Universidad Tecnológica Nacional
FACULTAD REGIONAL CORDOBA

PARADIGMAS DE PROGRAMACION

Unidad IV
Paradigma Funcional
Parte II

CONTENIDOS ABORDADOS

- Expresiones if-then-else, case y con guardas.
- Definiciones locales: expresiones let y where.
- Disposición de código: Layout.

Expresiones if-then-else

- En Haskell, no hay estructuras de control, pero si existe la función especial if – then – else, que implica otro modo de escribir expresiones cuyo resultado depende de cierta condición:

```
if exprBool
      then exprSi
      else exprNo
```

- Si la expresión booleana *exprBool* es True devuelve *exprSi*, si es False, devuelve *exprNo*.
- Una expresión de ese tipo sólo es aceptable si *exprBool* es de tipo Bool y *exprSi* y *exprNo* son del mismo tipo.

Expresiones if-then-else

- Ejemplos:
- En el interprete:

```
Hugs> if 5 > 2 then 10.0 else (10.0/0.0)
10.0 :: Double

Hugs> 2 * if 'a'<'z' then 10 else 4
20 :: Integer
```

Expresiones if-then-else

- Ejemplos, definiendo funciones:

  Mostrar la paridad de un número entero:

```
paridad :: Int -> String
paridad x = if (x `mod` 2) == 0
                    then "par"
                    else "impar"
```

  Mostrar el mayor de dos números:

```
max:: Int -> Int -> Int
max x y = if x >= y then x else y
```

  La evaluación sería:

```
Main> max 3 7
7 :: Integer
```

Expresiones case

- Una expresión case puede ser utilizada para evaluar una expresión y, dependiendo del resultado, devolverá uno de los posibles valores.

```
case (expresión) of
                valor1 -> resultado1
                valor2 -> resultado2
                valor3 -> resultado3
                otherwise -> otroresultado
```

- También se puede reemplazar otherwise por el termino _.

Expresiones case

- Ejemplo:

```
paridad :: Int -> String
paridad x = case (x `mod` 2) of
                  0 -> "par"
                  1 -> "impar"
```

Al evaluar la función:

```
Main> paridad 3
"impar" :: [Char]
```

Expresiones case

- Ejemplo:

```
--codigo 1(Basico), 2 (Completo) y 3 (Super)
tipo :: Int -> String
tipo cod = case (cod) of
                  1 -> "Basico"
                  2 -> "Completo"
                  3 -> "Super"
                  _ -> "Codigo No existe"
```

Al evaluar la función:

```
Main> tipo 3
"Super" :: [Char]
```

Expresiones con guardas

- Cada una de las ecuaciones de una definición de función podría contener **guardas** que requieren que se cumplan ciertas condiciones sobre los valores de los argumentos.
- En general una ecuación con guardas toma la forma:

```
f x1 x2 ... xn | condicion1 = e1
               | condicion2 = e2
                    .
                    .
               | condicionm = em
               | otherwise = exp
```

Expresiones con guardas

- Ejemplos:

```
paridad :: Int -> String
paridad x | ((x `mod` 2) == 0) = "par"
          | ((x `mod` 2) == 1) = "impar"
```

Al evaluar la función:

```
Main> paridad (3 + 4)
 "impar" :: [Char]
```

Expresiones con guardas

Otro ejemplo: codificar el rango de un numero:

[0 – 100] = 1, [101 – 200] = 2, [201-300] = 3, fuera de rango = 0

```
rango :: Int -> Int
rango num |(num >= 0 && num <=100) = 1
          |(num >= 101 && num <=200) = 2
          |(num >= 201 && num <=300) = 3
          | otherwise = 0
```

Al evaluar la función:

```
Main> rango 105
2 :: Int
Main> rango 500
0 :: Int
```

Definiciones locales

Expresiones Let

- Las *expresiones let* de Haskell son útiles cuando se requiere un conjunto de declaraciones locales. Se introducen en un punto arbitrario de una expresión utilizando una expresión de la forma:

```
let <decls> in <expr>
```

Ejemplo con una declaración:

```
Hugs> let x = 1 + 4 in x*x + 3*x + 1
41 :: Integer
```

Ejemplo con una función:

```
sumaLet::Int -> Int -> Int
sumaLet x y  = let z = 2 in z + x +y
```

Definiciones locales

Expresiones Where

- A veces es conveniente establecer declaraciones locales en una ecuación con varias con guardas, lo que podemos obtener a través de una *cláusula where*:

```
f x y | y > (x*x*x) = 1
      | y == (x*x*x) = 0
      | y < (x*x*x) = -1
```

```
f x y | y > z = 1
      | y == z = 0
      | y < z = -1
      where z = (x*x*x)
```

Definiciones locales

Expresiones Where Ejemplo

```
{-Dados dos argumentos la función retorna 1 si el 2° argumento es
    mayor que el cuadrado del 1°, 0 si el 2° argumento es igual que el
    cuadrado del 1° y -1 1 si el 2° argumento es menor que el cuadrado
    del 1° -}

comparaCuadrado::Integer -> Integer -> Integer
comparaCuadrado x y |y > z  = 1
                    |y == z = 0
                    |y < z  = -1
                    where z = x*x
```

Si evaluamos:

```
Main> comparaCuadrado 5 3
    -1 :: Integer
```

Disposición de código: Layout

- Para marcar el final de una ecuación, una declaración, etc. Se utiliza una sintaxis bidimensional denominada *espaciado (layout)* que se basa esencialmente en que las declaraciones están alineadas por columnas.

- Ejemplo:

```
porcComision(tipoV) =  case tipoV of
                                1 -> 15
                                2 -> 20
                                3 -> 30
                                otherwise -> -1
```

Disposición de código: Layout

Reglas :

- El siguiente caracter de cualquiera de las palabras clave (where, let, o of) es el que determina la columna de comienzo de declaraciones en las expresiones where, let, o case correspondientes.

- Es necesario asegurarse que la columna de comienzo dentro de una declaración está más a la derecha que la columna de comienzo de la siguiente cláusula.

Disposición de código: Layout

Por ejemplo, dada la siguiente expresión:

```
--usando layout
ejemplo1::Int
ejemplo1  = (a +a)
              where a = 6
                    --asigna el valor 6
ejemplo2::Int
ejemplo2  = (a + b)
                   where
                       a = 6
                       b = 5
```