

Facultad de Ciencias - UNAM

Estructuras Discretas 2023-2

Práctica 1: Introducción a Haskell

Laura Friedberg Gojman
Saúl Adrián Rojas Reyes
José Manuel Madrigal Ramírez

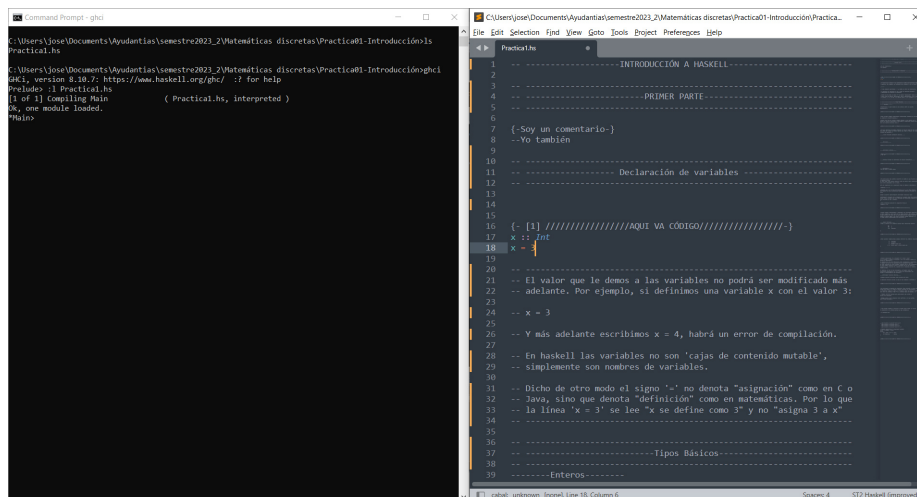
3/febrero/2023
Fecha de entrega: 12/febrero/2023

Instrucciones

Junto a este PDF, subí un archivo llamado **Practica1.hs**. Descárgalo y ábrelo en tu editor de texto. Abre también una terminal situada en la carpeta donde guardaste el archivo.

Escribe **ghci** en la terminal y pulsa enter, esto inicializará el compilador de Haskell. A continuación ejecuta el siguiente comando:

:l Practica1.hs



The screenshot shows two windows. On the left is a Command Prompt window with the following text:

```
C:\Users\Jose\Documents\Ayudantias\semestre2023_2\Matemáticas discretas\Practica01-Introducción>ls
Practica1.hs
C:\Users\Jose\Documents\Ayudantias\semestre2023_2\Matemáticas discretas\Practica01-Introducción>ghci
ghci, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :l Practica1.hs
(1 of 1) Compiling Main
OK, one module loaded.
*Main>
```

On the right is a text editor window showing the content of **Practica1.hs**. The file contains comments in Spanish explaining Haskell syntax and basic concepts. The code starts with a variable declaration `x :: Int` and a value assignment `x = 3`. The editor has a dark theme and line numbers on the left.

Con ello se cargará nuestro archivo **Practica1.hs** en el compilador.

Los comentarios dentro de GHCi siempre van precedidos de dos puntos. Es útil recordar algunos de ellos:

<code>:l <nombre de archivo></code>	carga el archivo en GHCi
<code>:r</code>	recarga el archivo
<code>:q</code>	cierra el compilador
<code>:h</code>	despliega todos los comandos disponibles

Esta práctica tiene como objetivo ser una introducción a los conceptos básicos de Haskell. En el archivo `Practical.hs` encontrarás las instrucciones de cada ejercicio a realizar. En la siguiente sección de este documento también encontrarás las instrucciones de los ejercicios junto con una serie de comentarios útiles al respecto.

Siempre que completes un ejercicio es recomendable guardar tu trabajo en el editor de texto y ejecutar `:r` en GHCi para cargar los cambios.

Ejercicios

Tipos de dato

1.- En Haskell cada variable está asociada a un tipo. Mientras nos familiarizamos con esta idea, siempre que vayamos a declarar una variable debemos comenzar con una línea parecida a la siguiente:

```
x :: Int
```

- `x` es el nombre (identificador) de la variable, podría ser cualquier otra letra que nos guste, una palabra o varias palabras. En este último caso, por convención, escribimos las palabras que compongan el nombre del identificador en camelCase.
- `::` es un símbolo que podemos leer como "es del tipo".
- `Int` es el tipo entero restringido por la arquitectura de la computadora (en breve más al respecto).

Tras especificar el tipo de la variable escribimos el valor que define:

```
x = 3
```

En Haskell las variables no son *cajas de contenido mutable*, simplemente son nombres de variables.

Dicho de otro modo el signo '=' no denota "asignación" como en C o Java, sino que denota "definición" como en matemáticas. Por lo que la línea '`x = 3`' se lee "x se define como 3" y no "asigna 3 a x".

Declara tres variables de tipo entero, asegúrate de incluir números positivos y negativos. Puedes utilizar los nombres que gustes.

2.- Las variables de tipo `Int` se encuentran limitadas por la arquitectura de nuestras computadoras. Si nuestra computadora es de 64 bits, el valor máximo del tipo `Int` será de:

$$2^{63} = 9223372036854775808$$

y el mínimo será lo mismo pero en negativo: -9223372036854775808.

`maxBound` y `minBound` son funciones predefinidas en Haskell que devuelven el valor de tipo `Int` que no tiene sucesor o que no tiene predecesor respectivamente.

Ejecuta las siguientes líneas de código para saber cuál es el máximo y el mínimo valor que puede tener un dato de tipo `Int` en tu computadora. Captura la pantalla con el resultado de la ejecución.

```
minimo, maximo :: Int
maximo = maxBound
minimo = minBound
```

3.- `Integer` es otro tipo de dato para designar enteros en Haskell cuyo límite queda en la memoria de nuestra computadora, y no en su arquitectura; por lo que será posible computar números realmente grandes.

- *Declara una variable de tipo `Integer` y defínela como un valor de 30 dígitos.*
- *Define el valor de la variable 'descomunal' como el resultado de elevar dos al cuadrado cuatro veces:*

$$2^{2^{2^2}}$$

Para elevar un número a una potencia dada utilizamos el símbolo: `^`. Por ejemplo:

$$2^4 \text{ se escribe } 2 \text{ } ^ 4$$

- *Completado el paso anterior ejecuta `numDigitos` en la consola. Esta es la longitud del número descomunal. Realiza una captura de tus resultados:*

```
numDigitos :: Int
numDigitos = length (show descomunal)
```

Es buena práctica de Haskell utilizar el tipo de dato `Int` con valores que estén por debajo del límite máximo e `Integer` cuando requiramos números que se encuentren por encima.

4.- En Haskell tenemos `Doubles` y `Floats`, que son tipos de datos primitivos para los números reales. Existen dos formas principales de expresarlos.

Utilizando directamente el punto decimal: 3.14503 o con notación científica: 9.023×10^{-5} que en Haskell se expresa como 9.023e-5 .

Declara tres variables de tipo Double y tres variables de tipo Float. utilizando punto decimal y notación científica.

5.- También contamos con valores de tipo Boolean, es decir, True y False. Que deben escribirse con la primer letra en mayúscula.

Declara un par de variables de tipo Bool. Una que guarde el valor de falso y otra el valor de verdadero.

6.- Finalmente, dentro del conjunto de tipos básicos en Haskell contamos con caracteres, cuyo tipo de dato propiamente dicho es Char.

Declara tres variables de tipo Char.

7.- Las cadenas son listas de caracteres y tienen su propia sintaxis. Con el siguiente par de líneas declaramos una cadena en Haskell:

```
s :: String
s = "Hola Haskell!"
```

Define un par de variables cuyos valores sean cadenas con enunciados ingeniosos.

Operaciones

8.- Haskell también nos permite realizar operaciones aritméticas. Podemos utilizar los operadores infijos: +, *, /, -, 'mod'. Al realizar estas operaciones con valores explícitos ya no es necesario especificar el tipo de dato de la variable que los define.

Por ejemplo, al escribir e1 = 3 + 2 Haskell entiende que e1 es de tipo Int.

Observa que 'mod' (módulo) se encuentra entre acentos graves. **Un par de acentos graves permiten convertir el nombre de una función en un operador infijo.**

Define seis variables con el resultado de utilizar cada operador infijo en un par de valores y responde: ¿Cómo indicaríamos al compilador que realice restas de números negativos?

9.- En Haskell no existen las conversiones implícitas de tipo. Esto quiere decir que no podemos dividir el Float 3.4 entre el Int 2 y esperar que el compilador convierta este último en Float 2.0. Por lo tanto desde un principio tenemos que declarar ambos números como Floats y utilizar el operador /. Para obtener el cociente de dos números enteros requerimos el operador 'div'.

Declara dos variables cuyo valor sea el cociente de números enteros.

10.- En Haskell contamos con los siguientes operadores lógicos:

	o
&&	y
not	negación

Realiza tres operaciones con los operadores lógicos y define con ellas tres variables.

11.- Las comparaciones se consideran de tipo booleano porque a final de cuentas se evalúan a verdadero o falso.

==	igualdad
/=	desigual
> <	mayor y menor qué
>= <=	mayor igual y menor igual qué

Define tres variables de tipo Bool utilizando los símbolos de comparación.

Funciones

12.- Una función básica que calcula el resultado de sumar los primeros n naturales se ve así:

```
sumN :: Int -> Int
sumN 0 = 0
sumN n = n + sumN (n-1)
```

- La primer línea de código se refiere al tipo de la función. Va de los enteros a los enteros pues su dato de entrada es un número entero n y su salida es otro número entero s.
- **El par de líneas siguientes**, que definen el valor de la función en caso de recibir 0 y algún otro valor n como entrada, **se llaman cláusulas de la función**. Las funciones se componen de clausulas que pueden pensarse como casos distintos. (Si has programado en java o C, puedes pensar en las clausulas como switch).
- *Escribe la función anterior en la práctica.*
- *Evalúa la función con tres valores distintos en GHCi, guarda una captura de pantalla de tus resultados.*
- *Describe de forma detallada la ejecución de la función cuando su valor de entrada es el 4, haciendo explícita la participación de las cláusulas.*

13.- La siguiente función divide los números pares y triplica los impares:

```
foo :: Integer -> Integer
foo n
```

```
| n 'mod' 2 == 0 = n 'div' 2
| otherwise      = 3* n
```

- Los símbolos “|”, llamados guardas (guards), cumplen la función de seleccionar una acción si el valor de entrada cumple con alguna condición. En el ejemplo anterior se seleccionaba la división si el número era par y se triplicaba en otro caso.

Las guardas actúan de manera análoga a las estructuras de control if-else.

- Podemos ejecutar la función con el argumento 3 en gchi escribiendo:

```
foo 3
```

- *Copia la función foo en el archivo de la práctica.*
- *Evalúa la función con tres valores distintos en GHCi, guarda una captura de pantalla de tus resultados.*
- *Describe de forma detallada la ejecución de la función cuando su valor de entrada es el 5, haciendo explícita la participación de las guardas.*

14.- Aquí tenemos otro ejemplo de una función:

```
foo2 :: Integer -> Integer
foo2 0 = 16
foo2 1
  | "Haskell" > "Java" = 3
  | otherwise          = 4
foo2 n
  | n < 0              = 0
  | n 'mod' 17 == 2    = -43
  | otherwise          = n + 3
```

Cópiala en el archivo de la práctica y explica lo que hace. Después responde las siguientes preguntas:

- ¿Qué resulta al evaluar foo2 (-3) ?
- ¿Qué resulta al evaluar foo2 1 ?
- ¿Qué resulta al evaluar foo2 36 ?
- ¿Qué resulta al evaluar foo2 38 ?

15.- Aquí tenemos una función más:

```
isEven :: Integer -> Bool
isEven n
  | n 'mod' 2 == 0 = True
  | otherwise      = False
```

Escribe una versión simplificada (con menos líneas de código) de la función anterior. Pista: Recuerda que las comparaciones se evalúan a un valor booleano de falso o verdadero.

Entrega

La entrega es **individual** y consistirá en un archivo comprimido que contenga:

- El archivo `Practica1.hs` con el código indicado en los ejercicios.
- Las capturas de pantalla indicadas en los ejercicios 2, 3, 12 y 13.
- Un archivo `readMe.txt` que incluya tu nombre, las respuestas a los ejercicios 8, 12, 13 y 14, así como algún comentario o sugerencia sobre la práctica.

Sube el archivo comprimido a la plataforma Google Classroom utilizando el siguiente formato para el nombre:

Practica01_Apellido1-Apellido2-Nombre(s).zip

Por ejemplo, si Ada Lovelace Turing entregase su práctica le pondría como nombre a su archivo comprimido:

Practica01_Lovelace-Turing-Ada.zip

Cualquier duda que tengas no dudes en enviarme un correo. 😊