```
                         Standardni typy v Haskellu
                         --------------------------
data Bool = True | False
     Int  = [-2^29 .. 2^29 -1]  omezený integer
     Integer = ( ..-1,0,1.. )   neomezený integer
     Ratio Int, Ratio Integer   zlomky s Intem nebo Integerem
     Float, Double
     Complex Float, Complex Double
data Char = znaky z unicode, momenálně čtyřbajtové; 'a' '\x158'
type String = [Char]
data () = () unit type

data Barva = R | G | B     data Bod = Bod Int Int Int
data BodOf a = Bod a a a   data BarvaOf a = RGB a a a | HSV a a a | Gray a | None

data [a] = []  |  a : [a]
data (a,b); data (a,b,c); data (a,b,c,d);

data Maybe a    = Nothing | Just a
data Either a b = Left a | Right b
data Ordering   = LT | EQ | GT

pi::Double
pi = 3.14159265358979
name::String
name = "Petronel"
                                  Funkce
                                  ------
triple::Int->Int
triple x = 3 * x
mul::Int->Int->Int         jako mul::Int->(Int->Int)
mul x y = x * y            pak také triple = mul 3
len::[a]->Int
len []    = 0
len (_:r) = 1+len r
head::[a]->a
head (a:_) = a             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 "Nene"
                 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)
```

Užitečné drobnosti

error::*String->a;* — ***Date.Trace.trace::String->a->a***

(.)::*(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 $>
```

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

Typové třídy

```
elem::a->[a]->Bool
elem _ []     = False
elem a (x:xs) = if a==x then True else elem a xs
```
Kde se vezme == ?

```
class Eq a where
  (==),(/=)::a->a->Bool
instance Eq Bool where
   True==True = True;  False==False = True;   _==_ = False
   True/=True = False; False/=False = False;  _/=_ = True

class Eq a where
  (==),(/=)::a->a->Bool
  (==) = not . (/=)
  (/=) = not . (==)
elem::(Eq a)=>a->[a]->Bool

class Eq a=>Ord a where ...
```
♣ Základní typové třídy: *Eq, Ord, Read, Show, Enum, Bounded*

♣ Číselné typové třídy: *Num, Integral, Fractional, Real, Floating, RealFrac, RealFloat*

♣ Další užitečné: *Bits, Random, Ix, IArray, MArray, Functor, Monad, MonadPlus, ...*

Parametrizovat se dá přes libovolný typ. A typ je například také `a->b` s typovým konstruktorem `(->)`.

```
data Tree a = E | V a [Tree a] deriving (Eq, Ord, Show, Read)
data Colour = Red | Green | Blue deriving (Eq, Ord, Show, Read, Enum, Bounded)
data Pair a = P a a deriving (Eq, Ord, Show, Read, Bounded)
```

Úkoly

♣ Napište skript, který přečte standardní vstup a na výstup vypíše jednu řádku:
`počet_řádek počet_slov počet_nemezerových_znaků`.
Je vhodné použít `main = interact w` pro **w :: *String -> String***.

♣ Napište skript, který se chová jako `cat -n`, tj. čte standardní vstup a každý řádek očísluje stylem `printf("%6d\t%s", číslo_řádku, text_řádku)`

Pro následující funkce si zadefinujte binární strom, který má hodnoty typu `a` v každém vrcholu.

♣ Napište funkci, která dostane setříděný seznam a vytvoří z něj perfektní binární strom. Perfektní znamená, že v každém vrcholu je velikost podstromu levého syna rovna velikosti podstromu pravého syna až na plus minus 1.

♣ Vytvořte funkci, které zadáte mocninu dvojky n=2^k a ona vytvoří úplný binární strom s k hladinami. V každém vrcholu je číslo 1. Optimalizujte na časovou složitost...

♣ Vytvořte funkci, které dáte binární strom s `Int`y, a ona vytvoří strom stejného tvaru, taktéž s `Int`y, který má v každém vrcholu uložený (celočíselný) průměr všech hodnot vrcholů původního stromu. Pozor, původní strom musíte projít pouze jednou a ten nový nemůžete procházet vůbec (můžete ho jenom vytvořit).