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 20 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. `Zarthelyi2` néven kell deklarálni a modult. A `.hs` fájlt `.zip`-be tömörítve kell beadni.

Elméleti kérdések (1 pont / kérdés)

1. Mi a különbség a `div` és a `(/)` függvények között? (1 pont)

2. Mit jelent a heterogén adatszerkezet? (1 pont)

Gyakorlati feladatok

Párok listájává (1 pont)

Definiáljuk a `toTupleList` függvényt, amely a paraméterül kapott listából előállít az egy ugyanolyan hosszúságú listát, melynek minden eleme a kapott lista elemeiből álló rendezett pár.

```
toTupleList :: [a] -> [(a,a)]
```

```
toTupleList [1..5] == [(1,1),(2,2),(3,3),(4,4),(5,5)]
take 15 (toTupleList [1..]) == [(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10),(11,11),(12
take 15 (toTupleList [1564]) == [(1564,1564)]
```

A végtelenbe és tovább (1 pont)

Definiáljuk a `toInfinityAndBeyond` függvényt, amely a paraméterül kapott listát végtelenségig ismétli úgy, hogy először az eredeti listát adja vissza, majd a fordítottját, majd ismét az eredetit és így tovább. A megoldásban rekurziót! Ne használj listagenerátort!

```
toInfinityAndBeyond :: [a] -> [a]
```

```
take 25 (toInfinityAndBeyond [1..3]) == [1,2,3,3,2,1,1,2,3,3,2,1,1,2,3,3,2,1,1,2,3,3,2,1,1]
take 10 (toInfinityAndBeyond [1..5]) == [1,2,3,4,5,5,4,3,2,1]
take 10 (toInfinityAndBeyond ['a'..'z']) == "abcdefghij"
take 10 (toInfinityAndBeyond [1..]) == [1,2,3,4,5,6,7,8,9,10]
take 11 (toInfinityAndBeyond ['a'..'e']) == "abcdeedcbaa"
```

Pontosan kétszer (2 pont)

Definiáljuk az `exactlyTwoTimes` függvényt, amely kap egy lehetséges listaelemet és egy listát és igazzal tér vissza, ha a listaelem pontosan kétszer szerepel a listában! A megoldásban ne használd a `take` függvényt és annak általános változatát!

```
exactlyTwoTimes :: Eq a => a -> [a] -> Bool
```

```
not (exactlyTwoTimes 5 [1..10])
exactlyTwoTimes 5 [1,2,5,4,5]
exactlyTwoTimes 5 [1,2,5,5,8]
not (exactlyTwoTimes 5 [5,5,1,5])
exactlyTwoTimes 5 [5,5]
not (exactlyTwoTimes 'a' (cycle "abc"))
```