Subiectul I

În acest subiect vom implementa funcționalități legate de jocul cinci-în-rând.

```
import           Data.List     (sort)

type Linie = Integer
type Coloana = Integer
```

O partidă este o listă de perechi linie-coloană reprezentând succesiunea de mutări ale celor doi jucători. Convenție: începe jucatorul cu "X".

Pentru a simplifica problema, vom considera câștigătoare doar secvențele de 5 piese unite pe aceeași linie. Piesele fiind reprezentate de pozițiile mutărilor corespunzătoare, două piese la pozițiile `(l1, c1)` și `(l2, c2)` sunt **unite** dacă `l1 = l2` și `c2 = c1 + 1`.

```
type Partida = [(Linie, Coloana)]

exemplu1 :: Partida
exemplu1 = [ (2, 2), (1, 3), (2, 3), (2, 4), (3, 5), (0, 2), (2, 1), (1, 4)
           , (2, 0), (1, 2), (3, 1), (1, 0)
           ]
```

1.1 (1,5p) Scrieți o funcție care dată fiind o partidă separă mutările celor 2 jucători. (Mutările primului jucător sunt în poziții pare)

Exemplu:

```
test11 :: Bool
test11 =
  separaX0 exemplu1
  ==  ( [(2,2),(2,3),(3,5),(2,1),(2,0),(3,1)]
      , [(1,3),(2,4),(0,2),(1,4),(1,2),(1,0)])

separaX0 :: [a] -> ([a], [a])
separaX0 [] = ([], [])
separaX0 (a:as) = (a:as2, as1)
  where (as1, as2) = separaX0 as
```

1.2. (1,5p) Dată fiind o listă de liste determinați lista cea mai lungă. Dacă sunt mai multe, întoarceți una din ele. Presupunem că lista nu e vidă.

Exemplu:

```
test12 :: Bool
test12 =
  maxLista [[1,2,3], [4,5], [6], [7, 8, 9, 10], [11, 12, 13]]
  == [7, 8, 9, 10]

maxLista :: [[a]] -> [a]
```

```haskell
maxLista l = l !! pozMax
  where
    lungimiPozitii = [(length x, px) | (x, px) <- l `zip` [0..]]
    (_, pozMax) = maximum lungimiPozitii
```

1.3. (2p) Scrieți o funcție care întoarce secvențele maximale de mutări unite (mutări cu aceeași linie și coloane consecutive). Presupuneți că lista de mutări este sortată lexicografic (mai întâi dupa linii, apoi după coloane).

```haskell
test130, test13X :: Bool
test130 =
  grupeazaUnite (sort [(1,3),(2,4),(0,2),(1,4),(1,2),(1,0)])
  == [[(0,2)],[(1,0)],[(1,2),(1,3),(1,4)],[(2,4)]]
test13X =
  grupeazaUnite (sort [(2,2),(2,3),(3,5),(2,1),(2,0),(3,1)])
  == [[(2,0),(2,1),(2,2),(2,3)],[(3,1)],[(3,5)]]

grupeazaUnite :: Partida -> [Partida]
grupeazaUnite =
  grupeazaDupa (\(l,c) (l',c') -> l == l' && c' == c+1 )

grupeazaDupa :: (a -> a -> Bool) -> [a] -> [[a]]
grupeazaDupa test = foldr go []
  where
    go a [] = [[a]]
    go a ((a':l):ls)
      | test a a' = (a:a':l):ls
      | otherwise = [a]:(a':l):ls
```

1.4. (0,5p) Dată fiind o partidă, scrieți o funcție care calculează cea mai lungă secvență de poziții unite pentru fiecare jucător.

```haskell
test14 :: Bool
test14 =
  maxInLinie exemplu1 == ([(2,0),(2,1),(2,2),(2,3)], [(1,2),(1,3),(1,4)])

maxInLinie :: Partida -> (Partida, Partida)
maxInLinie partida = (maxSecventa ics, maxSecventa zero)
  where
      (ics, zero) = separaX0 partida
      maxSecventa = maxLista . grupeazaUnite . sort
```

Subiectul II

Se consideră următoarea reprezentare pentru arbori binari.

```haskell
data Binar a = Gol | Nod (Binar a) a (Binar a)
```

De exemplu:

```
exemplu2 :: Binar (Int, Float)
exemplu2 =
  Nod
      (Nod (Nod Gol (2, 3.5) Gol) (4, 1.2) (Nod  Gol (5, 2.4) Gol))
      (7, 1.9)
      (Nod Gol (9, 0.0) Gol)
```

Un drum în acest arbore îl reprezentăm ca o secvență de direcții:

```
data Directie = Stanga | Dreapta

type Drum = [Directie]
```

2.1. (1p) Dat fiind un drum în arbore, determinați (dacă există) informația din nodul la care se ajunge parcurgând arborele după direcțiile date. Dacă se ajunge la un nod gol se va întoarce `Nothing`.

```
test211, test212 :: Bool
test211 = plimbare [Stanga, Dreapta] exemplu2 == Just (5, 2.4)
test212 = plimbare [Dreapta, Stanga] exemplu2 == Nothing

plimbare :: Drum -> Binar a -> Maybe a
plimbare _ Gol                      = Nothing
plimbare [] (Nod _ x _)             = Just x
plimbare (Stanga:is) (Nod st _ _)   = plimbare is st
plimbare (Dreapta:is) (Nod _  _ dr) = plimbare is dr
```

2.2. (1p) Presupunem că arborii conțin informație de tip `(Cheie, Valoare)` și sunt arbori de căutare după cheie (elementele din subarborele stâng au cheia mai mică decât cheia din rădăcină, iar cele din subarborele drept, au cheia mai mare decăt cea din rădăcină).

```
type Cheie = Int
type Valoare = Float
```

Definim monada `Writer` specializată la `String`:

```
newtype WriterString a = Writer { runWriter :: (a, String) }

instance Monad WriterString where
  return x = Writer (x, "")
  ma >>= k =  let (x, logx) = runWriter ma
                  (y, logy) = runWriter (k x)
              in  Writer (y, logx ++ logy)

tell :: String -> WriterString ()
tell s = Writer ((), s)
```

```haskell
instance Functor WriterString where
  fmap f mx = do { x <- mx ; return (f x) }

instance Applicative WriterString where
  pure = return
  mf <*> ma = do { f <- mf ; a <- ma ; return (f a) }
```

Scrieți o funcție care caută în arbore valoarea corespunzătoare unei chei date, întoarce această valoare (dacă există) și are ca efect lateral înregistrarea drumului parcurs.

```haskell
test221, test222 :: Bool
test221 = runWriter (cauta 5 exemplu2) == (Just 2.4, "Stanga; Dreapta; ")
test222 = runWriter (cauta 8 exemplu2) == (Nothing, "Dreapta; Stanga; ")

cauta :: Cheie -> Binar (Cheie, Valoare) -> WriterString (Maybe Valoare)
cauta cheie Gol = return Nothing
cauta cheie (Nod st (cheie', valoare) dr)
    | cheie == cheie' = return (Just valoare)
    | cheie < cheie' = do
        tell "Stanga; "
        cauta cheie st
    | otherwise = do
        tell "Dreapta; "
        cauta cheie dr
```