Introducción a los Algoritmos - 1er. cuatrimestre 2012

Guía 2: Definiciones y funciones

Docentes: Araceli Acosta, Javier Blanco, Paula Estrella, Pedro Sánchez Terraf

*El objetivo de los siguientes ejercicios es introducirnos en la **programación funcional**, es decir, al desarrollo de programas como funciones.*

Funciones

En el práctico anterior trabajamos con expresiones aritméticas. Para completar el lenguaje de manera que nos permita escribir programas utilizaremos definiciones de funciones.

1. **Para realizar en grupo:** Leé la información sobre *función matemática* que se encuentra en `Wikipedia`. También consultá en la bibliografía de la materia, Cálculo de Programas, el concepto de función (pags. 12 y 13)para responder a las preguntas.

   a) ¿Qué es una función?

   b) ¿Qué ejemplos de función conocés?

   c) ¿Qué son el *dominio* y *codominio* de una función?

   d) ¿Para qué servirán las funciones en programación?

   e) ¿Qué es la aplicación de una función?

2. En las siguientes definiciones identificá las variables, las constantes y el nombre de la función

   a) $f.x \doteq 5 * x$

   b) $duplica.a \doteq a + a$

   c) $por2.y \doteq 2 * y$

   d) $multiplicar.zz.tt \doteq zz * tt$

3. Escribí una función que dados dos valores, calcule su promedio.

4. Tomando las definiciones del punto 2 evaluá las siguientes expresiones. Justificá cada paso utilizando la notación aprendida. Luego, controlá los resultados en `Haskell`.

   a) $multiplicar.(f.5).2 + 1$

   b) $por2.(duplica.3 + 5)$

5. Tomando las definiciones en el punto 2 demostrá que *duplica* y *por*2 si son aplicadas al mismo valor dan siempre el mismo resultado. En otras palabras, la expresión $duplica.x = por2.x$ es **válida**.

6. Definí las funciones simples que describimos a continuación, luego implementalas en `haskell` . Por ejemplo:

   Enunciado: $signo : Int \to Int$, que dado un entero retorna su signo, de la siguiente forma: retorna 1 si x es positivo, -1 si es negativo y 0 en cualquier otro caso.

   Solución:

   $$\begin{array}{l} sgn : Num \to Num \\ sgn.x \doteq (\ 0 < x \to\ 1 \\ \qquad \Box\, x < 0 \to -1 \\ \qquad \Box\, x = 0 \to\ 0 \\ \qquad) \end{array}$$

   ```
   sgn :: Int -> Int
   sgn x | 0<x   = 1
         | x<0   = -1
         | x==0  = 0
   ```

   En `haskell` los conectivos para las condiciones se pueden escribir así:

$\wedge$	`&&`
$\vee$	`||`
$\neg$	`not`

   a) $entre0y9 : Int \to Bool$, que dado un entero devuelve *True* si el entero se encuentra entre 0 y 9.

   b) $segundo3 : (Int, Int, Int) \to Int$, que dada una terna de enteros devuelve su segundo elemento.

c) *ordena* : $(Int, Int) \to (Int, Int)$, que dados dos enteros los ordena de menor a mayor.

d) *rangoPrecio* : $Int \to String$, que dado un número que representa el precio de una computadora, retorne "muy barato" si el precio es menor a 2000, "demasiado caro" si el precio es mayor que 5000, "hay que verlo bien" si el precio está entre 2000 y 5000, y "esto no puede ser!" si el precio es negativo.

e) *absoluto* : $Int \to Int$, que dado un entero retorne su valor absoluto.

f) *esMultiplo2* : $Int \to Bool$, que dado un entero n devuelve *True* si n es múltiplo de 2.
Ayuda: usar `mod`, el operador que devuelve el resto de la división.

g) *rangoPrecioParametrizado* : $Int \to (Int, Int) \to String$ que dado un número x, que representa el precio de un producto, y un par $(menor, mayor)$ que represente el rango de precios que uno espera encontrar, retorne "muy barato" si x está por debajo del rango, "demasiado caro" si está por arriba del rango, "hay que verlo bien" si el precio está en el rango, y "esto no puede ser!" si x es negativo.

h) *mayor3* : $(Int, Int, Int) \to (Bool, Bool, Bool)$, que dada una una terna de enteros devuelve una terna de valores booleanos que indica si cada uno de los enteros es mayor que 3.
Por ejemplo: $mayor3.(1, 4, 3) = (False, True, False)$; $mayor3.(5, 1984, 6) = (True, True, True)$

Inducción

7. Dado
$f.x.0 = 1$
$f.x.(n+1) = x * f.x.n + 1$
demostrar

a) $f.x.n = \Sigma_{i=0}^{n} x^i$
b) $(x-1) * f.x.n = x^{(n+1)} - 1$

8. Dado
$f.0 = 0$
$f.(n+1) = f.n + 2$
demostrar
$\Sigma_{i=0}^{n} f.i = n * (n+1)$

9. Dado
$f.0 = 0$
$f.1 = 1$
$f.(n+2) = f.(n+1) + 2 * f.n$
probar que
$f.n = 1/3 * (2^n - (-1)^n)$

10. Dado
$f.0 = 2$
$f.1 = 1$
$f.(n+2) = f.(n+1) + 2 * f.n$
probar que
$f.n = 2^n + (-1)^n$