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. `Zarthelyi1Pot` néven kell deklarálni a modult. A `.hs` fájlt `.zip`-be tömörítve kell beadni.

Elméleti kérdések (2 pont)

Figyelem! Az elméleti kérdésekre csak indoklást tartalmazó válaszra adható pont.

1. Adott az alábbi függvény:

   ```
   f :: Int -> Int -> Double
   f x y = (x + y) / 5
   ```

   Ha helyes a függvény, akkor mi lesz az `(f 5 6)` függvényhívás eredménye? Ha nem helyes, akkor miért nem és hogyan lehet kijavítani úgy, hogy a "lényegi" működés ne változzon?

2. Add meg, hogyan zárójelezné a Haskell a következő kifejezést a tanult szabályok alapján. Értelmezhető (szintaxishelyes, típushelyes) kifejezés lesz-e az eredmény?

```
mod 1 (+) 2 3
```

Gyakorlati feladatok

Logikai értékek (1 pont)

Definiáljuk a `twoTrue` függvényt, amely akkor ad vissza `True` értéket, ha pontosan két paramétere `True` értékű! *Használjunk mintaillesztést!* **Ne használj elágazást, rekurziót, magasabb rendű függvényt!**

```
twoTrue :: Bool -> Bool -> Bool -> Bool
```

```
twoTrue False False False == False
twoTrue True False False ==  False
twoTrue False True True == True
twoTrue True False True == True
```

Vektorok összege (1 pont)

Adjunk meg függvényt, amely összead két darab 3-dimenziós vektort. Az összegvektor szintén egy 3-dimenziós vektor, melynek komponenseit úgy kapjuk, hogy a két vektor megfelelő komponenseit összegezzük. **Ne használj elágazást, rekurziót, magasabb rendű függvényt!**

```
sumVecs :: (Int, Int, Int) -> (Int, Int, Int) -> (Int, Int, Int)
```

```
sumVecs (1,1,1) (2,2,2)       == (3,3,3)
sumVecs (1,2,3) (10,20,30)    == (11,22,33)
sumVecs (-1,-2,-3) (10,20,30) == (9,18,27)
sumVecs (0,0,0) (-1,-2,-3)    == (-1,-2,-3)
```

Kártyapakli (2 pont)

Egy n franciakártyából álló kártyapaklit 4 darab kupacba szeretnénk rendezni úgy, hogy az egyes kupacok között körben haladva mindig a soron következő kupacra helyezünk egy kártyát.

Az első kupac sorszáma legyen 0. Feltehetjük, hogy a pakliban van legalább 1 kártya.

Adjunk függvényt, amely a kártyapakliban levő kártyák számát várja argumentumként, majd egy pár első tagjaként megadja melyik kupacba került a legutolsó kártya, a pár második tagjaként pedig megadja, hány kártya került az utolsó kupacba (hányszor értünk körbe a kupacokon).

Ne használj elágazást, rekurziót, magasabb rendű függvényt!

```
cards :: Int -> (Int, Int)
```

```
cards  1 == (0,0)
cards  2 == (1,0)
cards  3 == (2,0)
cards  4 == (3,1)
cards  5 == (0,1)
```

```
cards  6 == (1,1)
cards 11 == (2,2)
cards 12 == (3,3)
cards 13 == (0,3)
```