Laborator 7
Streamuri si list comprehension in Haskell

Documentatie:

Streamuri in Haskell

In continuare va prezint un exemplu de problema rezolvata cu ajutorul streamurilor. Deoarece in Scheme am lucrat deja cu streamuri, voi prezenta rezolvarea atat in Scheme cat si in Haskell (pentru a va usura intelegerea rezolvarii in Haskell)

Problema: Scrieti un program care sa reintoarca lista primelor 10 numere naturale, ordonate crescator.

```
; Scheme
(define succesor (lambda (n) (+ n 1)))
(define make_naturals
(lambda (k)
(cons k (lambda() (make_naturals (succesor k))))))
(define naturals_stream (make_naturals 0))
(define take1
(lambda (n stream)
(if (= n 0)
'()
(cons (car stream) (take1 (- n 1) ((cdr stream)))))))
(take1 10 naturals_stream)
```

```
-- Haskell
succesor k = k+1
make_naturals k = k:(make_naturals (succesor k))
naturals_stream = make_naturals 0
take1 n (x:stream) = if n == 0 then [] else x:(take1 (n-1) stream)
main = print (take1 10 naturals_stream)
```

Observatie: Haskell foloseste "lazy evaluation" (i.e. Haskell calculeaza ceva doar atunci cand are nevoie de rezultatul acelui calcul), in consecinta problema se rezolva mai simplu in Haskell decat in Scheme.

List comprehension in Haskell

- http://www.haskell.org/haskellwiki/List_comprehension
    - 1. Examples

Rezolvarea problemelor:

1. Scrieti un program care sa reintoarca lista numerelor cu cifre distincte <= 1000 folosind streamuri.

```
exista :: Integer -> [Integer] -> Integer
exista n [] = 0
exista n (x:ls) = if n == x then 1 + (exista n ls) else exista n ls

nr_list :: Integer -> [Integer] -> [Integer]
nr_list 0 l = l
nr_list n l = nr_list (div n 10) ((mod n 10) : l)

cif_dist_aux :: [Integer] -> Bool
cif_dist_aux [] = True
cif_dist_aux (x:xs) = (exista x xs == 0) && cif_dist_aux xs

cif_dist :: Integer -> Bool
cif_dist n = cif_dist_aux (nr_list n [])

succesor :: Integer -> Integer
succesor k = k+1

make_naturals :: Integer -> [Integer]
make_naturals k = k:(make_naturals (succesor k))

naturals_stream :: [Integer]
naturals_stream = make_naturals 0

take1 :: Integer -> [Integer] -> [Integer]
take1 n (x:stream) = if n == 0 then [] else if (cif_dist x) then x:(take1 (n-1) stream) else (take1 (n-1) stream)

main = print (take1 1000 naturals_stream)
```

2. Scrieti un program care sa reintoarca lista numerelor palindrom <= 1000 folosind streamuri.

```
nr_list :: Integer -> [Integer] -> [Integer]
nr_list 0 l = l
nr_list n l = nr_list (div n 10) ((mod n 10) : l)

inv :: [Integer] -> [Integer]
inv [] = []
inv (x:xs) = (inv xs) ++ [x]
```

```
succesor :: Integer -> Integer
succesor k = k+1

make_naturals :: Integer -> [Integer]
make_naturals k = k:(make_naturals (succesor k))

naturals_stream :: [Integer]
naturals_stream = make_naturals 0

take1 :: Integer -> [Integer] -> [Integer]
take1 n (x:stream) = if n == 0 then [] else if (nr_list x []) == (inv (nr_list x [])) then
x:(take1 (n-1) stream) else (take1 (n-1) stream)

main = print (take1 1000 naturals_stream)
```

3. Scrieti un program care sa reintoarca lista numerelor perfecte <= 1000 folosind streamuri.

```
divizori :: Integer -> Integer -> [Integer]
divizori n 0 = []
divizori n i = if (mod n i) == 0 then i:(divizori n (i-1)) else (divizori n (i-1))

sumlist :: [Integer] -> Integer
sumlist [x] = x
sumlist (x:xs) = x + (sumlist xs)

is_perfect :: Integer -> Bool
is_perfect n = sumlist (divizori n (div n 2)) == n

succesor :: Integer -> Integer
succesor k = k+1

make_naturals :: Integer -> [Integer]
make_naturals k = k:(make_naturals (succesor k))

naturals_stream :: [Integer]
naturals_stream = make_naturals 2

take1 :: Integer -> [Integer] -> [Integer]
take1 n (x:stream) = if n == 2 then [] else if (is_perfect x) then x:(take1 (n-1) stream) else
(take1 (n-1) stream)

main = print (take1 1000 naturals_stream)
```

4. Scrieti un program care sa reintoarca lista perechilor de numere prime gemene <= 100 folosind streamuri.

```
prim :: Integer -> Integer -> Bool
prim 0 _ = False
prim 1 _ = False
prim n 1 = True
prim n i = (mod n i) /= 0 && (prim n (i-1))

is_prim :: Integer -> Bool
is_prim n = prim n (div n 2)

succesor :: Integer -> Integer
succesor k = k+1

make_naturals :: Integer -> [Integer]
make_naturals k = k:(make_naturals (succesor k))

naturals_stream :: [Integer]
naturals_stream = make_naturals 2

take1 :: Integer -> [Integer] -> [(Integer,Integer)]
take1 n (x:stream) = if n == 2 then [] else if (is_prim x && is_prim (x+2)) then
(x,x+2):(take1 (n-1) stream) else (take1 (n-1) stream)

main = print (take1 100 naturals_stream)
```

5. Rezolvati problema 3 folosind list comprehension.

```
divizori :: Integer -> Integer -> [Integer]
divizori n 0 = []
divizori n i = if (mod n i) == 0 then i:(divizori n (i-1)) else (divizori n (i-1))

sumlist :: [Integer] -> Integer
sumlist [x] = x
sumlist (x:xs) = x + (sumlist xs)

is_perfect :: Integer -> Bool
is_perfect n = sumlist (divizori n (div n 2)) == n

list_perfect :: Integer -> [Integer]
list_perfect n = [ x | x <- [2..n], is_perfect x]

main = print (list_perfect 1000)
```

6. Rezolvati problema 4 folosind list comprehension.

```
prim :: Integer -> Integer -> Bool
prim 0 _ = False
prim 1 _ = False
prim n 1 = True
prim n i = (mod n i) /= 0 && (prim n (i-1))

is_prim :: Integer -> Bool
is_prim n = prim n (div n 2)

twinprim_list :: Integer -> [(Integer,Integer)]
twinprim_list n = [ (x,x+2) | x <- [2..n], (is_prim x) && (is_prim (x+2))]

main = print (twinprim_list 100)
```