1. Foldový najmenší a najväčší

```
:load Najmensi.hs

najmensich :: [Int] -> Int
najmensich xs = foldl (\c x -> if x == minim then c+1 else c) 0 xs
            where minim = foldr min 999999 xs

vacsiZaMensim :: [Int] -> Int
vacsiZaMensim (p:xs) = snd (foldl (\(a,c) x -> if x > a then (x,c+1)
                                               else (x,c))
                                               (p,0) xs)
```

Zaujímavá úloha na zamyslenie. V oboch prípadoch treba navyšovať nejaké počítadlo podľa špecifických podmienok. Vybral som si funckionál `foldl` (aj keď som pravičiar). Vo funkcii `najmensich` si vopred vypočítam minimum daného zoznamu tiež pomocou funkcionálu, tentokrát `foldr`. Podľa zadania by nemali hodnoty v zozname prekročiť `999999`, tak to využijem na porovnávanie a hľadanie minimálneho prvku. V hlavnom `foldl` potom už len priamočiaro navyšujem počítadlo, ak sa daný prvok zoznamu rovná nájdenému minimu.

V druhej časti si už nepomôžeme s jednou premennou, tak po vzore niekoľkých úloh v prednáškach/cvičeniach, si uložíme do dvojice prvý prvok zoznamu (ako predchodcu druhého prvku) a počítadlo. Funkcionál `foldl` potom prechádza zvyšok zoznamu a porovnáva prechádzaný prvok s predchodcom vo dvojici `(predchodca, counter)`. Ak je prechádzaný prvok väčší, zvýši počítadlo a v každom prípade vymení predchodcu za aktuálny prvok a posunie sa ďalej. Výsledný počet máme uložený vo dvojici na druhom mieste, tak si ho cez `snd` vypýtame.

```
*Najmensi> najmensich [-7,-2,3,2,2,1,1,-7,-7,1,-2,-7, 2, 2, 2, 2,-7]
5
*Najmensi> najmensich [3,2,1,2,3,4,5,1,1,2,3]
3
*Najmensi> najmensich [3,2,1,2,3,0,5,1,1]
1
*Najmensi> najmensich [1,1]
2
*Najmensi> vacsiZaMensim [3,3,1,2,3,4,5,1,1,2,3]
6
*Najmensi> vacsiZaMensim [0,0,0,0,0,0,1,0,0,0,0]
1
*Najmensi> vacsiZaMensim [1..100]
99
```

2. MID-HS-foldový

```
:load Priemery.hs
```

Začnime geometrickým priemerom a funkcionálom `foldr`, podobne ako na prednáške/cviku, medzivýsledky si ukladáme do dvojice `(product, n)`:

```
-- foldr vektor
geomR :: [Float] -> Float
geomR xs = p ** (1/n)
        where (p, n) = foldr (\x (p, n) -> (p * x, n + 1)) (1,0) xs

-- foldr matica
geomM :: [[Float]] -> Float
geomM xs = p ** (1 / fromIntegral n)
        where (p, n) = foldr (\x (p, n) -> (p * product x, n + length x)) (1,0) xs

-- taka sugar verzia s flattnutou maticou
geomM' :: [[Float]] -> Float
geomM' xs = geomR (concat xs)
```

Maticový príklad má asi mnoho riešení, zvolil som také pomerne `straightforward`, t.j. celkový `product` postupne násobime `product`-om daného riadku matice a do `n` pripočítavame dĺžky daných riadkov, teda počet prvkov (dúfam, že neboli zakázané použitia iných funkcií). Funkcia `length` asi spôsobila, že z `n` sa stal striktný `Int`, čím z nejakého dôvodu zlyhával podiel `(1/n)`, tak to trebalo fixnúť konvertnutím cez `fromIntegral`. Maticový geometrický priemer sa dá vypočítať aj za použitia vstavanej `concat` funkcie, ktorá prevedie maticu na vektor, čím umožní použiť prvú funkciu na výpočet geometrického priemeru `geomR` ale bez foldov :(.

```
*Priemery> geomR [2,3,6,7,7,8,9,9,9,10]
6.327492
*Priemery> geomR [5, 2, 7, 1, 4, 3, 2, 2, 2]
2.6623385
*Priemery> geomR [1..8]
3.7643507
*Priemery> geomM [[5, 2, 7], [1, 4, 3], [2, 2, 2]]
2.6623385
```

Poďme na harmonický priemer:

```
-- foldl vektor
harmoR :: [Float] -> Float
harmoR xs = uncurry (flip(/)) $ foldl (\(f,n) x -> (f+(1/x),n+1)) (0,0) xs

-- foldl matica
harmoM :: [[Float]] -> Float
harmoM xs = uncurry (flip(/)) $ foldl (\(fracs, n) x
    -> (fracs + sum (map (1/) x), n + fromIntegral (length x))) (0,0) xs

-- taka sugar verzia s flattnutou maticou
harmoM' :: [[Float]] -> Float
harmoM' xs = harmoR (concat xs)
```

Podobne ako pri geometrickom, len tentoraz cez `foldl`. Pri harmonickom priemere vektora si do dvojice `(fracs, count)` načítavajme prvky zoznamu v tvare `1/x`, kde `x` je prvok zoznamu a samozrejme počítadlo `count`, resp. `n`. Keďže na výslednej dvojici netreba robiť nejaké špecifické operácie, len delenie, využime funkciu `uncurry` a `flip`, vďaka ktorým dôjde k podielu týchto dvoch prvkov vo výslednej dvojici, avšak vo vymenenom poradí.

Podobne v matici, len miesto `1/x`, kde `x` je prvok zoznamu, do výsledného prvku `fracs` vo dvojici pripočítavame súčet všetkých prvkov `1/x` v danom riadku matice a podobne, počítadlo navyšujeme o dĺžku daného riadku. Ak si porovnáme výsledky, je vidno, že sme stratili nejaké desatinné miesta zaokrúhľovaním, zrejme pri explicitnom súčte v jednom riadku matice. Výsledok je ale principiálne správny.

```
*Priemery> harmoR [2,3,6,7,7,8]
4.2531643
*Priemery> harmoM [[2,3,6],[7,7,8]]
4.253165
*Priemery> harmoM' [[2,3,6],[7,7,8]]
4.2531643
```

3. Náhrdelníky

```
:load Nahrdelniky.hs
```

Pre začiatok si vygenerujme všetky kombinácie goraliek zo zadaných farieb nejakej dĺžky `n`:

```
vsetky :: Int -> [String] -> [[String]]
vsetky n xs | n == 0 = [[]]
            | otherwise = [color:others | color <- xs, others <- vsetky (n-1) xs]
```

```
*Nahrdelniky> vsetky 4 ["a", "b"]
[["a","a","a","a"],["a","a","a","b"],["a","a","b","a"],["a","a","b","b"], ...
```

Ďalej z týchto všetkých kombinácii budeme zrejme musieť vyfiltrovať také, ktoré sa tam nachádzajú viac krát v zmysle zadania. Ak som správne pochopil, tak ide o test, či sa v zozname nachádza nejaká iná kombinácia, ktorá vznikne nejakou rotáciou danej kombinácie. Preto si prichystajme funkciu `rotuj` ešte z DU6.

```
rotuj :: Int -> [a] -> [a]
rotuj 0 xs = xs
rotuj n xs = rotuj (n-1) (last xs : init xs)
```

Taktika pre samotnú kontrolu je taká, že pomocou funkcie `uzJe` zistíme, či sa teda nejaká rotácia danej kombinácie nachádza v nejakom zozname. Potom si rekurzívne pomocou funkcie `dajNerovnake` vytvoríme filter, kde pre jednotlivý prvok (kombináciu) v zozname kontrolujeme, či sa už nachádza vo zvyšku zoznamu.

```
uzJe :: [String] -> [[String]] -> Bool
uzJe xs ys = or [rotuj n xs `elem` ys | n <- [0..length xs - 1]]

dajNerovnake :: [[String]] -> [[String]]
dajNerovnake [] = []
dajNerovnake (x:xs) = [x | not (uzJe x xs)] ++ dajNerovnake xs
```

Tieto funkcie teda môžeme uplatniť na naše všetky kombinácie (`vsetky`) a dostaneme výledné kombinácie:

```
nahrdelniky :: Int -> [String] -> [[String]]
nahrdelniky n xs = dajNerovnake $ vsetky n xs
```

```
*Nahrdelniky> nahrdelniky 4 ["a", "b"]
[["a","a","a","a"],["b","a","a","a"],["b","a","b","a"],["b","b","a","a"],
["b","b","b","a"],["b","b","b","b"]]
*Nahrdelniky> length $ nahrdelniky 4 ["a", "b"]
6
```

4. HS3 - Druhý najväčší cez foldl/r

```
:load DruhyNajvacsi.hs
```

Zrejme by sme mali predpokladať, že vstupný zoznam má aspoň 2 prvky.

```
druhyNajvacsi :: [Int] -> Int
druhyNajvacsi (x:xs) = snd (foldl (\(naj1, naj2) p ->
                                    if p > naj1 then (p, naj1)
                                    else if p > naj2 && p < naj1 then (naj1, p)
                                    else (naj1, naj2)) (x, x) xs)
```

Idea programu je taká, že `foldl` bude kontrolovať a 'zbierať' prvý a druhý najväčší prvok prostredníctvom dvojice `(naj1, naj2)`. Začne s iniciálnou dvojicou `(prvy prvok, prvy prvok)`, kde `prvy prvok` je iniciálna najväčšia a zároveň aj druhá najväčšia hodnota (tu sa trebalo rozhodnúť, čo robiť so zoznamom tvaru `[a,a,a, ..., a]`, tak vrátim druhý najväčší ako prvý najväčší). Následne prechádza všetky zvyšné prvky zoznamu, zatiaľ čo kontroluje, či našiel väčšie maximum, ako doteraz (vtedy zadá nové maximum ako tento prvok a druhý najväčší prvok nahradí pôvodným najväčším). Zároveň kontroluje, či nenašiel nový druhý najväčší prvok (započíta ho miesto pôvodného).

`Pozn:` Funkcia `snd` vraj zapadá do množiny funkcií, ktoré môžeme 'pomenovať' (zdroj: vyučujúci).

```
*DruhyNajvacsi> druhyNajvacsi [2, 7, -4, 1, 5, 3]
5
*DruhyNajvacsi> druhyNajvacsi [6,2, 7, -4, 1, 5, 3, 10]
7
*DruhyNajvacsi> druhyNajvacsi [0..99]
98
*DruhyNajvacsi> druhyNajvacsi [-5, -10, -4, -2, -100, -19]
-4
*DruhyNajvacsi> druhyNajvacsi [5, 10]
5
```

5. HS 4 - foldový

```
:load ParnyNeparny.hs
```

Prvý pokus a idea:

```
parnyNeparny :: [a] -> ([a],[a], Int)
parnyNeparny xs = foldr (\x (z1, z2, c) ->
                    if c `mod` 2 == 0 then (x:z1, z2, c+1)
                    else (z1, x:z2, c+1)) ([], [], 0) xs
```

Logika tohto riešenia je pomerne jasná, využíva pri foldovaní vľavo trojicu `(zoz1, zoz2, counter)`, kde sa podľa párnosti hodnoty `counter` rozdeľujú prvky medzi prvý, resp. druhý zoznam. Jediná a závažná chybička krásy je, že takto definovaná funkcia nesedí s definíciou v zadaní, keďže vyprodukuje trojicu `([a],[a], Int)` namiesto dvojice `([a],[a])`.

Po chvíľke meditácie som si uvedomil, že miesto premennej počítadla mi stačí kontrolovať počet doteraz prerozdelených prvkov a dostanem tak totožnú hodnotu.

```
parnyNeparny :: [a] -> ([a],[a])
parnyNeparny xs = foldr (\x (z1, z2) ->
                    if (length z1 + length z2) `mod` 2 == 0 then (x:z1, z2)
                    else (z1, x:z2)) ([], []) xs
```

Ani toto ešte nebolo korektné riešenie, keďže pri párnom počte prvkov v zozname a `foldr` (ide od konca) nevie správne odhadnúť, či prvok, ktorým začína, je na párnom, alebo nepárnom indexe. Vyskúšajme teda prechádzať zoznam od začiatku (`foldl`) a prvky nerozdeľujme pridaním na začiatok, ale prilepme ich na koniec, hoci si efektívnosť dá pohov :(. Ale funguje. Poteší ma prípadný feedback, ako to zlepšiť.

```
parnyNeparny :: [a] -> ([a],[a])
parnyNeparny xs = foldl (\(z1, z2) x ->
                    if (length z1 + length z2) `mod` 2 == 0 then (z1++[x], z2)
                    else (z1, z2++[x])) ([], []) xs
```

```
*ParnyNeparny> parnyNeparny ["janko","marienka","ferko","masa","adamko"]
(["janko","ferko","adamko"],["marienka","masa"])
*ParnyNeparny> parnyNeparny [1..11]
([1,3,5,7,9,11],[2,4,6,8,10])
*ParnyNeparny> parnyNeparny [0..10]
([0,2,4,6,8,10],[1,3,5,7,9])
*ParnyNeparny> parnyNeparny [0,2..10]
([0,4,8],[2,6,10])
```