Práctica 2

Programación Funcional, UNQ

Tipos, currificación y alto orden

Aclaraciones:

- *Los ejercicios fueron pensados para ser resueltos en el orden en que son presentados. No se saltee ejercicios sin consultar antes a un docente.*
- *Recuerde que puede aprovechar en todo momento las funciones que ha definido, tanto las de esta misma práctica como las de prácticas anteriores.*
- *Pruebe todas sus 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 evaluarán principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*
- *No dude en manifestar observaciones y críticas sobre los ejercicios de esta práctica, que con gusto serán recibidas por los docentes.*
- *Los ejercicios del anexo pueden obviarse, pero recuerde que aportan una compresión más profunda sobre los temas que aborda esta práctica. Considere resolverlos si se encuentra practicando para una instancia de evaluación y ya resolvió todos los anteriores.*

Muchas de las definiciones presentadas a continuación pueden ser encontradas en los módulos `Prelude` y `Data.List`. Por lo que algunas se cargan automáticamente al iniciar un entorno. Considere cambiar el nombre de las funciones definidas si encuentra un problema con el interprete al testear su código.

1. Tipos

I. Dar tipo a las siguientes expresiones:

a) `True`
b) `[2]`
c) `Maybe ["Jorge"]`
d) `Nothing`
e) `[]`
f) `let x = [] in x ++ x`
g) `let f x = f x in f []`
h) `undefined` (o `bottom`)

II. Dar ejemplos de expresiones que posean los siguientes tipos:

a) `Bool`
b) `(Int, Int)`
c) `Int -> Int -> Int`
d) `a -> a`
e) `a`
f) `a -> b`

2. Typeclasses

Describa el propósito de las siguientes typeclasses:

a) Eq
b) Ord
c) Enum
d) Bounded
e) Num
f) Show
g) Read

3. Funciones de alto orden

Definir las siguientes funciones:

a) `const :: a -> b -> a`
b) `alph :: a -> b -> b`
c) `apply :: (a -> b) -> a -> b` (aka `($)`)
d) `twice :: a -> b -> b`
e) `flip :: (a -> b -> c) -> b -> a -> c`
f) `(.) :: (b -> c) -> (a -> b) -> (a -> c)`
g) `curry :: ((a,b) -> c) -> a -> b -> c`
h) `uncurry :: (a -> b -> c) -> (a,b) -> c`

4. Recorridos

Definir las siguientes funciones de alto orden sobre listas:

a) `map :: (a -> b) -> [a] -> [b]`
b) `filter :: (a -> Bool) -> [a] -> [a]`
c) `all :: (a -> Bool) -> [a] -> Bool`
d) `any :: (a -> Bool) -> [a] -> Bool`
e) `takeWhile :: (a -> Bool) -> [a] -> [a]`
f) `concatMap :: (a -> [b]) -> [a] -> [b]`
g) `partition :: (a -> Bool) -> [a] -> ([a], [a])`
h) `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`

5. Composición

Sea `h x y = f (g x y)`. Decidir cuáles de las siguientes afirmaciones son verdaderas:

a) `h = f . g`
b) `h x = f . (g x)`
c) `h x y = (f . g) x y`

6. Lógica de cortocircuito

Dada la siguiente definición para los Booleanos:

```
data Bool = True | False
```

Considerando el orden de evaluación Lazy, definir las siguientes funciones de forma tal que sólo se evalué el primer parámetro cuando sea posible (eg. `or True bottom` debería reducir a True):

a) `and :: Bool -> Bool -> Bool`

b) `or :: Bool -> Bool -> Bool`

c) `ifThenElse :: Bool -> a -> a -> a`

¿Es posible dar una definición para estas funciones que tenga el beneficio de cortocircuito aún con un orden de evaluación Eager?