Vordefinierte Typen, Typkonstruktoren und Datenkonstruktoren

Typkonstruktor	Datenkonstruktor(en)	Beschreibung
`Int`	0, 1, 2, ...	ganze Zahlen
`Float`	0.0 ...	Gleitkommazahlen
`Char`	' ', '!', ..., 'A', 'B', ...	ASCII Zeichen
`Bool`	`True, False`	Wahrheitswerte
`[ _ ]`	`[ ], [` el_1`, ..., ` el_n`]`	Listen
	`[ ], _ : _`	Listenkonstruktoren
`(_, ..., _)`	`(_, ..., _)`	Paare, Tupel
`_ -> _`	(entfällt)	Funktionen

Für Konstanten des Typs `[Char]` kann anstelle von `['s', 't', 'r', 'i', 'n', 'g']` auch die Schreibweise `"string"` benutzt werden. Der Typkonstruktor `->` assoziiert nach rechts, d.h. `a -> b -> c` ist gleichbedeutend zu `a -> (b -> c)`.

Vordefinierte Funktionen

Funktion	Typ	Beschreibung
`not`	`Bool -> Bool`	logische Negation
`ord`	`Char -> Int`	Codenummer im ASCII Zeichensatz
`chr`	`Int -> Char`	invers zu `ord`
`show`	`Text a => a -> String`	Umwandlung von Druckbarem in String
`error`	`String -> alpha`	Erzeugt einen Laufzeitfehler
`fst`	`(a, b) -> a`	erste Komponente eines Paares
`snd`	`(a, b) -> b`	zweite Komponente eines Paares
	arithmetische Operationen	
`abs`	`(Num a, Ord a) => a -> a`	Absolutbetrag
`acos`	`Float -> Float`	Arcus Cosinus
`asin`	`Float -> Float`	Arcus Sinus
`atan`	`Float -> Float`	Arcus Tangens
`atan2`	`Float -> Float -> Float`	`atan2 y x`: Arcus Tangens von `y/x`
`cos`	`Float -> Float`	Cosinus
`exp`	`Float -> Float`	Exponentialfunktion
`log`	`Float -> Float`	Logarithmus zur Basis e
`log10`	`Float -> Float`	Logarithmus zur Basis 10
`negate`	`Num a => a -> a`	Vorzeichenwechsel
`pi`	`Float`	die Zahl π
`signum`	`(Num a, Ord a) => a -> Int`	Vorzeichen
`sin`	`Float -> Float`	Sinus
`sqrt`	`Float -> Float`	Quadratwurzel

Vordefinierte Operatoren

In der folgenden Tabelle der vordefinierten Infixoperatoren geht die Assoziativität der Operatoren aus der Spalte „A" hervor. Ein r steht für rechtassoziativ, n für nicht-assoziativ und l für linksassoziativ.

Priorität	A	Operator	Beschreibung, Typ
stark	l	`_ _`	Funktionsanwendung
	r	`if, let`, Lambda (\), `case`	(nach links)
	r	`case`	(nach rechts)
Infix 9	r	`.`	Funktionskomposition
	l	`!!`	Zugriff auf Listenelement
Infix 8	r	`^`	Exponentiation
Infix 7	l	`*`	Multiplikation
	n	`/, 'div', quot`	Division
	n	`'mod', 'rem'`	Rest
Infix 6	l	`+, -`	Addition, Subtraktion
Infix 5	r	`++`	Listenverkettung, `[a] -> [a] -> [a]`
	r	$:_r$	Listenkonstruktion, `a -> [a] -> [a]`
	n	`\\`	Listendifferenz, `[a] -> [a] -> [a]`
Infix 4	l	`'elem', 'notElem'`	Elementtest
			`Eq a => a -> [a] -> Bool`
	n	`/=, ==`	(Un-) Gleichheit
			`Eq a => a -> a -> Bool`
	n	`<, <=, >, >=`	Vergleichsoperationen
			`Ord a => a -> a -> Bool`
Infix 3	r	`&&`	logisches Und, `Bool -> Bool -> Bool`
Infix 2	r	`||`	logisches Oder, `Bool -> Bool -> Bool`
Infix 1			
Infix 0			
schwach	r	`->`	Funktionstypen
	—	`=>`	Kontexte
	—	`::`	Typeinschränkungen
	r	`if, let`, Lambda (\)	(nach rechts)
	—	`..`	Sequenzen
	—	`<-`	Generatoren
	—	`(_, ..., _)`	Tupelkonstruktion
	—	`|`	Bedingungen, Wachen
	—	`->`	Alternativen (`case`)
	—	`=`	Definitionen
	—	`;`	Trennung von Deklarationen