Nombre y apellido: [redacted]
Carrera: [redacted]
L.U. o D.N.I.: [redacted]
Número de orden: [redacted]
Cant. de hojas:

Departamento de Computación – FCEyN – UBA

Taller de Álgebra I - Parcial

SEGUNDO CUATRIMESTRE 2018 – *27 de octubre de 2018*

Ap

Aclaraciones

- El parcial se aprueba con tres ejercicios bien resueltos.
- Programe todas las funciones en lenguaje Haskell. El código debe ser autocontenido. Si utiliza funciones que no existen en Haskell, debe programarlas. Incluya la signatura de todas las funciones que escriba.
- No está permitido alterar los tipos de datos presentados en el enunciado, ni utilizar técnicas no vistas en clase para resolver los ejercicios.

Ejercicio 1

Definir una función `suma2 :: (Integer,Integer,Integer) -> Integer -> Bool` que dada una tupla (a, b, c) de números enteros y un entero n devuelva verdadero si dos de las coordenadas de la tupla suman n y falso en caso contrario.

Por ejemplo:
`suma2 (5, -2, 10) 8` $\rightsquigarrow$ `True` porque $-2 + 10 = 8$.
`suma2 (5, -2, 10) 7` $\rightsquigarrow$ `False` porque $5 - 2 \neq 7$, $5 + 10 \neq 7$ y $-2 + 10 \neq 7$.

Ejercicio 2

Programar la función `sumatoria :: Integer -> Integer` que dado un entero $n \geq 1$ calcule:

$$S(n) = \sum_{i=1}^{n} (2i - 1)^2.$$

Ejercicio 3

Programe las funciones:

`todosIguales :: [Integer] -> Bool`, que dada una lista de enteros devuelve verdadero si y solamente si todos sus elementos son iguales.

`todosDistintos :: [Integer] -> Bool`, que dada una lista de enteros devuelve verdadero si y solamente si todos sus elementos son diferentes.

Ejercicio 4

Programar una función `sacarTodos :: [Integer] -> [Integer] -> [Integer]` que dadas dos listas de enteros, quite de la primera todas las ocurrencias de los elementos de la segunda lista.

Por ejemplo:
`sacarTodos [3,4,5,1,5,2,2] [1,6,5]` $\rightsquigarrow$ `[3,4,2,2]`.

Ejercicio 5

Programar la función `promedioDe :: [(Integer,Float)] -> Integer -> Float` que dada una lista de tuplas `(Integer,Float)` (donde la primera coordenada representa el código del alumno y la segunda la nota que obtuvo en cada examen) y un código de alumno, retorne el promedio de las notas de ese alumno. Pueden asumir que todo alumno tiene por lo menos una nota.

Por ejemplo:
`promedioDe [(101,9),(102,1.5),(101,4.5),(104,5),(102,5),(101,6),(102,5.4)] 101` $\rightsquigarrow$ `6.5`.

Hoja 1/1

```
-- Ejercicio 1:

suma2 :: (Integer, Integer, Integer) -> Integer -> Bool

suma2 (a,b,c) n | ((a+b)==n) || ((b+c)==n) || ((a+c)==n) = True
                | otherwise = False.

-- Ejercicio 2:

sumatoria :: Integer -> Integer.
sumatoria 1 = 1
sumatoria n = (2n-1)^2 + sumatoria (n-1)

-- Ejercicio 3:

todosIguales :: [Integer] -> Bool
todosIguales [] = True      -- por si se ingresa lista vacia.
todosIguales (x1:[]) = True
todosIguales (x1:x2:xs) | x1 /= x2 = False
                        | otherwise = todosIguales (x2:xs)

-- Aclaración: '/=' es el comparador 'distinto de'.
-- Para todosDistintos supongo lista de AL Menos 2 elementos, pues no tiene sentido
                                                  -- comparar con 0 ó 1
todosDistintos :: [Integer] -> Bool
todosDistintos [] = True    -- por si se ingresa lista vacia.
todosDistintos (x:[]) = True    -- con No hay dos elementos iguales, vale.
todosDistintos (x:xs) | (compara x xs) == True = todosDistintos xs
                      | otherwise = False
-- compara :: Integer -> [Integer] -> Bool
-- compara _ [] = True
-- sigue el dorso porque no entra :(
```

```
-- 'compara' compara 'y' con los elementos de una lista y
-- devuelve True si 'y' es diferente a todos ellos.

compara :: Integer -> [Integer] -> Bool
compara _ [] = True
compara y (x:xs) | y == x = False
                 | otherwise = compara y xs

-- Ejercicio 4:
sacarTodos :: [Integer] -> [Integer] -> [Integer]
sacarTodos [] _ = []          -- por si se ingresa lista vacia.
sacarTodos xs [] = xs
sacarTodos xs (n:ns) = sacarTodos (sacar n xs) ns

sacar :: Integer -> [Integer] -> [Integer]
sacar _ [] = []
sacar n (x:xs) | n == x = sacar n xs
               | otherwise = x : (sacar n xs)

-- Ejercicio 5: (suponga lista de tuplas de notas no-vacía)
promedioDe :: [(Integer, Float)] -> Integer -> Float
promedioDe ns id = calculoPromedioDe ns id 0 0

calculoPromedioDe :: [(Integer, Float)] -> Integer -> Float -> Float -> Float
calculoPromedioDe [] _ suma cont = suma / cont
calculoPromedioDe (n:ns) id suma cont | id == (fst n) = calculoPromedioDe ns id sumaNot incC
                                      | otherwise = calculoPromedioDe ns id suma cont
                                      where sumaNot = suma + (snd n)
                                            incC = cont + 1

-- el "where" lo agregué porque no me entraba la línea en la hoja.
-- NUNCA divido por cero porque asumimos que todo alumno tiene una nota
-- y no se ingresan alumnos que no existen
```