función

`expression :: String -> (Int -> Int -> Int, Int, Int)`

que transforme una expresión de la forma `<entero1> <op> <entero2>` en la terna `((<op>), <entero1>, <entero2>)`. Por ejemplo:

`‘‘325 + 17’’` reduce a la terna `((+), 325, 17)`

Se tendrá la precaución de que la función calcule bien independientemente de la cantidad de espacios que separen a la operación de sus argumentos.

(c) Defina una función `eval::(Int -> Int -> Int, Int, Int) -> Int` que evalúe la terna argumento. Esto es:

`eval ((+), 3, 7)` reduce a `10` de tipo `Int`

(d) Defina una función `calculator :: String -> String` que dado un string que represente una de las cuatro operaciones `+, -, *, /` ubicada entre dos expresiones `a, b :: Int` –e.g: `‘‘a + b’’`– devuelva el string que representa al resultado. Las expresiones `a, b` pueden ser enteros o más expresiones que involucren las operaciones `+, -, *, /` escritas entre paréntesis. Así la calculadora deberá poder resolver expresiones de la forma `‘‘(3 + 7) * (5 - 9)’’`.

5. El propósito de este ejercicio es definir funciones que traduzcan expresiones decimales que representan naturales a numeración romana y viceversa. La numeración romana satisface las siguientes condiciones:

   - Los números romanos se escriben combinando letras `I, V, X, L, C, D, M`; las cuales representan respectivamente los naturales $1, 5, 10, 50, 100, 500, 1000$.
   - Los que representan enteros de la forma 10^i se iteran, obteniendo así sus múltiplos: `II` y `III` representan respectivamente 2 y 3; `XXX` representa al 30; `MMCCCXX` representa al $2000 + 300 + 20 = 2320$.
   - Los que representan enteros de la forma 5×10^i se usan a lo sumo una vez. Así `VII` representa al 7, `DCCCLXVII` representa al 867, las expresiones como `VV, LLL, DD` son incorrectas.
   - La iteración de cuatro símbolos consecutivos se debe evitar reemplazándolo por la substracción de una unidad al símbolo siguiente (principio de sustracción). Así el 4 se denota `IV` en lugar de `IIII`, el 900 se denota `CM` en lugar de `DCCCC`.
   - Los símbolos se escriben de mayor a menor, excepto cuando se aplica el principio de sustracción.

En consecuencia, una expresión bien formada en numeración romana tiene a lo sumo una vez los símbolos `V, L, D` y hasta tres veces consecutivas los restantes. Se pide:

(a) Definir el tipo **data** `RomanDigit = I|V|X|L|C|D|M|Error`. Derivarla automáticamente como miembro de las clases **Ord**, **Eq**, **Show**. Definir `Roman` como sinónimo de `[RomanDigit]`. Obs: `Error` representa a todos los caracteres diferentes de `I, V, X, L, C, D, M`.

(b) Definir `isRoman::Roman -> Bool` que devuelve **True** o **False** según si el argumento es un número romano o no. (Sug.: observar que el análisis de la expresión es recursivo en los miembros de `Roman`, pero se deben considerar dos símbolos consecutivos para determinar si la expresión es correcta).

(c) Definir una función `romanToString::Roman -> String` tal que, si un miembro de `Roman` representa un número romano, transforme dicho elemento en la cadena de caracteres que lo representa. Definir una función `stringToRoman` que en cadenas de caracteres que representan números romanos sea inversa de `romanToString`.

(d) Definir una función `digitToInt::RomanDigit -> Int` que transforma un miembro de `RomanDigit` en el entero que representa. Definir una función `romanToInt::Roman -> Int` que transforme un número romano en el entero que representa (sug.: sumar los enteros representados por los `RomanDigit` de la lista, afectados del signo correcto).

(e) Definir una función `intToRoman::Int -> Roman` que, dado un entero positivo, lo transforme a numeración romana (sug: transformar dígito a dígito la expresión decimal del número entero).

(f) definir dos funciones

```
stringRomanToInt  ::  String -> Int
intToRomanString  ::  Int -> String
```

que hagan las conversiones de strings que representen números romanos a enteros y viceversa. Resolver "prolijamente" los casos en los que el string no represente un número romano.