Trabajo Práctico # 1

Programación Funcional, Universidad Nacional de Quilmes

12 de marzo de 2018

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 evaluación principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*

1. Funciones básicas

1. `id :: a -> a`
2. `const :: a -> b -> a`
3. `fst :: (a,b) -> a`
4. `snd :: (a,b) -> b`
5. `swap :: (a,b) -> (b,a)`
6. `head :: [a] -> a`
7. `tail :: [a] -> [a]`
8. `sum, product :: Num a => [a] -> a`
9. `elem, notElem :: Eq a => a -> [a] -> Bool`
10. `and, or :: [Bool] -> Bool`
11. `last :: [a] -> a`
12. `init :: [a] -> [a]`
13. `subset :: Eq a => [a] -> [a] -> Bool`
    Ejemplos:
    ```
    subset [1,2,3] [1,4,2,5,3,1] = True
    subset [1,2,3] [1,4,2,5] = False
    subset [1,1,1] [1] = True
    ```
14. `(++) :: [a] -> [a] -> [a]`
15. `concat :: [[a]] -> [a]`
16. `(!!) :: [a] -> Int -> a`

17. `take :: Int -> [a] -> [a]`
18. `drop  :: Int -> [a] -> [a]`
19. `zip :: [a] -> [b] -> [(a,b)]`
20. `splitAt :: Int -> [a] -> ([a], [a])`
21. `maximum, minimum :: Ord a => [a] -> a`
22. `lookup :: Eq a => a -> [(a,b)] -> Maybe b`
23. `unzip :: [(a,b)] -> ([a],[b])`
24. `tails :: [a] -> [[a]]`
25. `replicate :: Int -> a -> [a]`
    Ejemplo: `replicate 5 1 = [1,1,1,1,1]`
26. `repeat :: a -> [a]`
    Ejemplo: `repeat 1 = [1..]`
27. `cycle :: [a] -> [a]`
    Ejemplo: `cycle [1,2,3] == [1,2,3,1,2,3,...]`
28. `nats :: [Int]`
    Retorna todos los números naturales: `[1,2,3,4,...]`
29. `agrupar :: Eq a => [a] -> [[a]]`
    Ejemplo: `agrupar [1,1,2,2,1,3,3,3] == [[1,1],[2,2],[1],[3,3,3]]`

2. Tipo a expresiones

1. Dar tipo a las siguientes expresiones y funciones

   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)
   ```
   data Either a b = Left a | Right b
   x = Left True
   y = Right (Left [])
   z = Right (Left [Right []])
   ```

   i) `(:)`

   j) `Maybe`

   k) `Right`

   l) `(1:)`

   m) `error "ups"`

   n) `error`

   ñ) `undefined`

o) `undefined undefined`

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

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

3. Patterns

Indicar si los siguientes patterns son correctos

1. `(x, y)`
2. `(1, y)`
3. `(n+1)`
4. `('a',('a',b))`
5. `(a,(a,b))`
6. `([]:[4])`
7. `(x:y:[])`
8. `[x]`
9. `([]:[])`

4. Redex

Escribir los redex de las siguientes funciones

1. `sum [1,2,3]`
2. `length [1,2,3]`
3. `take 3 [1,2,3]`
4. `[1,2,3] !! 2`
5. `lookup 2 [1,2,3]`
6. `factorial 5`
7. `not (elem 2 [1,2,3])`

5. Terminación

Indicar qué programas terminan

1. `let nats = [1..] in nats`
2. `take 5 [1..]`
3. `let appendedNats = [1..] ++ [1..] in take 5 appendedNats`
4. `let x = x in x`
5. `undefined`
6. `undefined undefined`

6. Typeclasses

6.1. Prelude

Describa el propósito de las siguientes typeclasses:

- Enum
- Ord
- Eq
- Bounded
- Show
- Read
- Num

6.2. Tipado

Indicar el tipo y el resultado de las siguientes expresiones:

1. `5`
2. `2.0`
3. `(5, 2.0)`
4. `5 + 2.0`
5. `minBound`
6. `minBound && True`
7. `succ`
8. `succ False`
9. `succ True`
10. `succ []`
11. `succ 'a'`

12. `read "5"`
13. `read "5" :: Int`
14. `let xs = [1,2,3] in xs`
15. Dado `xs = [1,2,3]` en un archivo, `xs`

6.3. Implementación y Uso

Usando typeclasses escribir programas que cumplan los siguientes propósitos

1. Implementar `Eq` para `data Persona = P Int String`, donde `Int` es el dni de la persona.
2. Implementar `Ord` para la misma estructura, donde las personas se ordenan según el orden lexicográfico de sus nombres.
3. Enumarar todas las letras de la `a` a la `z` en minúscula
4. Implementar todos los typeclasses del `Prelude` que tengan sentido para los días de la semana, y escribir una función que los devuelva a todos (sin usar los constructores).
5. Implementar `Enum`, `Eq` y `Ord` para `data Nat = Z | S Nat`
6. Implementar `Num` para

```
data Entero = E Bool Nat
data Nat = Z | S Nat
```