```
                                    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 [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 take 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]

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

Typové třídy

```
class Eqq a where
  (===), (/==) ::a->a->Bool
  --defaultni metody
  a /== b = not $ a===b

instance Eqq Int where
  a===b = a==b

find::(Eqq a)=>a->[a]->Bool
find x []     = False
find x (y:ys) | x===y = True
              | True  = find x ys

class (Eqq a) => Ordd a  where
  ...
```

Úkoly

1) *Napiste* skript, ktery dostava jako argumenty soubory, a pro kazdy soubor na jednu radku vypise (stejne jako wc bez parametru): "pocet_radek pocet_slov pocet_nemezerovych_znaku jmeno_souboru"

2) *Napiste* skript, ktery se chova jako cat -n, tj. cte standardni vstup a kazdy radek ocisluje stylem printf("%6d\t%s", cislo_radku, text_radku)

3) *Zadefinujte* si strom s hodnotami ve vsech vrcholech. *Napiste* funkci, ktera dostane setrideny seznam a vytvori z nej perfektni binarni strom, tj. strom, ze v kazdem vrcholu je velikost podstromu leveho syna rovna velikosti podstromu praveho syna az na +-1.

4) *Vytvorte* funkci, ktere zadate mocninu dvojky n=2^k a ona vytvori uplny binarni strom s k hladinami. *V* kazdem vrcholu je cislo 1. *Optimalizujte* na casovou slozitost (nejde to rychleji nez linearne?)

5) *Vytvorte* funkci, ktere date strom s *Int*-y, a ona vytvori jiny strom s *Int*-y, ktery ma v kazdem vrcholu ulozeny (celociselny) prumer vsech hodnot vrcholu.