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 15 perc áll rendelkezésre (+ 5 perc feltöltésre)

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

- Minden teszten átmenő, a feladat kikötéseinek megfelelő megoldás ér 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. `Zarthelyi1` néven kell deklarálni a modult. A `.hs` fájlt `.zip`-be tömörítve kell beadni.

Elméleti kérdések (1 pont / kérdés)

- Mit jelent az, hogy "parciális függvény"?
- Tekintsük az alábbi függvényt. Helyes-e az alábbi kódrészlet? Ha igen, mi lesz az eredmény `f 5 2` kiértékelése esetén? Ha nem, hogyan lehetne kijavítani, hogy a "lényegi" működés ne változzon?

```
f :: Int -> (Int,Int) -> Int
f a b = (a + b)
```

Gyakorlati feladatok

És lőn világosság (1 pont)

Adott egy rendezett hármas, amely egy színt reprezentál `RGB (piros, zöld, kék)` formátumban. Növeljük meg a szín világosságát a paraméterül kapott pozitív számmal! Egy szín világosságát úgy növeljük meg, hogy minden komponenshez hozzáadjuk azt az értéket, amivel növelni szeretnénk a világosságot. Feltehetjük, hogy az értékek nem fognak túlcsordulni.

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

```
brighten (0, 0, 0) 10 == (10, 10, 10)
brighten (15, 100, 70) 30 == (45, 130, 100)
brighten (88, 92, 60) 14 == (102, 106, 74)
```

Ésből következik (1 pont)

Adjuk meg azt a függvényt, amely összeésel két paramétert, majd az eredményre és a harmadik paraméterére alkalmazza az implikációt. Használjunk mintaillesztést! A megoldás legyen a harmadik paraméterében lusta. (Tehát a tesztekben a harmadik paraméteren előfordulhatnak Exception-t okozó kifejezések is.) Az alábbi igazságtábla segítségünkre jöhet.

A	B	C	ANDIMPL(A,B,C)
Igaz	Igaz	Igaz	Igaz
Igaz	Igaz	Hamis	Hamis
Igaz	Hamis	Igaz	Igaz
Igaz	Hamis	Hamis	Igaz
Hamis	Igaz	Igaz	Igaz
Hamis	Igaz	Hamis	Igaz
Hamis	Hamis	Igaz	Igaz
Hamis	Hamis	Hamis	Igaz

```
andImpl :: Bool -> Bool -> Bool
```

```
andImpl True True True
not (andImpl True True False)
andImpl True False True
andImpl (1 == 0) True (10 `div` 0 == 10)
```

Szelektív inverzió (2 pont)

Adott egy egész számokból álló lista. Tartsuk meg a 7-tel osztható páratlan számokat, majd adjuk meg ezen számok ellentetjeit!

```
selectiveInversion :: [Integer] -> [Integer]
```

```
selectiveInversion [91, -21, 5, -49, 14, 7] == [-91,21,49,-7]
selectiveInversion [1, -78, 16, 14, 48] == []
selectiveInversion [] == []
take 15 (selectiveInversion [-10000..]) == [9989,9975,9961,9947,9933,9919,9905,9891,9877,9863,9849,9835,9821,9807,9793]
```