IB015 Haskell Cheat Sheet

GHCi

`:q[uit]`	vypne GHCi
`:l[oad] file.hs`	načte soubor `file.hs`
`:r[eload]`	znovu načte aktuální soubor
`:i[nfo] name`	vypíše informace o objektu `name`
`:t[ype] expr`	vypíše typ výrazu `expr`
`:m[odule] m`	načte modul `m`

Základní typy

`Bool`	logická hodnota (`True`, `False`)
`Char`	znak (např. `'a'`)
`String`	řetezec (např. `"abc"`; synonymum `[Char]`)
`Int`	celé číslo omezené velikosti
`Integer`	libovolně velké/malé celé číslo
`Float`	reálné číslo s plovoucí desetinnou čárkou
`Double`	přesnější verze `Float`

Složené typy

`(a,b)`	dvojice hodnot typů `a`, `b`
`(a,b,c)`	trojice hodnot typů `a`, `b`, `c`
`[a]`	seznam hodnot typu `a`
`a -> b -> c`	funkce s argumenty typů `a`, `b`, vrací `c`

Typové třídy

`Eq a`	lze testovat na rovnost (`==`, `/=`)
`Ord a`	lze porovnávat (`<`, `<=`, `>`, `>=`, `min`, `max`)
`Num a`	lze dělat aritmetické operace (`+`, `-`, `*`, `abs`, `negate`, `signum`)
`Integral a`	lze celočíselně dělit (`div`, `mod`, `quot`, `rem`, `divMod`, `quotRem`)
`Bounded a`	má nejmenší a největší prvek (`minBound`, `maxBound`)
`Show a`	lze převést na řetězec (`show`)
`Read a`	lze přečíst z řetězce (`read`) použití např. `((read "123") :: Int)`

Syntaktické konstrukce

`if b then x else y`	`x`, pokud `b` je `True`, jinak `y`
`let x = t in s`	`s` po nahrazení všech `x` za `t`
`f x =` *tělo* `where x = t`	lokální definice v těle funkce

Funkce

`f . g`	složení funkcí `f` a `g` (nejdřív `g`, pak `f`)
`\x y z -> t`	funkce s argumenty `x,y,z`, vrací `t`
`(+2)`	`\x -> x + 2`
`(2+)`	`\x -> 2 + x`
`` `f` ``	infixový zápis binární funkce `f`
`id`	funkce, která vrací nezměněný vstup
`const x`	funkce, která vždy vrací hodnotu `x`
`flip f`	prohození argumentů binární funkce `f`

Dvojice

`fst p`	první prvek z dvojice `p`
`snd p`	druhý prvek z dvojice `p`

Seznamy

`head xs`	první prvek `xs`
`tail xs`	`xs` bez prvního prvku
`last xs`	poslední prvek `xs`
`init xs`	`xs` bez posledního prvku
`xs !! k`	`k`-tý prvek `xs` (pozice počítá od 0)
`x : xs`	přidání prvku `x` na začátek seznamu `xs`
`xs ++ ys`	zřetězení seznamů `xs` a `ys`
`take k xs`	prvních `k` prvků z `xs`
`drop k xs`	`xs` bez prvních `k` prvků
`map f xs`	aplikuje `f` na všechny prvky `xs`
`filter p xs`	prvky `xs`, pro které `p` vrací `True`

```
zip [x,y,z] [a,b]         ≡  [(x,a),(y,b)]
zipWith f [x,y,z] [a,b]   ≡  [(f x a),(f y b)]
```

Agregační funkce na seznamech

```
foldl (+) 4 [1,2,3]  ≡  ((4 + 1) + 2) + 3)
foldr (+) 4 [1,2,3]  ≡  1 + (2 + (3 + 4))
foldl1 (+) [1,2,3]   ≡  (1 + 2) + 3
foldr1 (+) [1,2,3]   ≡  1 + (2 + 3)
```

Definice funkce podle vzoru

`f [] =` *tělo*	pro prázdný seznam
`f [x] =` *tělo*	pro jednoprvkový seznam
`f (x:xs) =` *tělo*	pro seznam délky alespoň 1
`f [x,y] =` *tělo*	pro dvouprvkový seznam
`f (x:y:xs) =` *tělo*	pro seznam délky alespoň 2
`f (x,y) =` *tělo*	pro dvojici
`f 0 =` *tělo*	pro nulu
`f 1 =` *tělo*	pro jedničku
`f 42 =` *tělo*	pro čtyřicet dva
`f x =` *tělo*	pro jakýkoliv vstup
`f _ =` *tělo*	pro jakýkoliv vstup (beze jména)

IO

`a1 >>= f`	spusť akci `a1`, její výsledek předej funkci `f`, jejíž výsledek spusť
`a1 >> a2`	spusť akci `a1` a poté akci `a2`
`putStr s`	akce; vypíše `s`
`putStrLn s`	akce; vypíše `s` a konec řádku
`getChar`	akce; přečte od uživatele znak, vrátí ho jako výsledek
`getLine`	akce; přečte od uživatele řádek, vrátí ho jako výsledek
`pure x`	akce s výsledkem `x`, nedělá nic