Laborator 13

A. Cateva detalii suplimentare despre limbaj:

Tipuri de baza

- 3 :: Integer
- :: Double
- 'a' :: Char
- True :: Bool
- [1,2,3] :: [Integer] -- liste
- [1.3,2.8] :: [Double]
- [True,True,True,False] :: [Bool]
- "gica" :: String (un alias pentru [Char]

[True,3] – **eroare de tip** (listele au un singur tip)

Apel functie nume_funct lista_arg

sqrt 2
member 3 [1,2,3,4,5]
take 4 [2, 3, 5, 7, 11, 13, 17]

operatorii infix sunt si ei functii 3+4

[0,5 .. 100] – lista multiplilor lui 5 in interval [0,100]
[1..] – lista N^+
[1,3 ..] lista infinita a numerelor impare

: este un operator inix 'cons' (constructor liste) operator. De ex 3 : [4,5]

++ este operator infix pentru adaugare/reuniune [1,2] ++ [3,4,5]

strings = liste de (type Char). ['y', 'u', 'p']

Definirea unei Functii

```
double :: Integer -> Integer
double x = 2*x
```

Atentie limbajul este case senzitive

B. Testati urmatoarele probleme simple:

```
action :: IO String
action = do
  putStrLn "This is a line."
  input1 <- getLine
  input2 <- getLine
```

```
  -- The type of the `do` statement is that of its last line.
  -- `return` is not a keyword, but merely a function
  return (input1 ++ "\n" ++ input2) -- return :: String -> IO String
```

-- se poate folosi ca si `getLine`:

```
main = do
   putStrLn "I will echo two lines!"
   result <- action
   putStrLn result
   putStrLn "This was all, folks!"
```

duplicari

```
pack :: Eq a => [a] -> [[a]]
pack [] = []
pack (x:xs) = (x:first) : pack rest
       where
        getReps [] = ([], [])
        getReps (y:ys)
              | y == x = let (f,r) = getReps ys in (y:f, r)
              | otherwise = ([], (y:ys))
        (first,rest) = getReps xs
```

afisare rezultat numeric de evaluare a unei functii

functia

....

```
main = do

          let result = func param

          putStrLn "rezultatul este: "

          print result
```

Este un numar prim (folosind ciurul lui Eratostene)

```
import Data.List
isPrime :: Integer->Bool
isPrime k = k > 1 &&
   foldr (\p r -> p*p > k || k `rem` p /= 0 && r)
```

```
    True primesTME

{-# OPTIONS_GHC -O2 -fno-cse #-}
-- tree-merging Eratosthenes sieve
--  producing infinite list of all prime numbers
primesTME = 2 : gaps 3 (join [[p*p,p*p+2*p..] | p <- primes'])
  where
    primes' = 3 : gaps 5 (join [[p*p,p*p+2*p..] | p <- primes'])
    join  ((x:xs):t)       = x : union xs (join (pairs t))
    pairs ((x:xs):ys:t)    = (x : union xs ys) : pairs t
    gaps k xs@(x:t) | k==x  = gaps (k+2) t
                    | True  = k : gaps (k+2) xs

main = do
    print (isPrime 11)
    print (isPrime 12)
    print (isPrime 13)
```

8 regine

Cu functii separate pentru generarea listelor de candidati si testarea fiecarui candidat

```
queens :: Int -> [[Int]]
queens n = filter test (generate n)
    where generate 0     = [[]]
          generate k     = [q : qs | q <- [1..n], qs <- generate (k-1)]
          test []        = True
          test (q:qs)    = isSafe q qs && test qs
          isSafe   try qs = not (try `elem` qs || sameDiag try qs)
          sameDiag try qs = any (\(colDist,q) -> abs (try - q) == colDist) $ zip [1..] qs
```

doua regine nu pot ocupa aceeasi coloana:
try `elem` alreadySet

verifica reginele pe aceasi linie:

abs (try - q) == col
verifica reginele in aceasi diagonala.

Tema 1: Sa se implementeze calculul factorialului.
Tema 2: Sa se implementeze calculul factorial al lungimii listei.
Tema 3: Sa se sorteze un text folosind implementarea quicksort din curs.
Tema 4: Sa se inverseze o lista cu foldr.
Tema 5: Sa se scrie un parser care verifica daca un numar este numar.

Tema 6: Sa se citeasca un text si sa se puna cuvintele intr-un arbore binar. Apoi sa fie folosia pe post de dictionary.
Tema acasa: scrieti o aplicatie simpla pentru parser (vezi parser.hs si parsing.hs)

Bibliografie suplimentara

http://www.haskellforall.com/2013/12/equational-reasoning.html
http://en.wikibooks.org/wiki/Haskell/List_processing
http://www.vex.net/~trebla/haskell/lazy.xhtml
http://dannynavarro.net/2014/03/17/an-opinionated-importing-style-for-haskell/
http://pragprog.com/magazines/2012-12/web-programming-in-haskell
http://urchin.earth.li/~ian/style/haskell.html
http://sdg.csail.mit.edu/projects.html