Feladatok

Üres lista sorszáma (1 pont)

Adjuk meg azt a függvényt, amely kap két listát paraméterül és Just-ba csomagolva megadja az üres lista sorszámát. Amennyiben mindkettőben van elem, úgy az eredmény legyen Nothing. Ha mindkettő üres, akkor az 1-es sorszámot adjuk vissza.

A sorszámozást kezdjük 1-től.

Segítség: Használjunk mintaillesztést.

whichIsEmpty :: [a] -> [b] -> Maybe Int

whichIsEmpty [] "alma" == Just 1

whichIsEmpty "alma" [] == Just 2

whichIsEmpty [] [] == Just 1

whichIsEmpty [12,23] ['a'] == Nothing

whichIsEmpty [12,23..] [] == Just 2

whichIsEmpty [] [12,23..] == Just 1

Dominó illesztés (1 pont)

Adott két dominó lapka rendezett párként reprezentálva. Adjuk meg azt a függvényt, amely eldönti, hogy a két lapka illeszthető-e egymáshoz! A válasz megadásához vegyük figyelembe, hogy a lapkák tetszőlegesen elforgathatóak.

match :: Eq a => (a,a) -> (a, a) -> Bool

match (2,0) (2,3) == True

match (2,0) (1,2) == True

match (2,4) (4,2) == True

match (2,5) (4,5) == True

match (2,6) (6,6) == True

match (3,4) (3,3) == True

match (0,1) (2,3) == False

match (2,2) (1,3) == False

match (2,2) (3,3) == False

Üres listák indexe (2 pont)

Adott egy tetszőleges listákat tartalmazó lista. Adjuk meg azt a függvényt, amelyik megadja a listában található üreslisták indexeit! Feltehetjük, hogy a paraméterül kapott lista véges, de a benne lévő belső listák között lehet végtelen is.

Az indexelést kezdjük 1-től.

indicesOfEmpties :: Eq a => [[a]] -> [Int]

indicesOfEmpties [] == []

indicesOfEmpties [[]] == [1]

indicesOfEmpties [[],[]] == [1,2]

indicesOfEmpties [[],[1..10], [], []] == [1,3,4]

indicesOfEmpties [[],[1..], [], []] == [1,3,4]

indicesOfEmpties [[],[1..], [], [], [2,3], [212], []] == [1,3,4,7]

indicesOfEmpties [[1,2,3,4],[5,6],[7,8,9],[10]] == []

indicesOfEmpties ["[1,2,3,4]","[5,6]","[7,8,9]","","[10]",""] == [4,6]

indicesOfEmpties [[10..],[1..], [12..], [20,40,60], [2,3], [212], [0..]] == []

Szavak módosítása (2 pont)

Adott egy leképezés, amely egy szó módosítására alkalmas. Adjuk meg azt a függvényt, amely egy mondat minden szavára alkalmazza a megadott módosító leképezést! A mondatról feltehető, hogy véges.

Segítség: Használhatjuk a words függvényt.

applyOnWords :: (String -> String) -> String -> String

applyOnWords (++ ", ") "" == ""

applyOnWords (take 1) "Az ipafai papnak fapipaja van, ezert az ipafai fapipa - papi fapipa." == "A i p f v e a i f - p f"

applyOnWords (\(x:xs) -> 'E':xs) "Az ipafai papnak fapipaja van, ezert az ipafai fapipa - papi fapipa." == "Ez Epafai Eapnak Eapipaja Ean, Ezert Ez Epafai Eapipa E Eapi Eapipa."

applyOnWords (drop 1) "En elmentem a vasarba fel penzzel!" == "n lmentem asarba el enzzel!"

applyOnWords (\a -> a ++ a) "A barack nem esett messze a fatol." == "AA barackbarack nemnem esettesett messzemessze aa fatol.fatol."

applyOnWords (\a -> take (length a div 2) a) "Hosszabb szavak vannak ezen szovegben." == "Hoss sza van ez szove"

Amíg... (2 pont)

Definiáld a restUntil függvényt, amely egy lista elejéről indulva, elhagyja annak elemeit addig, amíg az aktuális fejelemre és a fennmaradó listára nem teljesül a paraméterül kapott tulajdonság.

restUntil :: (a -> [a] -> Bool) -> [a] -> [a]

restUntil (\x xs -> True) [1,2] == [1,2]

restUntil (\x xs -> even x && null xs) [2] == [2]

restUntil (\x y -> odd x && all even y) [] == []

restUntil (\x y -> x == 'l' && length y == 2) "alma" == "lma"

restUntil (\x y -> x == 'r' && length y == 2) "barack" == []

restUntil (\x y -> length y == 6) "sok leveles almafa" == " almafa"

restUntil (\x y -> even x && all odd y) [1,2,3,6,7,9,11,13,15] == [6,7,9,11,13,15]

restUntil (\x y -> odd x && all odd y) [1,2,3,6,7,9,11,13,15] == [7,9,11,13,15]

restUntil (const null) "barack" == "k"

take 20 (restUntil (\x y -> x > 10 && not (null y)) [1..]) == [11..30]

take 8 (restUntil (\x y -> x mod 4 == 0 && sum (take 3 y) > x+200) [2,4..]) == [96,98,100,102,104,106,108,110]

restUntil (\x y -> x == last y && y == reverse y) [1,2,3,1,1,2,2,1] == [1,1,2,2,1]

Összes előfordulás cseréje (2 pont)

Cseréljük le egy elem összes előfordulását egy listában egy másik elemre! Feltehető, hogy a lista véges.

replaceAll :: Eq a => a {-mit-} -> [a] -> a {-mire-} -> [a]

replaceAll 1 [] 2 == []

replaceAll 1 [23,21,23,123,3,2,1,1,23,1] 2 == [23,21,23,123,3,2,2,2,23,2]

replaceAll 'a' "Az ipafai papnak fapipaja van, ezert az ipafai fapipa - papi fapipa." 'e' == "Az ipefei pepnek fepipeje ven, ezert ez ipefei fepipe - pepi fepipe."

replaceAll False [False, False, False] True == [True, True, True]

replaceAll 50 [44,45,46,47,48,49] 100 == [44,45,46,47,48,49]

replaceAll 99 [100, 111, 122, 100] 100 == [100, 111, 122, 100]

Nagybetűt tartalmazó szavak (2 pont)

Add meg egy szövegnek azokat a szavait, amik tartalmaznak nagybetűt! Feltehető, hogy a szöveg véges.

Segítség: Használhatjuk a words, és a Data.Char modul isUpper függvényét.

listWordsWithUpper :: String -> [String]

listWordsWithUpper "Az alma NEM esik messze." == ["Az", "NEM"]

listWordsWithUpper "Haskell is tHe BEST" == ["Haskell", "tHe", "BEST"]

listWordsWithUpper "egy meg egy" == []

listWordsWithUpper "" == []

listWordsWithUpper " " == []

listWordsWithUpper " Alma bArAcK " == ["Alma","bArAcK"]

listWordsWithUpper " alma szilva baracK " == ["baracK"]

Feltételes transzformáció (2 pont)

Definiálj egy függvényt, amely egy predikátumot, egy függvényt és egy listát kap paraméterül. A definiálandó függvény a feldolgozást a lista elejéről kezdi és a paraméterül kapott függvényt addig alkalmazza az elemeken, amíg azok megfelelnek a feltételnek. Amint egy elem nem teljesíti a megadott tulajdonságot, az elem és az összes ezt követő maradjon változatlan, így adjuk vissza a listát.

applyWhile :: (a -> Bool) -> (a -> a) -> [a] -> [a]

applyWhile (<5) (+3) [1,2,3,4,5,6,7,8,9] == [4,5,6,7,5,6,7,8,9]

applyWhile (>10) (+ (-1)) [1,2,3] == [1,2,3]

applyWhile isUpper toLower "ALMAfa SZILVA" == "almafa SZILVA"

take 10 (applyWhile (\x -> x mod 5 == 0) (div 5) [0,5..]) == [0..9]

take 15 (applyWhile (< 10) (div 2) [0..]) == [0,0,1,1,2,2,3,3,4,4,10,11,12,13,14]

map (\f -> f 3) (applyWhile (const True) (\f y -> f (y+1)) [(+1),(*2)]) == [5,8]

Elemek cseréje (2 pont)

Adjuk meg azt a függvényt, amely értékeket lecserél alapértelmezettekre, ha azok nem felelnek meg a feltételnek! A csere úgy valósul meg, hogy amennyiben a paraméterül kapott függvény nem teljesül az első lista elemére, úgy a második lista soron következő értékét fűzi be helyette a listába. Amennyiben a második lista elemei elfogytak, úgy ne változtasson a feldolgozandó listán. Feltehető, hogy az első lista véges.

replaceWithDefIfNot :: (a -> Bool) -> [a] -> [a] -> [a]

replaceWithDefIfNot even [] [] == []

replaceWithDefIfNot odd [] [1..10] == []

replaceWithDefIfNot odd [1,3] [] == [1,3]

replaceWithDefIfNot even [1..10] [2,4] == [2,2,4,4,5,6,7,8,9,10]

replaceWithDefIfNot even [1..10] [2,4..] == [2,2,4,4,6,6,8,8,10,10]

replaceWithDefIfNot odd [1..10] [1,3..] == [1,1,3,3,5,5,7,7,9,9]

replaceWithDefIfNot (>3) [1..10] [1,3..] == [1,3,5,4,5,6,7,8,9,10]

Felváltva alkalmazás (2 pont)

Adjuk meg azt a függvényt, amely két paraméterül kapott függvényt felváltva alkalmaz egy lista elemein! Azaz, a lista első elemére alkalmazza az első paraméterként kapott függvényt, majd a másodikra a másodikként kapott függvényt, a harmadikra ismét az első függvényt, stb.

applyAlternately :: (a -> b) -> (a -> b) -> [a] -> [b]

Megjegyzés: A tesztek kiértékeléséhez szükséges importálni a Data.Char modult.

applyAlternately odd even [] == []

applyAlternately odd even [1] == [True]

applyAlternately odd even [1,2] == [True,True]

applyAlternately odd even [0..5] == [False,False,False,False,False,False]

applyAlternately even odd [0..10] == [True,True,True,True,True,True,True,True,True,True,True]

applyAlternately toUpper toLower "almafa" == "AlMaFa"

take 15 (applyAlternately (+1) (*2) [1..]) == [2,4,4,8,6,12,8,16,10,20,12,24,14,28,16]

Cipzározás másképp (2 pont)

Add meg a cipzározás egy speciális változatát, amely akkor is elkészíti a sorozatot, amikor az egyik lista már elfogyott! Az eredmény lista rendezett párjainak komponensei Maybe a vagy Maybe b típusú értékek lesznek, így a rendezett párban az elem hiányát Nothing-al tudjuk jelezni. A listaelemek Just konstruktorba legyenek csomagolva.

zipMaybe :: [a] -> [b] -> [(Maybe a, Maybe b)]

zipMaybe [] [] == []

zipMaybe [1] [] == [(Just 1, Nothing)]

zipMaybe [] ['a'] == [(Nothing, Just 'a')]

zipMaybe [1] ['a'] == [(Just 1, Just 'a')]

zipMaybe [1,2] ['a'] == [(Just 1, Just 'a'), (Just 2, Nothing)]

zipMaybe [1..5] ['a'..'f'] == [(Just 1,Just 'a'),(Just 2,Just 'b'),(Just 3,Just 'c'),(Just 4,Just 'd'),(Just 5,Just 'e'),(Nothing,Just 'f')]

zipMaybe [1..] [0,-3..] !! 20 == (Just 21,Just (-60))

zipMaybe [1..] "Rovid mondat." !! 20 == (Just 21,Nothing)

zipMaybe [1..] "Rovid mondat." !! 10 == (Just 11,Just 'a')

zipMaybe "Meg kisebb!" [0,2..] !! 20 == (Nothing,Just 40)

zipMaybe "Meg kisebb!" [0,2..] !! 8 == (Just 'b', Just 16)

Hőmérséklet

Adattípus definiálása (1 pont)

Definiálj egy Temperature adatszerkezetet a levegő hőmérséklet mérésekhez! A hőmérsékletet mérjük nappal és éjszaka, ezért az alábbi konstruktorokat vegyük fel:

Night: a mérést éjszaka történt, egy Double paramétere legyen,

Daytime : a mérést nappal történt, egy Double paramétere legyen.

Kérjünk a fordítótól automatikus példányosítást a Show és Eq típusosztályokra!

Nappali mérés (1 pont)

Döntsd el egy mérésről, hogy nappal történt-e!

isDaytime :: Temperature -> Bool

isDaytime (Daytime 15)

isDaytime (Daytime 0)

isDaytime (Daytime (-2))

not (isDaytime (Night (-4)))

not (isDaytime (Night 0))

not (isDaytime (Night 2))

Átlaghőmérséklet (2 pont)

Adott egy méréssorozat. Állapítsd meg a nappal mért hőmérsékletek átlagát!

Feltesszük, hogy van legalább egy nappali mérés a listában és a lista véges.

daytimeAvg :: [Temperature] -> Double

daytimeAvg [Night (-5), Night (-6), Daytime 0, Daytime 3, Daytime 5, Daytime 1, Night (-7)] == 2.25

daytimeAvg [Night 5, Night 0, Daytime 1, Daytime 10, Daytime 8, Daytime 5, Night 2] == 6.0

daytimeAvg [Night 3, Night 0, Daytime 1, Daytime 10, Daytime 8, Daytime 15, Night 7] == 8.5

daytimeAvg [Daytime (-4), Night 0] == -4.0

Nyomda betűkészlete (3 pont)

Adott egy szöveg, melyet ki szeretnénk nyomtatni, továbbá egy betűkészlet. Feltehető, hogy minden betűből legfeljebb egy darab van és mind kisbetű a betűkészletben. Adjuk meg azt a függvényt, amely eldönti, hogy a megadott mondat kinyomtatható-e csupa kisbetűből! Ha a mondatban nagybetű van, akkor azt lehet kisbetűként is nyomtatni. Amennyiben nincs hiány, azaz a mondat kinyomtatható, az eredmény legyen Nothing. Ha valamilyen betű hiányzik, akkor azokat adjuk vissza Just konstruktorba csomagolva!

A listáról feltehetjük, hogy:

- véges,
- csak betűt tartalmaz, egyéb karaktereket nem (pl.szóköz).

Megjegyzés: A visszaadott betűk értelemszerűen legyenek kisbetűk és csak egyszer szerepeljen mindegyik, de a sorrendjük nem számít.

Segítség: Használhatjuk a Data.List modul nub függvényét, illetve a Data.Char modul toLower függvényét.

lackOfLetters :: String -> [Char] -> Maybe [Char]

Megjegyzés: A tesztesetek futtatásához szükséges a Data.List és a Data.Maybe modult importálni.

lackOfLetters "" "" == Nothing

lackOfLetters "" "almf" == Nothing

lackOfLetters "AlmaFa" "almf" == Nothing

lackOfLetters "FaPipa" ['a'..'z'] == Nothing

lackOfLetters "TheQuickBrownFoxJumpsOverTheLazyDog" (['a'..'z']) == Nothing

sort (fromJust (lackOfLetters "AlmaFa" "aeiou")) == (sort "lmf")

sort (fromJust (lackOfLetters "FaPipa" ['a'..'g'])) == (sort "pi")

sort (fromJust (lackOfLetters "FaPipa" ['g'..'z'])) == (sort "af")

sort (fromJust (lackOfLetters "FaPipa" ['g'..'m'])) == (sort "afp")

sort (fromJust (lackOfLetters "TheQuickBrownFoxJumpsOverTheLazyDog" ['b'..'l'])) == (sort "tqurownxmpsvazy")

Fixpont keresése (3 pont)

Definiáljuk azt a függvényt, amely a paraméterül kapott függvényről eldönti, hogy a szintén paraméterben kapott lépésszámon belül eléri-e a fixpontját egy adott pontból indulva! Ha igen, akkor adjuk meg egy Just konstruktorba csomagolva, hogy hányadik lépésben történt ez meg. Ha ez már a kezdeti állapotban is fennáll, ezt a 0. lépésnek tekintjük. Ha a függvény nem jut fixpontba, úgy az eredmény legyen Nothing.

Fixpont: Fixpontnak azt nevezzük, ahol a függvény ugyanazt az értéket rendeli a paraméteréhez, mint a paraméter maga. Vagyis azt a pontot, amelyet a leképezés helyben hagy. Például, az abs függvény a 3 értékre a 3 eredményt adja, azaz ez egy fixpontja.

Segítség: Használjunk segédfüggvényt.

fixedPointIn :: Eq a => (a -> a) -> a -> Int -> Maybe Int

```
fixedPointIn (\x -> x) 3 0 == Just 0

fixedPointIn abs (-3) 0 == Nothing

fixedPointIn abs (-3) 1 == Just 1

fixedPointIn abs (-3) 4 == Just 1

fixedPointIn abs (-3) (-1) == Nothing

fixedPointIn (\x -> if x == 0 then 0 else x div 2) (-3) (-1) == Nothing

fixedPointIn (\x -> if x == 0 then 0 else x div 2) (-3) 4 == Just 2

fixedPointIn (drop 2) [1..] 5 == Nothing

fixedPointIn (drop 2) [1..20] (-10) == Nothing

fixedPointIn (drop 2) [1..20] 9 == Nothing

fixedPointIn (drop 2) [1..20] 10 == Just 10

fixedPointIn (drop 2) [1..20] 13 == Just 10
```