Universidad Nacional Autónoma de México
Facultad de Ciencias
Estructuras Discretas

Práctica 1

Javier Enríquez Mendoza
javierem_94@ciencias.unam.mx

17 de Agosto de 2018
Fecha de entrega: 31 de Agosto de 2018

Objetivos

- Familiarizar al alumno con el lenguaje de programación Haskell y el intérprete GHC[1].
- Definir funciones sencillas en Haskell.
- Utilizar los tipos básicos y algunas funciones predefinidas del lenguaje.
- Aprender a usar el intérprete GHC para escribir pequeños programas.
- Reforzar los conocimientos obtenidos en el laboratorio.

Instrucciones generales

La práctica consiste en definir varias funciones que resuelvan el listado de problemas a continuación. La práctica debe resolverse en un archivo `Practica1.hs` y las firmas de las funciones deben ser idénticas a las que se muestran en cada ejercicio. Cada función debe estar debidamente comentada con la especificación de ésta.

Se tomará en cuenta la legibilidad y el estilo del código. **No** pueden usarse funciones predefinidas del lenguaje que resuelvan directamente los ejercicios.

Ejercicios

Ejercicio 1.1 (1 pt.) Definir la función `areaCirculo` que recibe el radio de un círculo y regresa el área de éste. Utilizar la constante `pi` predefinida en Haskell.

```
areaCirculo :: Float -> Float
```

[1] Glasgow Haskell Compiler

```
> areaCirculo 3.8
45.36459791783661
> areaCirculo 7.0
153.93804002589985
```

Ejercicio 1.2 (1 pt.) Definir la función `distancia` que recibe dos puntos y regresa la distancia entre ellos.

```
distancia :: (Float,Float) -> (Float,Float) -> Float
```

```
> distancia (2.0,8.0) (1.0,1.0)
7.071067811865
> distancia (3.0,3.0) (3.0,3.0)
0
```

Ejercicio 1.3 (1 pt.) Definir la función `imp` que recibe dos booleanos y regresa la implicación lógica de éstos.

```
imp :: Bool -> Bool -> Bool
```

```
> imp True Flase
False
> imp True True
True
```

Ejercicio 1.4 (1 pt.) Definir la función `xor` que recibe dos dos booleanos y regresa la disyunción exclusiva de éstos.

```
xor :: Bool -> Bool -> Bool
```

```
> xor True False
True
> xor False False
False
```

Ejercicio 1.5 (1 pt.) Definir una función `mes` que recibe un entero en el rango de 1 a 12 y regresa una cadena con el mes que representa el entero.

```
mes :: Int -> String
```

```
> mes 1
‘‘Enero’’
> mes 11
‘‘Noviembre’’
```

Ejercicio 1.6 (1 pt.) Definir la función `calculadora` que recibe dos parámetros: el primero indica la operación que se va a realizar y el segundo es una tupla con los operandos de la misma, las posibles operaciones son:

‘‘`first`’’ = devuelve el primer elemento de la tupla
‘‘`last`’’ = devuelve el segundo elemento de la tupla
‘‘`sum`’’ = suma
‘‘`rest`’’ = resta
‘‘`mul`’’ = multiplicación
‘‘`div`’’ = división
‘‘`pow`’’ = potencia (el primer elemento de la tupla elevado al segundo)

```
calculadora :: String -> (Int,Int) -> Int
```

```
> calculadora ‘‘pow’’ (2,8)
256
> calculadora ‘‘last’’ (365,74)
74
```

Ejercicio 1.7 (1 pt.) Al gato Loki le encanta pasar el día jugando en la calle, pero sólo lo hace cuando la temperatura está entre 15 y 25 grados, menos en verano que tolera un poco más el calor y sale a jugar cuando la temperatura ésta entre 20 y 30. Definir una función `loki` que recibe la temperatura y un booleano indicando si es o no verano, y le diga a Loki si puede salir a jugar.

```
loki :: Int -> Bool -> String
```

```
> loki 25 False
‘‘No sale a jugar’’
> loki 20 False
’’Sale a jugar’’
```

Ejercicio 1.8 (1 pt.) Se tiene un par de monos, que pueden o no estar sonriendo. Hay problemas cuando ambos monos están sonriendo o cuando ninguno de los dos sonríe. Definir una función `monos` que recibe 2 booleanos (indicando si los monos están sonriendo o no) e indique si hay algún tipo de problema.

```
monos :: Bool -> Bool -> String
```

```
> monos True True
‘‘Hay Problemas’’
> monos True False
‘‘No hay Problemas’’
```

Ejercicio 1.9 (1 pt.) Definir la función `multiplica` que recibe dos números y los multiplica. Utilizar la función `suma` vista en clase. **No** se puede utilizar la función de multiplicación predefinida en Haskell.

```
multiplica :: Int -> Int -> Int
```

```
> multiplica 4 25
100
> multiplica 3 7
21
```

Ejercicio 1.10 (1 pt.) Definir la función `potencia` que recibe dos números y regresa el primero elevado por el segundo. Utilizar la función `multiplica` del ejercicio anterior. **No** se puede utilizar las funciones de potencia predefinidas en Haskell.

```
potencia :: Int -> Int -> Int
```

```
> potencia 2 8
256
> potencia 3 7
2187
```

Entrega

- La entrega se realiza mediante correo electrónico a la dirección del ayudante de laboratorio (javierem_94+ed@ciencias.unam.mx).
- La practica deberá ser entregada de forma **individual**.
- Se debe entregar un directorio `numeroCuenta_P01`, dónde `numeroCuenta` es el número de cuenta del alumno. Dentro del directorio se debe incluir:
  - Un archivo readme.txt con el nombre y número de cuenta del alumno, comentarios, opiniones, críticas o ideas sobre la práctica.
  - Los archivos requeridos en la práctica. Debe enviarse código lo más limpio posible.
- El único archivo requerido es `Practica1.hs`.
- El directorio se deberá comprimir en un archivo con nombre `numeroCuenta_P01.tar.gz`, dónde `numeroCuenta` es el número de cuenta del alumno.
- El asunto del correo debe ser [ED-20191-P01].
- Se recibirá la práctica hasta las 23:59:59 horas del día fijado como fecha de entrega, cualquier práctica recibida después no será tomada en cuenta.
- **Cualquier práctica total o parcialmente plagiada, será calificada automáticamente con cero y no se aceptarán más prácticas durante el semestre.**