## Funkce

```
triple::Int->Int           mul::Int->Int->Int      jako mul::Int->(Int->Int)
triple x = 3 * x           mul x y = x * y         pak také triple = mul 3

len::[a]->Int              head::[a]->a
len []    = 0              head (a:_) = a
len (_:r) = 1+len r        {- head [] = error "Sakrišky sakrišky" -}

vse::[Bool]->Bool
vse [] = True
vse (b:bs) | b==True   = vse bs       nebo vse (True:r)  = vse r
           | otherwise = False             vse (False:_) = False
```

## Základní konstrukce

```
fact1 n = if n==1 then 1 else n * fact1 (n-1)
fact2 n = let fn' = fact2 (n-1)
          in if n==1 then 1 else n*fn'
fact3 n = let fn' = if n==1 then 1 else fact3 (n-1)
          in n*fn'
add a b = let (+)=(-)
              (-)=(+)
          in a-b
cycle x = let x' = x ++ x' in x'

fib n = fib' n 1 0
 where
  fib' 0 _   fn = fn
  fib' n fn' fn = fib' (n-1) fn (fn+fn')

hd x = case x of []  -> error "Prázdný seznam"
                 x:_ -> x
ones  = 1 : ones
prirozena n = n : prirozena (n+1)   nebo prirozena n = [n..]
mult5 = [5*i | i<-prirozena 1]      nebo mult5 = [5,10..]

fibs1 = 0:1:[a+b | a <- fibs1, b <- tail fibs1]
fibs2 = 0:1:[a+b | (a,b) <- zip fibs3 (tail fibs3)]
fibs3 = 0:1:[a+b | a <- fibs2 | b <- tail fibs2]     v GHC extensiona

primes = sieve [2..]
 where
  sieve (p:ns) = p : sieve [n | n <- ns, n `mod` p /= 0]
```

**where** nemůže být úplně všude. Může být jenom za definicí funkce nebo za libovolnou větví **case**.

```
a `plus` b = a+b        <=>  plus a b = a+b;
a `plus` b = (+) a b    <=>  plus a b = (+) a b;
inc1 = (1 +)
div3 = (/ 3)
```

## Datové typy

♣ Algebraické datové typy

```
data Tree a = Node a (Tree a) (Tree a) | Nil
data RoseTree a = RoseTree a [RoseTree a]
data Color a = RGB a a a | HSV a a a | Gray a
```

♣ Typová synonyma

```
type String = [Char]
```

♣ Newtype

```
newtype Parser a = Parser (String -> (String, a))
```

♣ Recordy

```
data S = S {znaku, slov, radek::Int, slovo::Bool}  Jako S Int Int Int Bool
newStav = S {slov=4, slovo=False, radek=2, znaku=1}
addWord s = let a=slov s in s {slov = a+1}          Nebo rovnou s {slov=1+slov s}
data S = A {slov::Int} | B Int | C {slov::Int}      Korektní, slov::S->Int
```

```
data S = A {slov::Int} | B {slov::Float}           Nejde!
data S = S {slov::Int}; slov::cokoliv              Nejde, slov nesmí být znovu deklarována
```
Rošíření GHC:
```
data C = C {a, b, c, d :: Int}
♣ NamedFieldPuns:
  f (C {a, b}) = a+b                       místo f (C {a=a, b=b}) = ...
♣ RecordWildCards:
  f (C {a=1,..}) = b+c+d                   místo C {a=1, b=b, c=c, d=d}
  let {a=1; b=2; c=3; d=4} in C {..}       místo C {a=a, b=b, c=c, d=d}
```

## Užitečné drobnosti

```
{-#LANGUAGE Rozsireni#-}          Zapne Rozsireni v tomto zdrojovém souboru
error::String->a
Debug.Trace.trace::String->a->a   Pro použití musíte import Debug.Trace
(.)::(b->c)->(a->b)->(a->c)       rev = a . b . c    místo rev x = a (b (c (x)))
($)::(a->b)->a->b                 rev x = a $ b $ c x místo rev x = a (b (c (x)))
infix priorita operátor           priority 0–9, infixl a infixr pro asociativní; default infixl 9

f .> g = g . f
x $> f = f $ x
infixl 0 $>
```

## Líné a striktní vyhodnocování

```
seq::a->b->b
```
Primitivum jazyka, při svém vyhodnocení vyhodnotí svůj první parametr, pak druhý a ten vrátí.
```
data Complex = !Float :+ !Float          Oba Floaty jsou striktní, tj. vždy vyhodnocené

...let !a=1`div`0 in...                  Rozšíření GHC; do a se ihned dosadí výsledek výpočtu
 ->let  a=1`div`0 in a`seq`...
```
Poznámka: všimněte si toho @:+ @. Je to datový kontruktor ve formě operátoru – ty musí začínat dvojtečkou.

## Typy polymorfních metod

```
apply f = (f 3, f True)
apply id

let f = id in (f 3, f True)
```

## Moduly

```
module Test (f) where

import B
import C (func)
import C hiding (func)
import qualified D
import qualified E as EEE

f x = func x
g x = ...
```

♣ Hierarchické moduly
```
module Data.Array
module Control.Monad
```

## Používání kompilátoru a interpreteru

- ♣ Kompilace: `ghc --make f.hs`, `-O2` optimalizace, `-fglasgow-exts` různá rozšíření. Musí obsahovat `main`.
- ♣ Interpreter: `ghci f.hs` nebo `hugs f.hs`, `hugs`u můžete dát `-98` místo `-fglasgow-exts`
  `:l f.hs` načte `f.hs`; `:r` reloadne načtené moduly; `:t expr` vypíše typ `expr`; ostatní provede daný příkaz
- ♣ Skripty: vytvořte soubor `.hs`, `chmod a+x`, první řádek `#!/usr/bin/runhaskell` či `runghc` či `runhugs`
- ♣ Literate: haskell zná i soubory `.lhs`: každá řádka je komentář, pokud nezačíná znakem > nebo není v bloku začínajícím `\begin{document}` a končícím `\end{document}`. Nemixujte to v jednom souboru. Navíc kolem bloku řádek začínajícím > musí být prázdná řádka.
- ♣ Haskell platform: distribuce GHC + množství (víceméně standardních) knihoven spravovaným komunitou.
- ♣ Hackage: na adrese `http://hackage.haskell.org/` je centrální repository balíků do Haskellu, po instalaci utility `cabal-install` stačí z příkazové řádky udělat `cabal install network-bytestring`.