Előzetes tudnivalók

Használható segédanyagok:

- Haskell könyvtárak dokumentációja,
- Hoogle,
- a tárgy honlapja, és a
- Haskell szintaxis összefoglaló.

Más segédeszköz nem használható.

Ha bármilyen kérdés, észrevétel felmerül, azt a gyakorlatvezetőnek kell jelezni, **nem** a diáktársaknak!

A feladatsor megoldására 35 perc áll rendelkezésre (+ 5 perc feltöltésre)

A feladatok tetszőleges sorrendben megoldhatóak. A pontozás szabályai a következők:

- Minden teszten átmenő, a feladat kikötéseinek megfelelő megoldás érhet teljes pontszámot.
- Funkcionálisan hibás (valamelyik teszteseten megbukó) megoldás nem ér pontot.
- Fordítási hibás kód esetén a teljes zh 0 pontos.

Ha hiányos/hibás részek lennének a feltöltött megoldásban, azok kommentben szerepeljenek.

Tekintve, hogy a tesztesetek, bár odafigyelés mellett íródnak, nem fedik le minden esetben a függvény teljes működését, határozottan javasolt még külön próbálgatni a megoldásokat beadás előtt!

Az elméleti kérdésekre adott válaszokat a forráskódban kell elhelyezni, kommentben. Minden függvénynek meg kell adni a típusszignatúráját is. A függvények elvárt neve és típusa meg van adva. `Zarthelyi3Pot` néven kell deklarálni a modult. A `.hs` fájlt `.zip`-be tömörítve kell beadni.

Elméleti kérdés (1 pont)

Mit jelent a "lambda függvény" kifejezés?

Gyakorlati feladatok

FONTOS: a tesztesetek lefuttatásához importáljuk be a `Data.List` és `Data.Char` modulokat!

Furcsa feltétel (1 pont)

Adott egy predikátum és egy véges lista. Számoljuk meg, hogy hány elemre teljesül az, hogy a listában lévő rendezett párok első komponensére teljesül a predikátum, de a második komponensére már nem!

```
strangeCondition :: Num i => (a -> Bool) -> [(a, a)] {- véges -} -> i



strangeCondition (const True) [] == 0
strangeCondition (==1) [(1,2)] == 1
strangeCondition (=='a') [('b', 'a')] == 0
strangeCondition (>4) [(5, 6)] == 0
strangeCondition even [(2,2), (4,7), (16, 6)] == 1
```

```
strangeCondition (\x -> x `mod` 3 == 0) [(3, 2), (2,2), (4, 3)] == 1
strangeCondition (<10) [(3,11), (11,11), (8,1100), (2, 16)] == 3
```

Hatékony adatszerkezet (2 pont)

Definiáljunk egy olyan függvényt, amely paraméterül vár egy listát, és visszaadja, hogy mindegyik elemből hány található a listában (a sorrend nem számít). Feltételezhetjük, hogy a lista véges. Segítség: érdemes a sort függvényt használni (nem kötelező).

```
toMultiMap :: (Num i, Ord a) => [a] {- véges -} -> [(a, i)]
```

```
toMultiMap [] == []
sort (toMultiMap [1,5,0,8,4,3,5,5,0,2,3,4,6]) == sort [(0,2),(1,1),(2,1),(3,2),(4,2),(5,3),(6,1),(8,1)]
sort (toMultiMap ("alma a fa alatt, nyari piros alma")) == sort [(' ',6),(',',1),('a',9),('f',1),('i',2),
sort (toMultiMap $ map (:[]) "Kek a kokeny, rececе, ha megerik fekete") == sort [(" ",6),(",",2),("K",1),
```

Feltételes transzformáció (2 pont)

Definiáljunk egy olyan függvényt, amely paraméterül vár egy függvényt, egy predikátumot és egy listát és alkalmazza a függvényt azokra a listaelemekre, amelyekre teljesül a predikátum! Azokat az elemeket, amelyekre nem teljesül, hagyja az eredménylistában változatlanul.

```
conditionalMap :: (a -> a) -> (a -> Bool) -> [a] -> [a]
```

```
conditionalMap (+5) odd [] == []
conditionalMap (*2) even [1..10] == [1,4,3,8,5,12,7,16,9,20]
conditionalMap id (\x -> x > 3) [1..100] == [1..100]
take 100 (conditionalMap (+2) (\e -> e `mod` 4  == 0) (1:[4,8..])) == take 100 (1:[6, 10 ..])
```