



Universidad
de Huelva



Universidad de Huelva

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 2

EJERCICIOS SOBRE RECURSIVIDAD

Autor: Alberto Fernández Merchán

Asignatura: Modelos Avanzados de Computación

Índice

1. Introducción	2
2. Ejercicios	3
2.1. Operador N sobre K	3
2.1.1. Cláusulas GUARDA	3
2.1.2. Cláusulas IF-ELSE	3
2.1.3. Cláusulas CASE-OF	4
2.1.4. Funciones individuales	4
2.1.5. Función Global	4
2.2. Obtener raíces de una ecuación de segundo grado	5
2.2.1. Cláusulas GUARDA	5
2.2.2. Cláusulas IF-ELSE	5
2.3. Secuencia de Fibonacci	6
2.3.1. Cláusulas GUARDA	6
2.3.2. Cláusulas IF-ELSE	6
2.3.3. Cláusulas CASE-OF	7
2.3.4. Funciones individuales	7
2.4. Pertenencia de un elemento en una lista	8
2.4.1. Cláusulas GUARDA	8
2.4.2. Cláusulas IF-ELSE	8
2.4.3. Cláusula CASE-OF	9
2.4.4. Funciones individuales	9

1. Introducción

En esta práctica se propone resolver algunos problemas utilizando funciones recursivas de diferentes formas:

- Mediante **funciones separadas**: Consiste en declarar varias funciones con el mismo nombre que actúen de forma diferente en función de los parámetros de entrada.
- Mediante **cláusulas guarda**: Se utilizan para definir el resultado de una función dependiendo de si se cumplen las condiciones de cada cláusula.
- Mediante **cláusulas if-else**: Se utiliza para que la función realice diferentes acciones en función de una condición.
- Mediante **cláusulas case of**: Se utiliza para que el resultado de una función varíe en función de un valor.

Debemos resolver cuatro problemas utilizando **recursividad** y de la mayor cantidad de formas vistas en clase (if-else, case-of, guardas...). Los ejercicios que hay que resolver son los siguientes:

1. **Operador N sobre K**: Este ejercicio permite calcular el número de combinaciones de N elementos cogidos de K en K.
2. **Obtener las raíces de una ecuación de segundo grado**: Este ejercicio calcula las raíces de la ecuación $ax^2 + bx + c = 0$.
3. **Obtener el valor correspondiente a un número en la secuencia de fibonacci**: Este ejercicio obtiene el número correspondiente al valor N en la secuencia de fibonacci.
4. **Comprobar si un elemento pertenece a una lista**: Este ejercicio permite comprobar si un número está contenido en una lista o no.

2. Ejercicios

2.1. Operador N sobre K

Se debe obtener la solución para la fórmula:

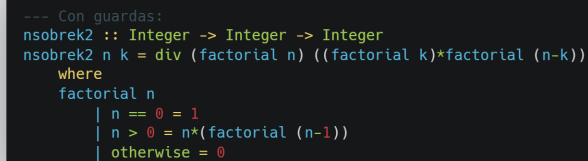
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

La idea es declarar una función local que implemente el factorial de un número para poder calcularla. Esta función se puede definir de diferentes maneras, sin embargo, todas tienen la misma estructura:

- **Caso base:** El factorial de 0 es 1.
- **Caso general:** Para los demás casos, el factorial de n será n multiplicado por el factorial de $n - 1$ ($n! = n * (n - 1)!$)

Podemos declarar dicha función de forma local utilizando la cláusula *where*.

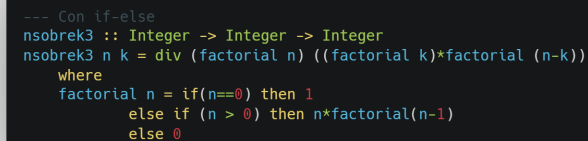
2.1.1. Cláusulas GUARDA



```
--- Con guardas:
nsobrek2 :: Integer -> Integer -> Integer
nsobrek2 n k = div (factorial n) ((factorial k)*factorial (n-k))
  where
    factorial n
      | n == 0 = 1
      | n > 0 = n*(factorial (n-1))
      | otherwise = 0
```

Figura 1: Implementación de nsobrek utilizando guardas

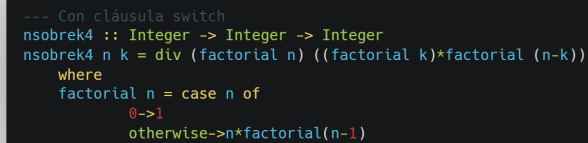
2.1.2. Cláusulas IF-ELSE



```
--- Con if-else
nsobrek3 :: Integer -> Integer -> Integer
nsobrek3 n k = div (factorial n) ((factorial k)*factorial (n-k))
  where
    factorial n = if(n==0) then 1
                  else if (n > 0) then n*factorial(n-1)
                  else 0
```

Figura 2: Implementación de nsobrek utilizando if-else

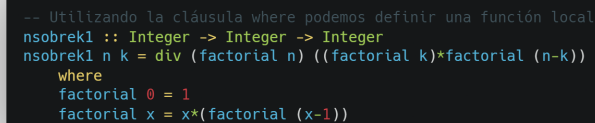
2.1.3. Cláusulas CASE-OF



```
--- Con cláusula switch
nsobrek4 :: Integer -> Integer -> Integer
nsobrek4 n k = div (factorial n) ((factorial k)*factorial (n-k))
  where
    factorial n = case n of
      0->1
      otherwise->n*factorial(n-1)
```

Figura 3: Implementación de nsobrek utilizando case-of

2.1.4. Funciones individuales

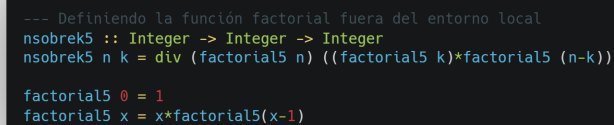


```
-- Utilizando la cláusula where podemos definir una función local
nsobrek1 :: Integer -> Integer -> Integer
nsobrek1 n k = div (factorial n) ((factorial k)*factorial (n-k))
  where
    factorial 0 = 1
    factorial x = x*(factorial (x-1))
```

Figura 4: Implementación de nsobrek utilizando varias funciones locales

2.1.5. Función Global

En este apartado hemos declarado la función **factorial** como una función global e vez de local.



```
--- Definiendo la función factorial fuera del entorno local
nsobrek5 :: Integer -> Integer -> Integer
nsobrek5 n k = div (factorial5 n) ((factorial5 k)*factorial5 (n-k))

factorial5 0 = 1
factorial5 x = x*factorial5(x-1)
```

Figura 5: Implementación de nsobrek utilizando una función global

2.2. Obtener raíces de una ecuación de segundo grado

En este ejercicio se debe resolver la ecuación:

$$ax^2 + bx + c = 0 \rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Debemos tener en cuenta que la mayoría de ecuaciones de segundo grado tienen dos soluciones, sin embargo, no siempre es así ya que podría ser la misma o incluso no tener solución en los números reales. Por ello debemos tener en cuenta estos casos y resolver la ecuación de forma distinta o indicar que no tiene solución.

2.2.1. Cláusulas GUARDA

```
-- cláusula guarda
raices1::(Floating x, Ord x) => x -> x -> x -> [x]
raices1 a b c
  | (a == 0) && (b /= 0) = [(-c)/b]
  | (a == 0) && (b == 0) = error "No hay variable X"
  | (b*b - 4*a*c) < 0 = error "La raíz es negativa"
  | otherwise = unico [(-b + sqrt (b*b - 4*a*c)) / (2*a), (-b - sqrt (b*b -
4*a*c)) / (2*a)]
  where
    unico [] = []
    unico lista = (head lista):unico (filter (\x -> x /= (head lista)) lista)
```

Figura 6: Implementación de raíces utilizando guardas

2.2.2. Cláusulas IF-ELSE

```
-- cláusulas if-else
raices2::(Floating x, Ord x) => x -> x -> x -> [x]
raices2 a b c = if (a == 0) && (b /= 0) then [(-c)/b]
  else if (a == 0) && (b == 0) then error "No hay variable X"
  else if (b*b - 4*a*c) < 0 then error "La raíz es negativa"
  else unico [(-b + sqrt (b*b - 4*a*c)) / (2*a), (-b - sqrt (b*b - 4*a*c))
)/(2*a)]
  where
    unico [] = []
    unico (h:hs) = h:unico (filter (\x -> x /= h) (hs))
```

Figura 7: Implementación de raíces utilizando if-else


2.3. Secuencia de Fibonacci

Obtener el número correspondiente a la sucesión de Fibonacci. Siendo esta sucesión la siguiente

$$\begin{cases} f_0 &= 1 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \end{cases}$$

En este ejercicio tenemos los casos base definidos por la propia función a trozos. Solo debemos tenerlos en cuenta a la hora de describir como se comporta nuestra función.

2.3.1. Cláusulas GUARDA



```
--- Mediante cláusulas guarda
fibonacci1 :: Integer -> Integer
fibonacci1 n
  | n == 0 = 0
  | n == 1 = 1
  | otherwise = (fibonacci1 (n-1)) + (fibonacci1 (n-2))
```

Figura 8: Implementación de Fibonacci utilizando guardas

2.3.2. Cláusulas IF-ELSE



```
--- Mediante cláusulas if-else
fibonacci2 :: Integer -> Integer
fibonacci2 n = if ( n == 0 ) then 0
               else if ( n == 1 ) then 1
               else (fibonacci2 (n-1)) + (fibonacci2 (n-2))
```

Figura 9: Implementación de Fibonacci utilizando if-else

2.3.3. Cláusulas CASE-OF



```
--- Mediante cláusulas case
fibonacci3 :: Integer -> Integer
fibonacci3 n = case n of
  0 -> 0
  1 -> 1
  otherwise -> (fibonacci3 (n-1)) + (fibonacci3 (n-2))
```

Figura 10: Implementación de Fibonacci utilizando case-of

2.3.4. Funciones individuales



```
--- Mediante funciones
fibonacci4 :: Integer -> Integer
fibonacci4 0 = 0
fibonacci4 1 = 1
fibonacci4 n = (fibonacci4 (n-1)) + (fibonacci4 (n-2))
```

Figura 11: Implementación de Fibonacci utilizando funciones individuales

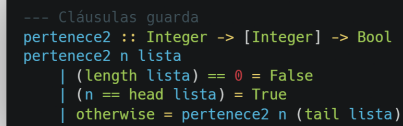
2.4. Pertenencia de un elemento en una lista

Utilizando recursividad, comprobar si un elemento está o no en una lista.

La estructura de este algoritmo recursivo es:

- **Caso base:** La lista vacía no contiene al elemento que se busca.
- **Caso general:** Si el elemento es el mismo que el primero de la lista pertenece a la lista, en caso contrario, repetir con la cola de la lista.

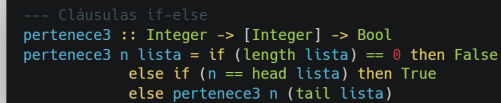
2.4.1. Cláusulas GUARDA



```
--- Cláusulas guarda
pertenece2 :: Integer -> [Integer] -> Bool
pertenece2 n lista
  | (length lista) == 0 = False
  | (n == head lista) = True
  | otherwise = pertenece2 n (tail lista)
```

Figura 12: Implementación de pertenece utilizando guardas

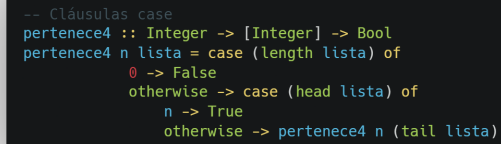
2.4.2. Cláusulas IF-ELSE



```
--- Cláusulas if-else
pertenece3 :: Integer -> [Integer] -> Bool
pertenece3 n lista = if (length lista) == 0 then False
                     else if (n == head lista) then True
                     else pertenece3 n (tail lista)
```

Figura 13: Implementación de pertenece utilizando if-else

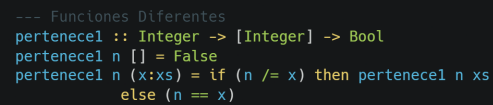
2.4.3. Cláusula CASE-OF



```
-- Cláusulas case
pertenece4 :: Integer -> [Integer] -> Bool
pertenece4 n lista = case (length lista) of
  0 -> False
  otherwise -> case (head lista) of
    n -> True
    otherwise -> pertenece4 n (tail lista)
```

Figura 14: Implementación de pertenece utilizando case-of

2.4.4. Funciones individuales



```
--- Funciones Diferentes
pertenece1 :: Integer -> [Integer] -> Bool
pertenece1 n [] = False
pertenece1 n (x:xs) = if (n /= x) then pertenece1 n xs
                     else (n == x)
```

Figura 15: Implementación de pertenece utilizando funciones individuales