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 25 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. `Zarthelyi3` néven kell deklarálni a modult. A `.hs` fájlt `.zip`-be tömörítve kell beadni.

Elméleti kérdés (1 pont)

- Mit jelent a parciális függvényalkalmazás?

Gyakorlati feladatok

for ciklus szimulálása (2 pont)

Definiáld a `for` függvényt, amivel az imperatív nyelvekből ismert `for` ciklust szeretnénk Haskellben szimulálni. A függvény paraméterei a következők (ebben a sorrendben):

1. A ciklusváltozó kezdeti értéke.
2. Egy függvény, amellyel ellenőrizzük, hogy kell-e tovább futtatni a ciklust.
3. Egy függvény, amellyel léptethetjük a ciklusváltozó értékét.
4. A ciklus magjában módosítandó érték.
5. A ciklusmag, tehát a művelet, amelyet a ciklus előtt létrehozott értéken hajtunk végre.

A feladat megértését segítő példa (remélhetőleg):

```
string alma = "" // <- 4.
//   1. ↴     2. ↴   3. ↴
for(int i = 0; i < 10; i++)
```

```
{
    alma = alma + "alma" // <- 5.
}
```

Ezeket felhasználva addig léptessük a ciklusváltozót, amíg igaz rá a feltétel. Emellett minden egyes lépésben a ciklus magját is frissíteni kell.

```
for :: a -> (a -> Bool) -> (a -> a) -> b -> (b -> b) -> b
```

```
for 0 (<10) (+1) 0 (+1) == 10
for 1 (>10) (*100) 1 (+1) == 1
for 0 (<5) (+2) True not == False
for True id not [] (1:) == [1]
for 10 (>0) (+(-1)) 1 (*2) == 1024
snd (for 1 (<=10) (+1) (0,[]) (\(n,xs) -> (n+1,xs ++ [n]))) == [0..9]
length (take 1000 (for 0 (const True) id [] (0:))) == 1000
```

Listafelbontás (3 pont)

Bonts fel egy listát két részre az összes lehetséges módon úgy, hogy az eredmény egyes elemei olyan rendezett párok legyenek, amelyeket ugyanazon sorrendben összefűzve az eredeti listát kapjuk vissza.
(Azaz: `((x :: [a]) -> [(y :: [a], z :: [a])] -> x == y ++ z)`)

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

```
splitList [1,2,3] == [([],[1,2,3]),([1],[2,3]),([1,2],[3]),([1,2,3],[])]
splitList "alma" == [("","alma"),("a","lma"),("al","ma"),("alm","a"),("alma","")]
splitList "F" == [("","F"),("F","")]
take 20 (snd (splitList [10..] !! 5)) == [15..34]
fst (splitList [10..] !! 5) == [10..14]
take 20 (snd (splitList [10..] !! 100)) == [110..129]
```