UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

Paradigmas de Programación

TRABAJO PRÁCTICO N° 5

Programación Funcional

1. Problemas

Ejercicio 1.1. Definir, por comprensión y por recursion, la función:

A. suma de cuadrados

B. suma de cuadrados impares

C. suma de cuadrados pares

Ejercicio 1.2. Definir por comprensión los siguientes enunciados (si es necesario, definir funciones de ayuda para computar algunas tareas extras):

A. Lista de números pares hasta el valor n ingresado.

B. Dado dos valores ingresados n y m, generar la lista de valores pares hasta n teniendo en cuenta que solo pueden ser generados aquellos mayores a m

C. Mostrar los divisores de un número

D. Para una lista de números positivos reemplazar cada número x por x copias del mismo.

E. Definir una función que permita recibir una tupla de 3 elementos que indique el número de elementos pares

F. Devolver una lista de números primos de 1 a n

G. Un tupla de tres elementos (x, y, z) de valores enteros positivos se puede denominar pitagórico si $x^2 + y^2 = z^2$. Utilizando una lista por comprensión, definir una función $pitagoricamente :: Int \rightarrow [(Int, Int, Int)]$ que permita devolver las tuplas que cumplan con la ecuación pitagórica

H. Un número entero positivo es perfecto si es igual a la suma de todos sus factores, excluyendo el número mismo. Usando una lista por comprensión.

   Definir una función numeroPerfecto :: Int → [Int], que permite devolver la lista de todos los números perfectos hasta un límite dado

I. El producto escalar de dos listas de enteros *xs* e *ys* de longitud *n* esta conformada por la suma de los productos de los enteros correspondientes. Utilizando listas por comprensión, definir una función que devuelva el producto escalar de dos listas "

J.
```
buscaCrucigrama :: Char -> Int -> Int -> [String] -> [String]
```

   tal que (buscaCrucigrama l pos lon ps) es la lista de las palabras de la lista de palabras ps que tienen longitud lon y poseen la letra l en la posición pos (comenzando en 0). Por ejemplo,

```
ghci> buscaCrucigrama 'c' 1 7 ["ocaso", "acabado", "ocupado"]
["acabado","ocupado"]
```

K. `posiciones :: String -> Char -> [Int]`

tal que (posiciones xs y) es la lista de la posiciones del carácter y en la cadena xs. Por ejemplo,

```
posiciones "Salamamca" 'a' == [1,3,5,8]
```

L. Se consideran las siguientes reglas de mayúsculas iniciales para los titulos:

- la primera palabra comienza en mayúscula y
- todas las palabras que tienen 4 letras como mínimo empiezan con mayúsculas.

```
titulo :: [String] -> [String]
```

tal que (titulo ps) es la lista de las palabras de ps con las reglas de mayúsculas iniciales de los títulos. Por ejemplo,

```
ghci> titulo ["eL","arTE","DE","La","proGraMacion"]
["El","Arte","de","la","Programacion"]
```

M. La criba de Erastótenes es un método para calcular números primos.

- Se comienza escribiendo todos los números desde 2 hasta (supongamos) 100.
- El primer número (el 2) es primo.
- Ahora eliminamos todos los múltiplos de 2.
- El primero de los números restantes (el 3) también es primo.
- Ahora eliminamos todos los múltiplos de 3.
- El primero de los números restantes (el 5) también es primo . . . y así sucesivamente.
- Cuando no quedan números, se han encontrado todos los números primos en el rango fijado.

Definir la función cribaErastotenes que permita calcular los números primos hasta el valor n.

Ejercicio 1.3. Teniendo en cuenta la definición de la función predefinida foldr, redefinir la suma y el producto

Ejercicio 1.4. Teniendo en cuenta las siguientes funciones predefinidas en Haskell

```
map       :: (a -> b) -> [a] -> [b]
filter    :: (a -> Bool) -> [a] -> [a]
takeWhile :: (a -> Bool) -> [a] -> [a]
dropWhile :: (a -> Bool) -> [a] -> [a]
iterate   :: (a -> a) -> a -> [a]
zipWith   :: (a -> b -> c) -> [a] -> [b] -> [c]
```

Definir las siguientes funciones de alto orden:

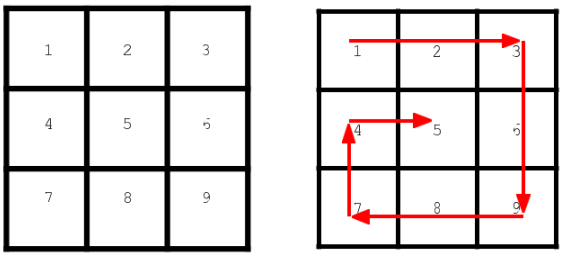
- Da el número de elementos alineados en dos listas.
- Crea una función que toma una lista y otra función como entrada, aplicar esa función a cada elemento de esa lista y devolver la nueva lista.
- Filtrar los números de una lista que sean menores a 5
- Crear una función que multiplique todos los elementos mayores a 0×2

Ejercicio 1.5. Implementar las soluciones para los siguientes problemas mediante uso de orden superior:

A. Definir la función de orden superior $dosVeces$ tal que repita una función y un argumento dos veces.

B. Definir la función que imprima cada elemento de una matriz de forma en espiral transversal hacia el centro de la matriz.

   Por ejemplo, ruta de impresión de una matriz 3×3 como el de la imagen da como resultado 1,2,3,6,9,8,7,4,5.



C. En los documentos financieros, como los cheques, los números a veces se deben escribir con palabras completas. Ejemplo: 175 debe ser escrito como uno-siete-cinco. Escriba un predicado full-words/1 para imprimir números enteros (no negativos) en palabras completas.

Ejercicio 1.6. Evaluar el funcionamiento mediante evaluación ansiosa y perezosa.

A.
```
inf :: Int
inf = 1 + inf
```

B.
```
cuadrado :: Int -> Int
cuadrado n = n * n
```

C.
```
square x = x * x

square (square (square 2))
```

D.
```
unos :: [Int]
unos = 1 : unos
```

E.
```
naturalesDesde x = x : naturalesDesde (x+1)
```

F.
```
negate $ 5 * sqrt 16
```

G.
```
mult(a, b) = a * a
```

H.
```
loop(x) = loop(x)
```

I.
```
mult(2, loop(2))
```

J.
```
foldl :: (b -> a -> b) -> b -> [a] -> b
foldl _ z []     = z
foldl f z (x:xs) = foldl f (f z x) xs

foldl (+) 0 [1,2,3]
```

Ejercicio 1.7. Analizar como funciona en python y haskell el siguiente trozo de código la evaluación perezosa, teniendo en cuenta las siguientes expresiones:

A. agregar 4 5 y agregar 10 (89/0)

```
def agregar( x , y ):
   return x + x

agregar x y = x + x
```

B. Con x=1 e y =2, hipotenusa x+1 y+2

```
def hipotenusa(a,b):
   return math.sqrt(a ** 2 + b ** 2)

hipotenusa a b = sqrt (a ^ 2 + b ^ 2)
```

2. Ejercicios Integradores

Ejercicio 2.1. Dada una cadena que contiene corchetes [], llaves , paréntesis () o cualquier combinación de estos, se debe verificar que todos y cada uno de los pares coincidan y estén anidados correctamente.

Ejercicio 2.2. Si dijeramos que alguien tiene una edad de 1000000000 segundos tambien podriamos poder decir que tiene 31.69 años terrestres.

El periodo orbital es el tiempo que le toma a un astro recorrer su órbita y tiene un valor en años de nuestro planeta. Teniendo en cuenta los siguientes periodos orbitales de nuestro sistema solar:

- Mercurio: periodo orbital equivale a 0.2408467 años terrestres
- Venus: periodo orbital equivale a 0.61519726 años terrestres
- Tierra: periodo orbital equivale a 1.0 años terrestres, 365.25 días terrestres, or 31557600 segundos
- Marte: periodo orbital equivale a 1.8808158 años terrestres
- Jupiter: periodo orbital equivale a 11.862615 años terrestres
- Saturno: periodo orbital equivale a 29.447498 años terrestres
- Urano: periodo orbital equivale a 84.016846 años terrestres
- Neptuno: periodo orbital equivale a 164.79132 años terrestres

Calcular en base a una cantidad de segundos y un planeta mostrar los años terrestres equivalente:

```
ghci> edad Marte 1000000000
16.848055
```

Ejercicio 2.3. A partir de una frase, contar las apariciones de cada palabra en dicha frase. Algunas acotaciones sobre la conformación de palabras:

- Una palabra puede ser un número compuesto por uno o más dígitos ASCII (es decir, *0* o *bb232a3*)
- Una palabra simple compuesta de una o más letras ASCII (es decir, *a* o *ellos*) O
- Una contracción de dos palabras simples unidas por un solo apóstrofe (es decir, *es* o *son*)

Al contar palabras, puede asumir las siguientes reglas:

- El recuento no distingue entre mayúsculas y minúsculas (es decir, “Usted”,“usted” y “USTED” son 3 usos de la misma palabra)
- El recuento no está ordenado; las pruebas deben ignorar cómo se ordenan las palabras y los recuentos
- Aparte del apóstrofe en una contracción, se ignoran todas las formas de puntuación
- Las palabras se pueden separar por cualquier forma de espacio en blanco (es decir, \t, \n, “”)

Por ejemplo, para la frase “Mi hija es un dolor de cabeza, diosss”. el conteo va a ser:

```
ghci> contarPalabras "Mi hija es un dolor de cabeza, diosss"

[("mi",1),("hija",1),("es",1),("un",1),("dolor",1),("de",1),("cabeza",1),("diosss",1)]
```

Para completar este ejercicio, debe implementar la función contarPalabras, que toma un texto y devuelve cuántas veces aparece cada palabra. Se puede utilizar la siguiente firma como base(no es obligatorio).

```
contarPalabras :: String -> [(String, Int)]
```

Ejercicio 2.4. El proceso de verificación ISBN-10 se utiliza para validar los números de identificación de libros. Estos normalmente contienen guiones y su codificación así: 3-598-21508-8

El formato ISBN-10 tiene 9 dígitos (0 a 9) más un carácter de verificación (ya sea un dígito o solo una X). En el caso de que el carácter de verificación sea una X, esto representa el valor '10'. Estos se pueden comunicar con o sin guiones, y se puede verificar su validez mediante la siguiente fórmula:

```
(x1 * 10 + x2 * 9 + x3 * 8 + x4 * 7 + x5 * 6 +
 x6 * 5 + x7 * 4 + x8 * 3 + x9 * 2 + x10 * 1) mod 11 == 0
```

Si el resultado es 0, entonces es un ISBN-10 válido; de lo contrario, no es válido.

Si tomamos como ejemplo el ISBN-10 3-598-21508-8. Lo evaluamos mediante la fórmula y obtenemos:

```
(3 * 10 + 5 * 9 + 9 * 8 + 8 * 7 + 2 * 6 + 1 * 5 + 5 * 4 + 0 * 3 + 8 * 2 + 8 * 1) mod 11 == 0
```

Dado el resultado = 0, esto significa que nuestro ISBN es válido.

Se solicita que dado una cadena, el programa debe verificar si dicha cadena es un ISBN-10 válido. El programa debe poder verificar ISBN-10 con y sin utilizar guiones.

Ejercicio 2.5. Escribir la letra de una canción conocida en los bares irlandeses: 99 Botellas de cerveza(ron, whisky, manaos, lo que más le guste) en la pared. Tenga en cuenta que no todos los versos son idénticos.

- 99 botellas de cerveza en la pared, 99 botellas de cerveza. Una se cayó y quedaron 98 botellas de cerveza en la pared.
- 98 botellas de cerveza en la pared, 98 botellas de cerveza. Una se cayó y quedaron 97 botellas de cerveza en la pared.
- 97 botellas de cerveza en la pared, 97 botellas de cerveza. Una se cayó y quedaron 96 botellas de cerveza en la pared.
- 96 botellas de cerveza en la pared, 96 botellas de cerveza. Una se cayó y quedaron 95 botellas de cerveza en la pared.
- ...
- ...
- ...
- 2 botellas de cerveza en la pared, 2 botellas de cerveza. Una se cayó, 1 botella de cerveza en la pared.
- 1 botella de cerveza en la pared, 1 botella de cerveza. Una se cayó, no más botellas de cerveza en la pared.
- No hay más botellas de cerveza en la pared, no más botellas de cerveza.
- Ve a la tienda y compra más, 99 botellas de cerveza en la pared.

Ejercicio 2.6. Ayudar a evaluar el funcionamiento de un lindo R2-D2. Se necesita hacer una prueba de fábrica de robots necesita un programa para verificar los movimientos del robot.

Los robots tienen tres movimientos posibles:

- doblar a la derecha
- doblar a la izquierda
- avanzar

Los robots se colocan en una cuadrícula hipotéticamente infinita donde el inicio de los ejes es en la esquina superior oeste, con una dirección particular (Norte, Este, Sur u Oeste) en un conjunto de coordenadas (x, y), por ejemplo, (3,8), con coordenadas que aumentan hacia el norte y el este.

Luego, el robot recibe una serie de instrucciones, momento en el cual la prueba verifica la nueva posición del robot y en qué dirección apunta.

Una cadena de letras "DAALAL"significa:

- Girar a la derecha
- Avanzar dos veces
- Girar a la izquierda
- Avanzar una vez
- Gira a la izquierda una vez más

Un robot comienza en (7, 3) ubicado hacia el norte. Luego, ejecutando esta secuencia de instrucciones queda en (9, 4) hacia el oeste.

Se pide modelar un robot que se mueva desde una posición (x,y) a otro par de coordenadas mediante un conjunto de instrucciones.

Recomendaciones:

- Se aconseja crear el tipo de datos Robo y ubicación e implementar las siguientes funciones:
  - crearRobot
  - simular
  - coordenadas
  - girar a la izquierda
  - girar a la derecha

  NOTA: Las funciones a definir son de ayuda, si ud. desea implementar más o menos funciones puede hacerlo tranquilamente.

3. Teoría

Ejercicio 3.1. Definir que significa currificación.

Ejercicio 3.2. Que significa que una función sea de orden superior. Justifique su respuesta.

Ejercicio 3.3. Definir el proceso de evaluación perezosa en un programa funcional. Que ventajas ofrece sobre una evaluación ansiosa.

Ejercicio 3.4. Las monadas nos permiten solventar diferentes problemas. ¿Cuales serían ellos? Justifique como estos problemas afectan a un lenguaje funcional.

Ejercicio 3.5. ¿Existen diferencias entre el polimorfismo tipado y no tipado en haskell? Justificar

Ejercicio 3.6. Definir y describir mediante ejemplos los constructores de tipo versus los constructores de datos utilizados en haskell.