Előzetes tudnivalók

Használható segédanyagok:

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

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

FONTOS:

- A megoldásban legalább az egyik (tetszőleges) függvényt **rekurzívan** kell megadni. Azaz a vizsga csak akkor érvényes, ha az egyik feladatot rekurzív függvénnyel adtátok meg és az helyes megoldása a feladatnak. A megoldást akkor is elfogadjuk, ha annak egy segédfüggvénye definiált rekurzívan. A könyvtári függvények (length, sum, stb.) rekurzív definíciója nem fogadható el rekurzív megoldásként.

- A programozási részből **legalább 7** pontot kell szerezni az évényes vizsgához!

- A feladatokat a kiírásnak megfelelően, az ott megadott típusszignatúrának megfelelően kell megoldani. A típusszignatúra nem változtatható meg. Megváltoztatott típusszignatúra esetén a feladat 0 pontot ér.

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

- Minden teszten átmenő megoldás érhet teljes pontszámot.
- Funkcionálisan hibás (valamelyik teszteseten megbukó) megoldás nem ér pontot.
- Fordítási hibás megoldás esetén a **teljes vizsga** 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*** *vagy megkérdezni a felügyelőket!*

A *Visual Studio Code Haskell Syntax Highlighting* bővítmény a csatolt fájlok között megtalálható.

Telepítés:

1. Bővítmények megnyitása bal oldalt (4 kicsi négyzet) (`Ctrl + Shift + X`)
2. `...` a megnyíló ablak jobb felső sarkában
3. `Install from VSIX...`, majd a letöltött állomány kitallózása

Feladatok

Üres listák elhagyása (2 pont)

Adott egy listákat tartalmazó lista. Adjuk meg azt a függvényt, amelyik a listából elhagyja az üres listákat és csak a legalább egyeleműeket tartja meg!

Segítség: Vegyük észre, hogy a típus **nem engedi**, hogy az `(==)` műveletet használjuk!

```
removeEmpties :: [[a]] -> [[a]]
removeEmpties [] == []
removeEmpties [[1,2], [], [], []] == [[1,2]]
removeEmpties [[1,2], [], [], [], [1,5,2,1]] == [[1,2],[1,5,2,1]]
removeEmpties [[]] == []
map (take 10) (removeEmpties [[1..]]) == [[1..10]]
removeEmpties [[],[]] == []
removeEmpties [[1],[2]] == [[1],[2]]
removeEmpties [[],[2]] == [[2]]
removeEmpties [[1],[]] == [[1]]
map (take 5) (removeEmpties [[1..],[10,9..]]) == [[1..5],[10,9..6]]
map (take 6) (removeEmpties [[1..],[]]) == [[1..6]]
removeEmpties [[],[],[],[],[]] == []
removeEmpties [[],[2,3,4],[5],[]] == [[2,3,4],[5]]
map (take 10) (removeEmpties ["alma","szilva","",repeat 'a',"",""]) ==
["alma","szilva","aaaaaaaaaa"]
map (take 5) (take 7 (removeEmpties (cycle [[True,True],[],[],[False,False,True],repeat
False,[]]))) ==
[[True,True],[False,False,True],[False,False,False,False,False],[True,True],[False,False,Tr
ue],[False,False,False,False,False],[True,True]]
```

Elemek eldobása (2 pont)

Definiáld a `dropUntil` függvényt, amely elhagyja egy lista elemeit addig, amíg a fennmaradó listára nem teljesül egy megadott feltétel. Végtelen lista esetén a függvénynek csak abban az esetben kell terminálnia, ha a feltétel teljesül valamelyik fennmaradó listarészre. Ha a feltétel nem teljesül egyik állapotban sem, akkor az eredmény legyen üres lista.

```
dropUntil :: ([a] -> Bool) -> [a] -> [a]
dropUntil (all even) [] == []
dropUntil null "barack" == []
dropUntil (\x -> length x == 6) "alma" == []
dropUntil (\x -> length x == 6) "barack" == "barack"
dropUntil (\x -> length x == 6) "sok leveles almafa" == "almafa"
dropUntil (all odd) [1,2,3,6,7,9,11,13,15] == [7,9,11,13,15]
take 20 (dropUntil (not . null) [1..]) == [1..20]
take 10 (dropUntil ([33,34,35] `isPrefixOf`) [1..]) == [33..42]
take 8 (dropUntil (\x -> head x == 10) [2,4..]) == [10,12,14,16,18,20,22,24]
dropUntil (\x -> x == reverse x) [1,2,3,1,2,1,2,0] == [0]
dropUntil (\x -> head x == 2) [1,3,5,7,9,11,13] == []
(\x -> case x of [f,g] -> (f 5, g 5); _ -> error "Baj van!") (dropUntil (\x -> foldr (.) id
x 1 == 21) [(+1),(+2),(*3),(+6)]) == (15, 11)
```

Számjegyek összege (2 pont)

Definiáljuk a `sumOfDigits` függvényt, amely megadja egy egész szám számjegyeinek összegét! Negatív szám esetén az előjelet hagyjuk figyelmen kívül.

Segítség: A feladat megoldásában a `div` és `mod` függvények segítenek. Ne feledjük, hogy 10-es számrendszerben dolgozunk.
Például:
`324 = 3 * 10² + 2 * 10¹ + 4 * 10⁰ = 3 * 100 + 2 * 10 + 4`

```
sumOfDigits :: Integral a => a -> a
sumOfDigits 324 == 9
sumOfDigits 0 == 0
sumOfDigits (-1) == 1
sumOfDigits 42 == 6
sumOfDigits (12034 :: Int) == 10
sumOfDigits (8723 :: Integer) == 20
sumOfDigits (-582) == 15
sumOfDigits 9 == 9
sumOfDigits 823746291 == 42
sumOfDigits (2741 :: Int) == 14
sumOfDigits (800001 :: Integer) == 9
sumOfDigits (-738) == 18
```

Adott mentén benne van (2 pont)

Definiáld a `hasBy` függvényt, amely ellenőrzi, hogy ha egy listányi értékre alkalmaz egy műveletet a függvény, akkor az úgy kapott lista valamely eleme benne van-e a harmadik paraméterként kapott listában!

```
hasBy :: Eq b => (a -> b) -> [a] -> [b] -> Bool
not (hasBy (> 2) [] [])
not (hasBy (`div` 0) [] [9,0,1,2,9,8,7,6,13,15,-1])
not (hasBy (`div` 0) [] [1..])
not (hasBy (+3) [1,2,3,4] [])
not (hasBy (< 2) [1,2,3] [])
not (hasBy (*2) [1] [1,3,5,7,9,11])
hasBy (*2) [1] [1,3,5,7,9,2,11]
hasBy id [1,2,3] [6,5,4,1,7]
hasBy (`div` 2) [1,4..100] [9,8,7,6,5,4,-1,10,9,25,15]
hasBy (`div` 2) [1,4..100] [9,8..]
hasBy (`div` 3) [1,6..] [6,5,10,11,15,7,9,123,-6,-3,345,76]
hasBy (`div` 3) [1,6..] [9,8..]
```

Morse-kódolás (2 pont)

Valósítsuk meg az `encodeMorse` függvényt a `morse :: [(Char,String)]` konstans lista segítségével (tehát a megoldásban fel kell használni ezt a listát), amely egy rendezett párokat tartalmazó listában tárolja az angol ábécé kisbetűit és az azokat reprezentáló morze kódokat. A függvény alakítson át minden karaktert a morze megfelelőjére. Amennyiben nem létezik ilyen, helyettesítsük a karaktert kérdőjellel (`'?'`). A karakterek között legyen pontosan egy darab szóköz, hogy egyértelmű legyen a karakter. Mindent az angol ábécé kisbetűin és a szóköz karakteren kívül tekintsünk ismeretlen karakternek.
A következő listára szükség van a megoldáshoz, így ezt másold be a megoldásodba.

```
morse :: [(Char,String)]
morse = [(' ',"/"),('a',".-"),('b',"-..."),('c',"-.-."),('d',"-.."),('e',"."),('f',"..-
."),('g',"--."),('h',"...."),('i',".."),('j',".---"),('k',"-.-"),('l',".-.."),('m',"--
"),('n',"-."),('o',"---"),('p',".--."),('q',"--.-"),('r',".-."),('s',"..."),('t',"-
"),('u',"..-"),('v',"...-"),('w',".--"),('x',"-..-"),('y',"-.--"),('z',"--..")]
```

Segítség: Használhatjuk az `unwords` függvényt, illetve az elemek kikereséséhez definiáljunk segédfüggvényt, vagy használhatjuk a `Data.List` modul `lookup` függvényét is.
A `lookup` használatára néhány példa:

```
lookup 1 [(2,'a'),(1,'d'),(3,'a')] == Just 'd'
lookup 4 [(2,'a'),(1,'d'),(3,'a')] == Nothing
lookup 1 [(2,'a'),(1,'c'),(3,'a'),(1,'d')] == Just 'c'
lookup True [(True,'a'),(False,'c'),(False,'b'),(True,'d')] == Just 'a'
encodeMorse :: String -> String
encodeMorse "" == ""
encodeMorse "aaa" == ".- .- .-"
take 6 (encodeMorse "aaaaaaaaaaa") == ".- .- "
encodeMorse "ABC" == "? ? ?"
encodeMorse "abcdefghijklmnopqrstuvwxyz" == ".- -... -.-. -.. . ..-. --. .... .. .--- -.-
.-.. -- -. --- .--. --.- .-. ... - ..- ...- .-- -..- -.-- --.."
encodeMorse "alma" == ".- .-.. -- .-"
encodeMorse "űéáúő" == "? ? ? ? ?"
encodeMorse "Alma" == "? .-.. -- .-"
encodeMorse "az alma leesett a fáról" == ".- --.. / .- .-.. -- .- / .-.. . . ... . - - / .-
/ ..-. ? .-. ? .-.."
take 10 (encodeMorse $ repeat 'a') == ".- .- .- ."
encodeMorse "\"" == "?"
encodeMorse " " == "/"
```

Feltételes szorzás (2 pont)

Definiáld azt a függvényt, amely egy tulajdonságok listáját és egy rendezett párokat tartalmazó listát kap paraméterül. A függvény szorozza össze a rendezett párok második komponensében található mindazon számokat, amelyek első felére az **összes** megadott tulajdonság teljesül! Feltehető, hogy a rendezett párok listája véges!

```
productIfs :: Num b => [a -> Bool] -> [(a, b)] -> b
productIfs [] [] == 1
productIfs [id] [] == 1
productIfs [odd, (\x -> x `mod` 5 == 0)] [] == 1
productIfs (repeat odd) [] == 1
productIfs [] [('a',1),('b',2)] == 2
productIfs [] [('c',2),('d',4)] == 8
productIfs [] [('c',2),('d',4),('e',7)] == 56
productIfs [(\x -> x == 'a' || x == 'b')] [('c',10),('b',5),('a',4),('a',3)] == 60
productIfs [(\x -> x == 'a' || x == 'b'), (/= 'b')] [('c',10),('b',5),('a',4),('a',3)] ==
12
productIfs [(\x -> x == 'a' || x == 'b'), (/= 'b'), (== 'c')]
[('c',10),('b',5),('a',4),('a',3)] == 1
productIfs [even] [(8,0),(9,2),(10,4),(11,3),(12,5),(14,6),(13,2),(14,4)] == 0
productIfs [(> 10)] [(8,0),(9,2),(10,4),(11,3),(12,5),(14,6),(13,2),(14,4)] == 720
productIfs [even,(> 10)] [(8,0),(9,2),(10,4),(11,3),(12,5),(14,6),(13,2),(14,4)] == 120
productIfs [(< n) | n <- [15,14..]] [(8,0),(9,2),(10,4),(11,3),(12,5),(14,6),(13,2),(14,4)]
== 1
```

Első 5-tel osztható (3 pont)

Definiáld a `first5` függvényt, amelynek a bemenete egy egész számokat tartalmazó lista. Az eredmény a bemeneti listának olyan **legrövidebb nem üres prefixe** legyen, amelynek az összege 5-tel osztható. Ha nincs ilyen prefix, az eredmény legyen `Nothing`!

```
first5 :: Integral a => [a] -> Maybe [a]
first5 [] == Nothing
```

```
first5 [2] == Nothing
first5 [5] == Just [5]
first5 [-1,2] == Nothing
first5 [1,-6] == Just [1,-6]
first5 [0,5] == Just [0]
first5 [-2,-2] == Nothing
first5 [-4,-5,2,2,10,-3] == Just [-4,-5,2,2]
first5 [2,3,4,6] == Just [2,3]
first5 [2,5,7,9,10,11,13,15] == Nothing
first5 [2,5,7,8,9,11,13,15,17] == Just [2,5,7,8,9,11,13]
first5 [10,11,2,2,3] == Just [10]
first5 [3,3,2,2] == Just [3,3,2,2]
first5 [16..] == Just [16,17,18,19]
```

Térképészet (3 pont)

Definiáld a `Place` paraméteres adatszerkezetet, amelynek legyen egy típusparamétere. Az adatkonstruktorai a következők:

- `Lake :: String -> a -> Place a`, az első paraméter a tó neve, a második egy hozzátartozó azt jellemző érték.
- `Mountain :: String -> a -> Place a`, az első paraméter a hegy neve, a második egy hozzátartozó azt jellemző érték.
- `Cave :: String -> a -> Place a`, az első paraméter a barlang neve, a második egy hozzátartozó azt jellemző érték.

Kérjük meg a fordítót, hogy automatikusan példányosítsa a `Show` és `Eq` típusosztályokat.

Szintvonal - tavak

A feladatban a tulajdonságok az alábbiak:

- `Lake` esetén a számérték a tó mélységét jelenti.
- `Mountain` esetén a számérték a hegy magasságát jelenti.
- `Cave` esetén a számérték a barlang hosszát jelenti.

Definiáld a `levelLineL` függvényt, amely listában megkapja a vizsgálandó helyeket és egy mélységet jelölő értéket. A függvény a listából kiválogatja azokat a tavakat, amelyek szigorúan sekélyebbek a paraméterként megadott mélységnél, majd visszaadja azok neveit.

```
levelLineL :: (Num a, Ord a) => [Place a] -> a -> [String]
levelLineL [] 200 == []
levelLineL [(Lake "Huron" 229)] 229 == []
levelLineL [(Lake "Matana" 590)] 589 == []
levelLineL [(Lake "Tanganyika" 676.0)] 800.3 == ["Tanganyika"]
levelLineL [(Lake "Winnipeg" 31.3)] (-15.6) == []
levelLineL [(Mountain "Denali" 6194)] 229 == []
levelLineL [(Mountain "Mount Everest" 8846)] 9321 == []
levelLineL [(Cave "Sistema Huautla" 10010)] 978321 == []
levelLineL [(Lake "Winnipeg" 31),(Mountain "Denali" 6194)] 98765 == ["Winnipeg"]
levelLineL [(Cave "Mammoth Cave" 685.6), (Mountain "Aconcagua" 6959)] 7123.3 == []
levelLineL [(Lake "Great Bear Lake" 373), (Lake "Titicaca" 177)] 450 == ["Great Bear
Lake","Titicaca"]
levelLineL [(Lake "Great Bear Lake" 373), (Lake "Titicaca" 177)] 200 == ["Titicaca"]
levelLineL [(Lake "Great Bear Lake" 373), (Lake "Titicaca" 177)] 100 == []
```

```
levelLineL [(Lake "Matana" 590), (Lake "Ontario" 244), (Lake "Winnipeg" 31) ] 200 ==
["Winnipeg"]
levelLineL [(Lake "Matana" 590), (Lake "Ontario" 244), (Lake "Winnipeg" 31), (Lake "Great
Bear Lake" 373) ] 521 == ["Ontario","Winnipeg", "Great Bear Lake"]
levelLineL [(Cave "Mammoth Cave" 685.6), (Lake "Matana" 590), (Mountain "Denali" 6194),
(Lake "Winnipeg" 31), (Lake "Great Bear Lake" 373), (Cave "Sistema Huautla" 10010)] 521 ==
["Winnipeg", "Great Bear Lake"]
take 10 (levelLineL [(Lake [s] n) | s <- (cycle ['A'..'Z']), n <- [100,300..700]] 450) ==
["A","A","B","B","C","C","D","D","E","E"]
take 10 (levelLineL (concat [[(Cave [s] n),(Mountain [s] n),(Lake [s] n)] | s <- (cycle
['A'..'Z']), n <- [100,300..700]]) 450) == ["A","A","B","B","C","C","D","D","E","E"]
```