Trabajo Práctico: Programación Funcional

Paradigmas y Lenguajes de Programación - Trelew

Lic. Pablo Toledo Margalef Lic. Lautaro Pecile

Año 2023

Ejercitación

Ejercicio 1

Definir la función `vabs` (`abs` de Prelude), que retorne el valor absoluto de un entero dado.

Ejercicio 2

Analizar la función `gcd` de Prelude y determinar su objetivo y los conceptos que introduce, sabiendo que:

- `abs` es la función valor absoluto
- `rem` es la función resto
- `Integral` es un *typeclass* que engloba a los enteros de máquina y los de precisión arbitraria
- `error` es una función que muestra un mensaje de error por pantalla

```
gcd :: Integral a => a -> a -> a
gcd 0 0 = error "Prelude.gcd: gcd 0 0 is undefined"
gcd x y = gcd' (abs x) (abs y)
            where   gcd' x 0 = x
                    gcd' x y = gcd' y (x `rem` y)
```

Ejercicio 3

Definir el prototipo que Haskell infiere de la siguiente función y analizar el objetivo y prototipo de la función `(++)`:

```
hola "Juan" = "Hola, qué dia ¿no?"
hola nombre = "Buenas " ++ nombre ++ " ¿todo bien?"
```

Ejercicio 4

Definir las funciones `fact1`, `fact2`, `fact3`, que retornen el factorial de un entero dado. Una versión debe aplicar *guardas*, otra *pattern matching*, y otra definida como lo deseen. Verificar el prototipo de cada función inferido por Haskell.

Ejercicio 5

Definir la función `vmax`/`vmin` (funciones `max`/`min` de Prelude) tal que dados 2 enteros retorne el mayor/menor. Analizar las modificaciones a introducir si se desea obtener el mayor/menor de 2 caracteres. Ídem para 2 strings. Definir en forma genérica las funciones `vmax` y `vmin` tal que retornen el mayor/menor de 2 valores dados.

Ejemplos

```
Prelude> vmax "hola" "chau"     -- "hola"
Prelude> vmin 'b' 'z'           -- 'b'
Prelude> vmin 17.5 (-20.8)      -- -20.8
```

Ejercicio 6

Definir la función `mayor/menor` tal que dada una lista retorne el mayor/menor de sus elementos.

Ejercicio 7

Dada la lista `[("juan",7), ("ana",9), ("luis",6), ("maria",8)]`, analizar cómo se comportan las funciones `length`, `head`, `tail`, `last`, `take n`, `drop n`, `sum`, definidas en `Prelude`.

Ejercicio 8

Definir las funciones `esta/noEsta` tal que dado un elemento y una lista verifican que el elemento pertenece/no pertenece a la lista.

Ejercicio 9

Aplicando la siguiente función, definir las funciones `cuadrado`, `cubo` y `cuarta`, tales que dado un entero retornan la correspondiente potencia del mismo:

```
pot :: Integer -> Integer -> Integer
pot n x = x ^ n
```

Ejercicio 10

Definir la función `collect` (`map` de Prelude) tal que dada una función y una lista aplica la función a cada elemento de la lista y retorna la lista de resultados.

Ejemplos:

```
Prelude> collect (length) ["hola", "gente"]      -- [4,5]
Prelude> collect (*8) [2,4,3,5]                  -- [16,32,24,40]
```

Ejercicio 11

Definir la función `select` (`filter` de Prelude) tal que dado un criterio de selección y una lista retorna los elementos de la lista que verifican el criterio.

Ejemplos:

```
Prelude>  select even [2,7,-1,0,4]           -- [2,0,4]
Prelude> select (>'d') "probando letras"     -- "pronoletrs"
```

Ejercicio 12

Analizar la función `(.)` definida en Prelude, definir su objetivo y determinar si los ejemplos de aplicación son correctos o no:

```
(.)  :: (b -> c) -> (a -> b) -> (a -> c)
(f . g) x = f (g x)

-- Ejemplos
g1 x = (not . esta) x
g2 x = not . esta x
g3 x xs = not . esta x xs
g4 = cuadrado . cuadrado
g5 x = (cubo . cubo) x
g6 = sum . map length
g7 = filter even . map length
g8 = (+1) . flip div 2
g9 x = length . (++ x)
g10 x y = length . (++ x)
```

Ejercicio 13

a) Definir la función `sin_consonantes` tal que dada una lista de palabras retorne la lista con las palabras sin sus consonantes.

Ejemplo:

```
Prelude> f ["Luis", "Pedro", "Marta"]   -- ["ui", "eo", "aa"]
```

b) Definir la función `cuantas_consonantes` tal que dada una lista de palabras retorne la lista con la cantidad de consonantes de cada palabra.

Ejemplo:

```
Prelude> g ["Luis", "Pedro", "Marta"]   -- [4, 5, 5]
```

c) Definir la función `h` tal que dada una lista de palabras y una función pueda comportarse como las anteriores.

Ejemplos:

```
Prelude> h f ["Luis", "Pedro", "Marta"]    -- ["ui", "eo", "aa"]
Prelude> h g ["Luis", "Pedro", "Marta"]    -- [4, 5, 5]
```

Ejercicio 14

Dadas las siguientes funciones, determinar su comportamiento ante las invocaciones y el concepto que se aplica:

```
ignoroArg1 x = "No me importa"
ignoroArg2 x = x
```

```
Prelude> ignoroArg1 (1/0)
Prelude> ignoroArg2 (1/0)
```

Ejercicio 15

Analizar las siguientes funciones, definir su objetivo y determinar los conceptos que se aplican:

```
queHace :: [Int] -> Int
queHace [] = error "lista vacía!"
queHace [x] = x
queHace (x:y:ys) = queHace (queHaceAux1 x y : ys)

queHaceAux1::Int->Int->Int
queHaceAux1 x y     |x>y        = queHaceAux2 y [x,x+x ..]
                    |otherwise  = queHaceAux2 x [y,y+y .. ]

queHaceAux2 :: Int -> [Int] -> Int
queHaceAux2 x (y:ys)    |mod y x == 0   = y
                        |otherwise  = queHaceAux2 x ys
```

Ejercicio 16

Definir la función `mcd` tal que dados 2 enteros retorne el máximo común divisor.

Ejercicio 17

Definir una función `listanios`, y todas sus auxiliares, que permitan generar una lista de números que representan años pares, capicúas y no bisiestos, a partir de un año dado como parámetro, siempre positivo. Recordar que un año bisiesto cumple las siguientes condiciones: es divisible por 4 y, si es divisible por 100, también debe serlo por 400.

Ejemplo:

```
Prelude> listanios 800 -- [818, 838, 858, 878, 898, 2002, 2222, . . . ]
```

Nota: Se dispone de la función `revers` tal que dado un entero devuelve la reversa del entero

Ejemplo:

```
Prelude> revers 235  --   532

revers :: Int -> Int
revers n = reversa2 n 0

reversa2 :: Int -> Int -> Int
reversa2 n x    | n < 10    = n + (x * 10)
                | otherwise     = reversa2 (div n 10)  ((mod n 10) + (x * 10))
```

Ejercicio 18

Definir la función `extremo` tal que dado un criterio de selección y una lista retorna el elemento de la lista que verifica el criterio.

Ejemplos:

```
Prelude> extremo (>) [7, -3, 2, 5]                  -- 7
Prelude> extremo (masCorta)  [“hola”, “gente”]      -- "hola”
```

Ejercicio 19

Definir la función `ordenar` tal que dado un criterio y una lista retorna la lista ordenada según el criterio.

Ejemplos:

```
Prelude> ordenar (>) [(segunda ("vvvv", 1)), (segunda ("vv", 2))]
-- [2,1]

Prelude> ordenar (<) [(primera ("vvvv", 1)), (primera ("vv", 2))]
-- ["vv","vvvv"]

Prelude> ordenar (>) [4, 6, 5, 1, 2, 3]
-- [6,5,4,3,2,1]

Prelude> ordenar (<) (map (longitud.primera) [("hola", 1), ("b", 2)])
-- [1,4]
```

Ejercicio 20

Definir la función `aplica` tal que dadas una lista de funciones, una lista de elementos y una lista de valores resultado, permita obtener una lista de tuplas `(elemento, resultado, posiciónDeLaPrimeraFunciónQueProduceElResultado)`

Ejemplos:

```
Prelude> aplica [cuadrado,cubo,(+1)] [1,2,3] [1,3,9]
--   [(1,1,1),(2,3,3),(3,9,1)]
Prelude> aplica [primero,ultimo] ["hola","linda","chau"] ['h','l','u']
-- [("hola",'h',1),("linda",'l',1),("chau",'u',2)]
```

Ejercicio 21

Definir la función `igualValor` que reciba como parámetro una lista de funciones, y devuelva el menor entero positivo para el cual todas las funciones retornan el mismo resultado.

Ejemplo:

```
Prelude>  igualValor [((+5).(+1)),cuadrado]    --  3
```

Ejercicio 22

Desarrollar la función `izquierda/derecha` que reciba como parámetro una lista y desplace los elementos de la misma hacia la izquierda/derecha una posición. Desarrollar la función `desplaza`, aplicando las funciones anteriores, que reciba como parámetro una función, un entero y una lista, y genere el desplazamiento de los n elementos de la misma en el sentido indicado por la función.

Ejemplos:

```
Prelude> izquierda  "Paradigmas"            -- "aradigmasP"
Prelude> derecha  "Paradigmas"              -- "sParadigma"
Prelude> desplaza derecha 3 "Paradigmas"    -- "masParadig"
Prelude> desplaza izquierda 4 "Paradigmas"  -- "digmasPara"
```

Ejercicio 23

Definir la función `soloUnSaltarin` tal que reciba como parámetros dos listas comparables y verifique que la segunda lista presenta a lo sumo un elemento "saltarín", que cambió su posición original respecto de la primera lista, avanzando sobre el resto de la lista.

Ejemplos:

```
Prelude> soloUnSaltarin "causalidad" "casualidad"   -- True
Prelude> soloUnSaltarin  "amor" "mora"              -- True
Prelude> soloUnSaltarin  [1,2,3,4] [1,3,4]          -- False
```

Ejercicio 24

Definir una función `verifIncremento` tal que dadas dos listas de listas, verifica que las sublistas tomadas de a pares presentan n elementos diferentes, con n igual a la cantidad de elementos diferentes del par anterior incrementada en 1.

Ejemplos:

```
Prelude> verifIncremento ["este","si"] ["esta","no"]               -- True
Prelude> verifIncremento ["este","si"] ["este","si?"]              -- True
Prelude> verifIncremento [[1,1],[2,2,2]] [[1,2],[2,2,2,4]]         -- False
Prelude> verifIncremento ["este","ejemplo","no"] ["este","ejemp."] -- False
```

Ejercicio 25

Dada la función:

```
f [] x = x
f (x:xs) y = f xs (x y)
```

a) Definir su prototipo y objetivo.

b) Aplicando f definir las siguientes funciones y todas las auxiliares necesarias:

1. Codificar la función `verifican`, tal que dada una lista de valores y una lista de funciones, retorna aquellos valores que luego de aplicar las funciones devuelven un resultado que se encuentra en la lista de valores.

Ejemplos:

```
Prelude> verifican [1,2,-1,0,9] [pot 2, suma 3, resta 2]
-- [1,-1,0]
Prelude> verifican ["a","hola","lo"] [reversa, aLaCabeza 'h', aLaCola 'a']
-- ["lo"]
```

2. Definir la función `saldoFinal`, tal que dado el saldo inicial de una cuenta bancaria, y una serie de movimientos realizados sobre la misma, se desea obtener el saldo final. Los movimientos se indican de acuerdo al código que los representa, según la siguiente tabla:

Código	Operación
A	Acreditación de un cheque depositado
B	Débito de un cheque emitido
D	Depósito en efectivo
E	Extracción en efectivo
G	Gastos bancarios

Ejemplo:

```
Prelude> saldoFinal [   ('D', 138),
                        ('E', 180),
                        ('B', 62.5),
                        ('G', 3.6),
                        ('E', 300),
                        ('A', 120)  ]
                       200
-- -88.1
```

Ejercicio 26

a) Definir una función `esCerrada`, y todas las auxiliares necesarias, tal que dada una función y una lista de valores, verifica que la función aplicada a cada valor produce un resultado que se encuentra en la lista de valores.

Ejemplos:

```
Prelude> esCerrada (concatenar "xx") ["esta","sera","cerrada?","s","e"] -- False
Prelude> esCerrada (inters "en") ["esta","sera","cerrada?","s","e",[]]  -- True
Prelude> esCerrada not [True,False]                                     -- True
Prelude> esCerrada (suma 1) [1,0,-1]                                    -- False
```

b) Definir una función `obtieneCompCerradas`, y todas las auxiliares necesarias, tal que dadas dos listas de funciones y una lista de valores, devuelve la lista de funciones compuestas `(.)` de a pares (tomando una y una de cada lista), que verifican ser cerradas.

Ejemplos:

```
Prelude> obtieneCompCerradas [suma 1, pot 3, resta 2] [suma 9, pot 4, suma 2] [-1,0,1]
-- [pot 3 . pot 4,resta 2 . suma 2]

Prelude> obtieneCompCerradas [suma 1, pot 3] [suma 9, pot 4] [10]
-- []

Prelude> obtieneCompCerradas [inters "bb",extraeNDesde 1 2]
                               [extraeNDesde 1 2,concatenar "aa"] ["aa",[]]
-- [inters "bb" . extraeNDesde 1 2]
```

Ejercicio 27

a) Definir una función `listaDuplas` que reciba como parámetros una función y una lista, y devuelva la lista de duplas que resultan de aplicar la función a cada elemento de la lista.

b) Definir una función `filtraDuplas` que reciba como parámetros una función y una lista de duplas, y devuelva la lista de duplas cuya 2da. componente verifique la función.

c) Definir una función de orden superior `filtraListaDuplas`, aplicando el operador `(.)`, componiendo las dos anteriores.

Ejemplos:

```
Prelude> filtraListaTuplas (<6) longitud ["buenas","tardes","a","todos"]
-- [("buenas",6),("tardes",6)]

Prelude> filtraListaTuplas (/='s') ultimo ["buenas","tardes","a","todos"]
-- [("buenas",'s'),("tardes",'s'),("todos",'s')]

Prelude> filtraListaDuplas ((=='h').primero) sinPrimero ["hola","ala","chau"]
-- [("hola","ola"),("ala","la")]
```

Ejercicio 28

a) Definir una función `aplicarFunciones` que reciba como parámetros una lista de funciones y una lista de valores, y devuelva la lista de duplas formadas por cada valor y los resultados que se obtienen al aplicar las funciones de la lista al mismo.

b) Definir una función `eliminarValores` que reciba como parámetros una función y una lista de duplas, y devuelva la lista de duplas sin aquellos valores de la 2da. componente que verifiquen la función.

c) Describir el objetivo y determinar si es correcta la siguiente definición de la función `aplicarYEliminar`

```
aplicarYEliminar fs f = eliminarValores f . aplicarFunciones fs
```

Si no fuera correcta, modificar la definición con el fin de lograr el mismo objetivo.

Ejemplos:

```
Prelude> aplicarYEliminar [cuadrado, cubo, (mod 3)] (>10) [4,3,2]
-- [(4,[1]), (3,[9,0]), (2,[4,8,2])]

Prelude> aplicarYEliminar [primero, ultimo] (/='a') ["a", "la", "loca","mora"]
-- [("a","aa"),("la","a"),("loca","a"),("mora","a")]
```

Ejercicio 29

Dada la función:

```
func _ _ _ [] = []
func f g x (y:ys)   | f (g y) x = y : func f g x ys
                    | otherwise = func f g x ys
```

a) Definir su prototipo y objetivo.
b) Aplicando `func`, definir las siguientes funciones y todas las auxiliares necesarias:

- `sublistas`, tal que dada una lista de valores y una función, devuelve una lista conformada por sublistas que agrupan aquellos elementos que, al aplicarle la función, producen el mismo resultado.

Ejemplos:

```
Prelude> sublistas [1,3,-2,-1,2] cuadrado
-- [[1,-1], [3], [-2,2]]
Prelude> sublistas ["hola","este","ejemplo","puede","andar"] longitud
-- [["hola","este"], ["ejemplo"], ["puede","andar"]]
```

- `superanProm`, tal que dada una lista de materias, una lista de tuplas `(alumno, materia, nota1, nota2)`, que contienen las notas obtenidas por distintos alumnos en dos parciales de distintas materias, y un promedio mínimo, permite obtener una lista de duplas `(materia, alumnosAprobados)`, que indican para cada materia la lista de alumnos que obtuvieron un promedio mayor al promedio mínimo indicado.

Ejemplo:

```
Prelude> superanProm    ["algoritmos","paradigmas"]
                        [    ("juan","algoritmos",6,7),
                             ("sofia","sintaxis",8,5),
                             ("maria","paradigmas",9,7),
                             ("diego","algoritmos",2,6),
                             ("maria","algoritmos",6,8),
                             ("pedro","paradigmas",7,4),
                             ("jose","paradigmas",2,5)    ]
                        6
-- [("algoritmos",["juan","maria"]),("paradigmas",["maria"])]
```

Ejercicio 30

Dada la siguiente función:

```
func _ [] [] = True
func _ _ [] = False
func _ [] _ = False
func g (x:xs) (y:ys) = g x y && func g xs ys
```

Definir su prototipo y objetivo.

Ejercicio 31

Aplicando la función anterior, codificar las siguientes funciones, y todas las auxiliares necesarias:

a) La función `sumasDivisibles`, que recibe un entero y dos listas de enteros, y verifica que la suma de cada par conformado por un elemento de cada lista es divisible por el entero indicado.

Ejemplos:

```
Prelude> sumasDivisibles  3  [1,3,5,-2]  [0,0,4,5] -- False
Prelude> sumasDivisibles  3  [1,3,5,-2]  [8,0,4,5] -- True
```

b) La función `espejo`, que recibe dos listas y verifica que las mismas sean iguales, o a los sumo "iguales en espejo", es decir, que la primera sea igual a la segunda reflejada en un espejo.

Ejemplos:

```
Prelude> espejo "hola" "aloh"                        -- True
Prelude> espejo [1,2,3] [1,2,4]                      -- False
Prelude> espejo ["Feliz","2004!"] ["Feliz","2004!"] -- True
```