Predefined.pdf

realGCabrones

Estructuras de Datos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Useful Predefined Functions and Operators in Haskell

@ José E. Gallardo

Functions and operators	Description
`(!!) :: [a] -> Int -> a`	Element at index in list. Index for first element is 0
`(&&) :: Bool -> Bool -> Bool`	Logical conjunction (and)
`(-) :: (Num a) => a -> a -> a`	Subtraction
`(*) :: (Num a) => a -> a -> a`	Multiplication
`(**) :: (Floating a) => a -> a -> a`	Raising to a real exponent
`(/) :: (Fractional a) => a -> a -> a`	Real division
`(/=) :: (Eq a) => a -> a -> Bool`	Inequality
`(:) :: a -> [a] -> [a]`	Prepends element to a list
`(^) :: (Num a, Integral b) => a -> b -> a`	Raising to natural exponent
`(||):: Bool -> Bool -> Bool`	Logical disjunction (or)
`(+) :: (Num a) => a -> a -> a`	Addition
`(.) :: (b -> c) -> (a -> b) -> (a -> c)`	Function composition
`(++) :: [a] -> [a] -> [a]`	List concatenation
`(<), (<=), (>), (>=) :: (Ord a) => a -> a -> Bool`	Order relations
`(==) :: (Eq a) => a -> a -> Bool`	Equality
`abs :: (Num a) => a -> a`	Absolute value
`all :: (a -> Bool) -> [a] -> Bool`	Do all elements in list fulfil a predicate?
`and :: [Bool] -> Bool`	Logical conjunction (and) over a list of Booleans
`any :: (a -> Bool) -> [a] -> Bool`	Does any element in list fulfil a predicate?
`ceiling :: (RealFrac a, Integral b) => a -> b`	Real to Integral conversion (returns least integer not less than argument)
`chr :: Int -> Char (import Char)`	UNICODE code to character. Inverse to `chr`
`concat :: [[a]] -> [a]`	Concatenates a list of lists
`const :: a -> b -> a`	Constant function (`const x` is a 1 argument function that always returns x)
`cos :: (Floating a) => a -> a`	Cosine (uses radians)
`curry :: ((a, b) -> c) -> (a -> b -> c)`	Turns a function taking a 2-element tuple into a curried one
`cycle :: [a] -> [a]`	Infinite list repeating forever argument list
`div :: (Integral a) => a -> a -> a`	Integer division
`divMod :: (Integral a ) => a -> a -> (a, a)`	Integer division and modulo
`drop :: Int -> [a] -> [a]`	Eliminates first n elements from a list
`dropWhile :: (a -> Bool) -> [a] -> [a]`	Eliminates longest prefix of elements in a list fulfilling a predicate
`elem :: (Eq a) => a -> [a] -> Bool`	Is a value in a list?
`even :: (Integral a) => a -> Bool`	Is an integral value multiple of 2?
`exp :: (Floating a) => a -> a`	Exponential function (base is *e*)
`filter :: (a -> Bool) -> [a] -> [a]`	Retains elements in list fulfilling predicate
`flip :: (a -> b -> c) -> (b -> a -> c)`	Turns a 2-argument function in another one taking arguments in reverse order
`floor :: (RealFrac a, Integral b) => a -> b`	Real to integral conversion (returns greatest integer not greater than argument)
`foldl :: (b -> a -> b) -> b -> [a] -> b`	Reduces elements in a list from left to right using a binary function and an initial value
`foldr :: (a -> b -> b) -> b -> [a] -> b`	Reduces elements in a list from right to left using a binary function and an initial value
`fromIntegral :: (Integral a, Num b) => a -> b`	Integral to another numeric type conversion
`fst :: (a,b) -> a`	First component in a 2-element tuple
`gcd :: (Integral a) => a -> a -> a`	Greatest common divisor
`head :: [a] -> a`	First element in a list
`id :: a -> a`	Identity function (returns its argument)
`init :: [a] -> [a]`	Eliminates last element in a list
`iterate :: (a -> a) -> a -> [a]`	Successively iterates a function from an initial value
`last :: [a] -> a`	Last element in a list
`lcm :: (Integral a) => a -> a -> a`	Least common multiple

`length :: [a] -> Int`	Number of elements in a list
`log :: (Floating a) => a -> a`	Natural logarithm (to the base *e*)
`logBase :: (Floating a) => a -> a -> a`	Logarithm to some base (1st argument)
`map :: (a -> b) -> [a] -> [b]`	Applies some function to all elements in a list
`maximum :: (Ord a) => [a] -> a`	Maximum element in a list
`min :: (Ord a) => a -> a -> a`	Minimum of two values
`minimum :: (Ord a) => [a] -> a`	Minimum element in a list
`mod :: (Integral a) => a -> a -> a`	Integer division modulo
`not :: Bool -> Bool`	Logical negation
`null :: [a] -> Bool`	Is the list empty?
`odd :: (Integral a) => a -> Bool`	Is an integral value non-multiple of 2?
`or :: [Bool] -> Bool`	Logical disjunction (or) over a list of Booleans
`ord :: Char -> Int` `(import Char)`	Character to UNICODE CODE. Reverse to `ord`
`pi :: (Floating a) => a`	Π
`read :: (Read a) => String -> a`	String to value conversion. Inverse to `Show`
`reverse :: [a] -> [a]`	List with same elements in opposite order
`repeat :: a -> [a]`	Infinite list repeating same value
`replicate :: Int -> a -> [a]`	List with a value repeated a number of times
`round :: (RealFrac a, Integral b) => a -> b`	Real to integral rounded conversion
`show :: (Show a) => a -> String`	Value to String conversion. Inverse to `read`
`sin :: (Floating a) => a -> a`	Sine (uses radians)
`snd :: (a,b) -> b`	Second component in a 2-element tuple
`sort :: (Ord a) => [a] -> [a]` `(import List)`	Sorts a list in ascending order
`span :: (a -> Bool) -> [a] -> ([a], [a])`	Combines results from `takeWhile` and `dropWhile`
`splitAt :: Int -> [a] -> ([a], [a])`	Combines results from `take` and `drop`
`sqrt :: (Floating a) => a -> a`	Square root
`tail :: [a] -> [a]`	Eliminates first element in list
`tan :: (Floating a) => a -> a`	Tangent (uses radians)
`take :: Int -> [a] -> [a]`	First n elements in a list
`takeWhile :: (a -> Bool) -> [a] -> [a]`	Longest prefix of elements in list fulfilling predicate
`truncate :: (RealFrac a, Integral b) => a -> b`	returns the integer nearest argument between zero and argument
`uncurry :: (a -> b -> c) -> ((a,b) -> c)`	Turns a 2-argument curried function into one on tuples
`unzip :: [(a, b)] -> ([a], [b])`	Transforms a list of pairs into a list of first components and a list of second components. Inverse to `zip`
`unzip3 :: [(a, b, c)] -> ([a], [b], [c])`	Takes a list of triples and returns three lists. Inverse to `zip3`
`words :: String -> [String]`	Breaks a string into words (delimited by whitespaces)
`zip :: [a] -> [b] -> [(a,b)]`	Takes two lists and returns a list of corresponding pair
`zip3 :: [a] -> [b] -> [c] -> [(a, b, c)]`	Takes three lists and returns a list of corresponding triples
`zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`	Operates pairwise elements in two lists
`zipWith3 :: (a->b->c->d) -> [a] -> [b] -> [c] -> [d]`	Operates triplewise elements in three lists

Priority	Associativity	Predefined Operators
10	left	Function application
9	left	!!
9	right	.
8	right	^ ^^ **
7	left	* /
6	left	+ -
5	right	: ++
4	non associative	== /= < <= > >=
3	right	&&
2	right	||
1	left	>> >>=
1	right	=<<
0	right	$ $!