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
```

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...
 ->let  a=1`div`0 in a`seq`...
```
Rozšíření GHC; do a se ihned dosadí výsledek výpočtu

Poznámka: všimněte si toho **:+** . Je to datový kontruktor ve formě operátoru − ty musí začínat dvojtečkou.

Standardní pole

```
array     ::Ix a=>(a,a)->[(a,b)]->Array a b   array (1,10) [(i,i)|i<-[1..10]]
listArray ::Ix a=>(a,a)->[b]->Array a b       listArray (1,10) [1..10]
(!)       ::Ix a=>Array a b->a->b             a!1
bounds    ::Ix a=>Array a b -> (a,a)  indices::(Ix a)=>Array a b->[a]
elems     ::Ix a=>Array a b -> [b]    assocs ::(Ix a)=>Array a b->[(a,b)]
(//)      ::Ix a=>Array a b->[(a,b)]->Array a b a//[(1,2), (3,4)], dělá celou kopii
```

Pole jsou líná v hodnotách − nevyhodnocují, dokud nemusí.
Vícerozměrná pole pomocí `array ((1,1),(100,100)) [((i,j),i+j)|i<-[1..100],j<-[1..100]]`

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`, hugsu 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.

Funkce pro práci se seznamy z Prelude

```
map :: (a->b) -> [a] -> [b]              map f xs = [ f x | x <- xs ]
(++) :: [a] -> [a] -> [a]
filter :: (a->Bool) -> [a] -> [a]        filter p xs = [ x | x <- xs, p x ]
concat :: [[a]] -> [a]
concatMap :: (a->[b]) -> [a] -> [b]      concatMap f = concat . map f

head, last :: [a] -> a
tail, init :: [a] -> [a]

foldl, foldl' :: (b->a->b) -> b -> [a] -> b
foldl1 :: (a->a->a) -> [a] -> a
scanl :: (b->a->b) -> b -> [a] -> b
scanl1 :: (a->a->a) -> [a] -> a
foldr :: (a->b->b) -> b -> [a] -> b
foldr1, scanr, scanr1

sum = foldl (+) 0
prod = foldl (*) 1

map f = foldr ((:) . f) []
filter p = foldr (\x xs -> if p x then x : xs else xs) []
```

```
reverse :: [a] -> [a]
reverse = foldl (flip (:)) []

all, any :: (a->Bool) -> [a] -> Bool
and, or :: [Bool] -> Bool
and = all (==True)
or  = any (==True)

repeat :: a -> [a]
replicate :: a -> Int -> [a]
iterate :: (a->a) -> a -> [a]
cycle :: [a] -> [a]
```

Domácí úkoly

♣ Napište příkaz funkci **pyth::*[(Int,Int,Int)]***, která bude vracet všechny pythagorejské trojice, tj. trojice kladných přirozených čísel, že a2+b2=c2. Tato funkce musí vracet všechny takové trojice a přitom každou právě jednou ((3,4,5) a (4,3,5) jsou stejné trojice).

♣ Napište příkaz `ls`, který dostane na příkazové řádce několik adresářů a každý adresář rekurzivně projde a vypíše.

♣ Napište příkaz `diff`, který dostane na příkazové řádce dva soubory (pokud jeden chybí, tak je to standardní vstup) a najde diff, tj. nejmenší počet řádek, které je třeba odstranit či přidat, aby se z prvního souboru stal druhý. Na výstup