Facultad de Ciencias, UNAM
Práctica 1

Alejandro Hernández Mora Pablo Camacho González Luis Manuel Martínez Dámaso
Edith Araceli Reyes López Adrián Felipe Vélez Rivera

Fecha de entrega: 23 de octubre

Introducción.

En esta práctica vamos a trabajar con los conceptos básicos del lenguaje de programación **Haskell**.

Haskell.

Haskell es un lenguaje de programación **funcional**. Esto quiere decir que toda su operabilidad está basada en funciones.

Recordemos la definición de una función matemática. Primero hay que especificar el dominio y contradominio de la función. La notación para representar lo anterior es lo siguiente:

$$f : A \longrightarrow B$$

Donde A es el conjunto de donde vienen los elementos a los que le aplicaremos la función f y B el conjunto al que pertenece el resultado de aplicar la función.

Falta definir el comportamiento de la función, lo definiremos de la siguiente forma:

$$f(a) = b$$

De lo anterior, decimos que si al elemento a que pertenece al conjunto A, le aplicamos la función f, nos dará como resultado el elemento b, que pertenece al conjunto B.

Ya que Haskell trabaja con funciones, hay que ver la sintaxis con la que vamos a representarlas en nuestros programas.

2.1. Tipos.

Primero hay que ver de donde podemos tomar elementos para nuestras funciones. En Haskell hay distintos tipos de elementos con los que podemos trabajar. En esta práctica usaremos los siguientes tipos:

- **Int**. El tipo para representar a los números enteros.
- **Float**. El tipo para representar a los números reales con precisión simple.
- **Double**. El tipo para representar a los número reales con precisión .
- **Bool**. El tipo para representar a los valores boleanos **True** y **False**.

Con estos tipos ya podemos especificar que clase de elementos tendrán los conjuntos con los que trabaja la función. Para definir lo anterior en Haskell, haremos lo siguiente:

```
f :: A -> B
```

Con lo anterior decimos que la función f toma elementos del tipo A como parámetro y el resultado de aplicar la función es un elemento del tipo B. El tipo A y B pueden ser cualquiera de los tipos que mencionamos anteriormente. A esto le llamaremos **firma de la función**.

Para definir el comportamiento de la función f agregamos los siguiente:

```
f :: A -> B
f a = b
```

2.2. Operadores

Para trabajar con los diferentes tipos de elementos que hay en Haskell, vamos a ver los operadores que usaremos para resolver la práctica:

- **a+b**. Operador para sumar dos elementos de tipo numérico.
- **a−b**. Operador para restar dos elementos de tipo numérico.
- **a∗b**. Operador para multiplicar dos elementos de tipo numérico.
- **a/b**. Operador para dividir dos elementos de tipo **Float** o **Double**.
- **a^b**. Operador para elevar al elemento **a** a la potencia **b**. Los elementos **a** y b deben de ser de tipo **Float** o **Double**.
- **a>b**. Operador para saber si **a** es mayor que **b**. El resultado de usar este operador es de tipo **Bool**.
- **a<b**. Operador para saber si **a** es menor que **b**. El resultado de usar este operador es de tipo **Bool**.
- **a==b**. Operador para saber si **a** es igual que **b**. El resultado de usar este operador es de tipo **Bool**.

2.3. Funciones.

Ya sabemos como definir una función y los operadores que podemos usar en Haskell. Ahora veamos un ejemplo. Vamos a crear la función $f(x) = x + 1$, que tome como parámetro elementos de tipo **Float** y el resultado sea un elemento de tipo **Float**:

```
f :: Float -> Float
f x = x + 1
```

En el siguiente ejemplo tenemos una función para calcular el área de un rectángulo. La función se llama **areaRectangulo**, recibe como parámetros la base y la altura del rectángulo que queremos calcular el área. La base y la altura son de tipo **Float**. El resultado de la función es de tipo **Float**

```
areaRectangulo :: Float-> Float -> Float
areaRectangulo b a = b * a
```

Con el ejemplo anterior, podemos darnos cuenta que el nombre de nuestras funciones puede ser el que mejor describa a la función. Además, la función puede recibir cualquier número de parámetros. Cuando definimos una función, por cada parámetro debemos agregar el tipo de elemento que será en la firma de la función. Como la función **areaRectangulo** recibe dos elementos de tipo **Float** y el resultado es de tipo **Float** entonces la firma de la función queda asi:

```
areaRectangulo :: Float-> Float -> Float
```

2.4. Tuplas

Otra carácteristica de las funciones es poder usar tuplas, que son elementos que pertenecen al producto cartesiano de dos conjuntos.

Por ejemplo, queremos una función que tome como parámetro un punto (x, y) , los recorra a, b unidades en la entrada x y y respectivamente. Con x, y, a, b de tipo **Float**. La función queda así:

```
recorrePunto :: (Float,Float) -> Float -> Float -> (Float,Float)
recorrePunto (x,y) a b = (x+a,y+b)
```

2.5. Estructura de control if

En ocasiones, para definir una función, necesitamos que la función tome determinados valores si se cumple alguna condición. Por ejemplo, supongamos que queremos una función que dados dos números, nos devuelva el máximo de éstos. Con lo que tenemos en este momento es complicado definir la función mencionada. Para resolver este problema veremos la sentencia **if**, que es una forma obtener un resultado u otro a partir de una condición.

La sintaxis para usar el **if** es la siguiente:

```
if a then b else c
```

Donde **a** debe ser de tipo **Bool**, **b** será el elemento que vamos a devolver si **a** es **True** y **c** es el valor que vamos a devolver si **a** es **False**.

De esta forma ya es más sencillo definir la función que calcula el máximo entre dos elementos de tipo númerico. En este caso los elementos deben ser de tipo **Float**. La función queda de la siguiente forma:

```
maximo :: Float -> Float -> Float
maximo a b = if a > b then a else b
```

Ejercicios.

Para esta práctica programarás algunas funciones en Haskell. Debes usar las firmas que están en cada ejercicio.

1. **Evaluación de una función cuadrática**.La función recibe cuatro parámetros a, b, c y v La función debe calcular calcular la evaluación de valor v en la ecuación $ax^2 + bx + c$ sustituyendo a x por v. **(1 punto)**

   ```
   hipotenusa :: Float -> Float -> Float
   ```

2. **Hipotenusa de un triagulo**. La función debe calcular la hipotenusa de un triángulo con base b y altura h. **(1 punto)**

   ```
   hipotenusa :: Float -> Float -> Float
   ```

3. **Norma de un vector en el plano**. La función debe calcular la norma del vector (x, y). **(1 punto)**

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

4. **Comparador**. La función recibe dos números tipo **Float** a y b. La función debe devolver 0 si los a es igual a b, 1 si a es mayor a b y -1 si a es menor a b. **(1 punto)**

   ```
   comparador :: Float -> Float -> Int
   ```

5. **Suma de fracciones**. La función recibe dos tuplas que representa a las fracciones a/b y c/d. La función debe devolver una tupla con el resultado de sumar las fracciones que recibe comó parámetros. **(1 punto)**

   ```
   sumaFraciones :: (Int,Int) -> (Int,Int) -> (Int,Int)
   ```

6. **Producto punto en el plano**. La Función debe calcular el producto punto entre los vectores (x_1, y_1) y (x_2, y_2). **(1 punto)**

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

7. **Función distancia entre dos puntos en el plano**. La función debe calcular la distancia entre los puntos (x_1, y_1) y (x_2, y_2). **(1 punto)**

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

8. **Pendiente de la recta que pasa por dos puntos**. La función debe calcular la pendiente de la recta que pasa por los puntos (x_1, y_1) y (x_2, y_2). **(1 punto)**

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

9. **Raices de una ecuación cuadratica**. La función recibe como parámetros tres valores de tipo **Float** a, b y c que representan a la ecuación $ax^2 + bx + c = 0$. El resultado debe ser una tupla con las dos raices. Cada raíz es una tupla que representa a un número complejo. Si la ráiz no tiene parte imaginaria, la segunda entrada debe ser 0. **(2 puntos)**

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

10. **Volumen de una pirámide regular**. La función recibe como parámetros tres valores de tipo **Float** l, h y n. El parámetro l representa es el tamaño de los lados de la pirámide, h es la altura y n es el número de lados de la base. La función debe calcular de volumen de la piramide representada por los parámetros l, h y n. **(2 puntos)**

```
volumenPiramidal :: Float -> Float -> Float -> Float
```

Entrega

Deberás entregar un archivo llamado **practica1.hs** (respetando el nombre y la extensión del archivo) junto con el archivo **ReadMe.txt** conforme a los lineamientos de entrega de prácticas. La entrega es a través de la plataforma *classroom*, en la actividad asignada a la tarea. Se tiene como límite las **23:59:59** de la fecha de entrega, cada día de retraso se penalizará con puntos menos.

¡Que tengas éxito en tu primer práctica!