Tema 2. Tipos predefinidos

2.1 Tipos simples predefinidos

El tipo Bool

✓ Los valores de este tipo representan expresiones lógicas cuyo resultado puede ser verdadero o falso.

✓ Solo hay dos valores para el tipo: $True$ y $False$.

Funciones y operadores

- $(\&\&) :: Bool \to Bool \to Bool$ conjunción lógica.
- $(||) :: Bool \to Bool \to Bool$ disyunción lógica.
- $not :: Bool \to Bool$ negación lógica.
- $otherwise :: Bool$ función constante que devuelve el valor $True$.

Comportamiento de las funciones anteriores:

$v1$	$v2$	$v1$ && $v2$	$v1$ || $v2$
$True$	$True$	$True$	$True$
$True$	$False$	$False$	$True$
$False$	$True$	$False$	$True$
$False$	$False$	$False$	$False$

v	$not\ v$
$True$	$False$
$False$	$True$

? $True$ && $False$
$False :: Bool$

? $not\ (True$ && $False)$
$True :: Bool$

El tipo Int

✓ Números enteros de precisión limitada que cubren al menos el rango $[-2^{29}, 2^{29}-1]$.

Funciones y operadores

- $(+), (-), (*) \ :: \ Int \to Int \to Int$. Suma, resta y producto de enteros.
- $(\hat{})$ $:: \ Int \to Int \to Int$. Operador potencia. El exponente debe ser mayor o igual a cero.
- $div, \ mod \ :: \ Int \to Int \to Int$. Cociente y resto de dividir dos enteros.
- $abs \ :: \ Int \to Int$. Valor absoluto.
- $signum \ :: \ Int \to Int$. devuelve +1, -1 o 0, según el signo del entero argumento.
- $negate \ :: \ Int \to Int$. Invierte el signo de su argumento. También puede usarse un signo menos prefijo.
- $even, \ odd \ :: \ Int \to Bool$. Comprueban la naturaleza par o impar de un número.

El tipo Integer

- ✓ Los valores de este tipo son números enteros de precisión ilimitada.
- ✓ Para los valores del tipo *Integer* están disponibles las mismas operaciones que para el tipo *Int*.
- ✓ Los cálculos con datos de tipo *Integer* son menos eficientes que con datos de tipo *Int*.

```
? 2^100
1267650600228229401496703205376 :: Integer

? 111111111 * 111111111
12345678987654321 :: Integer
```

El tipo Float

✓ Subconjunto de un intervalo de los números reales.

✓ Hay dos modos de escribir valores reales:

- ◇ La notación habitual: Por ejemplo, $1.35,\ -15.345,\ 1.0,\ 1$
- ◇ La notación científica: Por ejemplo, $1.5e7,\ 1.5e-17$

Funciones y operadores

- $(+), (*), (-), (/)\ ::\ Float \to Float \to Float$. Suma, producto, resta y división de reales
- $(\hat{})\ ::\ Float \to Int \to Float$. Potencia, de base real, pero exponente entero y positivo.
- $(**)\ ::\ Float \to Float \to Float$. Potencia, de base y exponente real.
- $abs\ ::\ Float \to Float$. Valor absoluto.
- $signum\ ::\ Float \to Float$. Devuelve -1.0, 0.0 ó +1.0 dependiendo del signo del real argumento.
- $negate\ ::\ Float \to Float$. Devuelve el valor del real argumento negado. Puede usarse también el signo menos prefijo.

Funciones y operadores (2)

- sin, $asin$, cos, $acos$, tan, $atan$ :: $Float \to Float$. Funciones trigonométricas (trabajan con radianes)
- $atan2$ :: $Float \to Float \to Float$. $atan2\ x\ y$ devuelve la arcotangente de $\frac{x}{y}$.
- log, exp :: $Float \to Float$. Funciones logarítmicas y exponenciales.
- $sqrt$ :: $Float \to Float$. Raíz cuadrada.
- pi :: $Float$. El valor del número π.
- $truncate$, $round$, $floor$ y $ceiling$ :: $Float \to Integer$ o $Float \to Int$. Funciones de redondeo.
- $fromInt$ :: $Int \to Float$ y $fromInteger$ :: $Integer \to Float$. Funciones de conversión de tipo.

El tipo Double

✓ Se trata de un subconjunto de un intervalo de los números reales.

✓ El subconjunto es mayor que el correspondiente al tipo $Float$ y las aproximaciones más precisas.

✓ Todas las operaciones disponibles para el tipo $Float$ están también disponibles para el tipo $Double$.

El tipo Char

- ✓ Un valor de tipo $Char$ representa un carácter (una letra, un dígito, un signo de puntuación, etc.).
- ✓ Un valor constante de tipo carácter se escribe entre comillas simples. `'a'`, `'1'`, `'?'`
- ✓ Algunos caracteres especiales se escriben precediéndolos del carácter \:
  - ◇ `'\n'` es el carácter de salto de línea.
  - ◇ `'\t'` es el carácter tabulador.
  - ◇ `'\''` es el carácter comilla.
  - ◇ `'\"'` es el carácter comilla doble.
  - ◇ `'\\'` es el carácter \.

Funciones

- $ord :: Char \to Int$. código ASCII del carácter argumento.
- $chr :: Int \to Char$. Función inversa a la anterior.
- $isUpper$, $isLower$, $isDigit$, $isAlpha :: Char \to Bool$. Comprueban si un carácter es una letra mayúscula, minúscula, un dígito o una letra.
- $toUpper$, $toLower :: Char \to Char$. Convierten un carácter a mayúscula o minúscula.

Operadores de igualdad y orden

✓ Para todos los tipos básicos comentados están definidos los siguientes *operadores binarios*, que devuelven un valor booleano:

(>)	mayor que
(>=)	mayor o igual que
(<)	menor que
(<=)	menor o igual que
(==)	igual que
(/=)	distinto que

✓ El tipo de los dos argumentos debe ser el mismo (no se pueden comparar valores de tipos distintos).

Ejemplos

```
? 10 <= 15
True :: Bool

? 'x' == 'y'
False :: Bool

? 'x' /= 'y'
True :: Bool

? True < 'a'
ERROR                : Type error in application
*** Expression      : True < 'a'
*** Term            : True
*** Type            : Bool
*** Does not match  : Char
```

✓ Para el tipo *Char* el orden viene dado por el código ASCII del carácter.

✓ Para el tipo *Bool*, el valor *False* se considera menor que *True*.

2.2 Constructores de tipo predefinidos

✓ Haskell define *tipos estructurados* que permiten representar colecciones de objetos.

Tuplas

✓ Una *tupla* es un dato compuesto donde el tipo de cada componente puede ser distinto.

Tuplas

Si $v_1, v_2, \ldots, v_n$ son valores con tipo $t_1, t_2, \ldots, t_n$
entonces $(v_1, v_2, \ldots, v_n)$ es una tupla con tipo $(t_1, t_2, \ldots, t_n)$

Ejemplos:

```
? ()
() :: ()

? ('a', True)
('a', True) :: (Char, Bool)

? ('a', True, 1.5)
('a', True, 1.5) :: (Char, Bool, Double)
```

Las tuplas son útiles cuando una función tiene que devolver más de un valor.

```
predSuc     :: Integer -> (Integer, Integer)
predSuc x   =  (x - 1, x + 1)
```

Listas

✓ Una *lista* es una colección de cero o más elementos **todos del mismo tipo**.

Hay dos constructores para listas:

- [] Representa la lista vacía (lista con cero elementos).
- (:) Permite añadir un elemento a principio de una lista. Si xs es una lista con n elementos, y x es un elemento, entonces $x : xs$ es una lista con $n + 1$ elementos.

Listas

Si $v_1, v_2, \ldots, v_n$ son valores con tipo t
entonces $v_1 : (v_2 : (\ldots (v_{n-1} : (v_n : [])))$ es una lista
con tipo $[t]$

Ejemplos:

- `1 : []` Una lista que almacena un único entero. Tiene tipo $[Integer]$.
- `3 : (1 : [])` Una lista que almacena dos enteros. El valor `3` ocupa la primera posición dentro de la lista. El valor `1` la segunda.
- `'a' : (1 : [])` Es una expresión errónea (produce un error de tipos).

Listas (2)

✓ El constructor $(:)$ es asociativo a la derecha:

Asociatividad derecha de $(:)$

$$x_1 : x_2 : \ldots x_{n-1} : x_n : [\,] \Longrightarrow x_1 : (x_2 : (\ldots (x_{n-1} : (x_n : [\,]))))$$

Aún así, la notación sigue siendo engorrosa.

✓ Haskell permite una sintaxis para listas más cómoda:

Sintaxis para listas

$$[x_1, x_2, \ldots x_{n-1}, x_n] \Longrightarrow x_1 : (x_2 : (\ldots (x_{n-1} : (x_n : [\,]))))$$

Tres modos de escribir la misma lista:

```
? 1 : (2 : (3 : []))
[1,2,3] :: [Integer]

? 1 : 2 : 3 : []
[1,2,3] :: [Integer]

? [1,2,3]
[1,2,3] :: [Integer]
```

Cadenas de caracteres (Strings)

✓ Una *cadena de caracteres* es una secuencia de cero o más caracteres.

✓ En Haskell, las cadenas de caracteres son listas de caracteres.

✓ El tipo asociado a las cadenas de caracteres es $String$ (un modo equivalente de escribir el tipo $[Char]$).

✓ Haskell permite una sintaxis más cómoda para escribir cadenas de caracteres: escribir el texto entre comillas dobles:

Cadenas de caracteres

$$"x_1\, x_2\, \ldots\, x_{n-1}\, x_n" \Longrightarrow ['x_1', 'x_2', \ldots 'x_{n-1}', 'x_n']$$

Ejemplos:

```
? 'U' : 'n' : ' ' : 'C' : 'o' : 'c' : 'h' : 'e' : []
  "Un Coche"   :: [Char]

? ['U' , 'n' , ' ' , 'C' , 'o' , 'c' , 'h' , 'e']
  "Un Coche"   :: [Char]

?  "Un Coche"
  "Un Coche"   :: String
```

El constructor de tipo ($\rightarrow$)

✓ Es posible declarar el tipo correspondiente a las distintas funciones. Para ello disponemos de un único constructor: ($\rightarrow$).

Tipos Funcionales

Si $t_1, t_2, \ldots, t_n, t_r$ son tipos válidos
entonces $t_1 \rightarrow t_2 \rightarrow \ldots t_n \rightarrow t_r$ es el tipo
de una función con n argumentos

El tipo del resultado es t_r

Ejemplos:

```
inc     :: Integer -> Integer
inc x = x + 1
```

```
esCero     :: Integer -> Bool
esCero x =  (x == 0)
```

```
sumaCuadrados       :: Integer -> Integer -> Integer
sumaCuadrados x y = x ^ 2 + y ^ 2
```

Objetivos del tema

El alumno debe:

- ✓ Conocer los distintos tipos simples predefinidos
- ✓ Conocer las distintas funciones y operadores predefinidos para cada tipo
- ✓ Conocer los tipos estructurados predefinidos