

Csoportok

Funkcionális programozás
IP-18FUNPEG 2
Funkcionális programozás
IP-18FUNPEG 604

Vizsga: 2024.06.04.

Kategória:	Vizsgafeladatok
Elérhető:	2024. 06. 04. 11:03
Pótolható határidő:	
Végső határidő:	2024. 06. 04. 12:33
Kiírta:	Bozó István

Leírás:

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 típuszignatúrának megfelelően kell megoldani. A típuszignatúra nem változtatható meg. Megváltoztatott típuszignatúra 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 teszeseten 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!*

Visual Studio Code

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

Telepítés:

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

Feladatok

Páros index szűrése (2 pont)

Írjunk egy olyan függvényt, amely két listát kap paraméterül és az eredménylistában a páros indexű elemeket tartja meg úgy, hogy az elemeket az eredményben felváltva rendre az egyik, illetve másik listából kapjuk. Ha az egyik

lista elfogy, akkor a másik lista maradék páros indexű elemeit kell hozzáfűzni még.

Az *indexelést* **1** -től kezdjük.

```
keepEvenIndexes :: [a] -> [a] -> [a]
```

```
keepEvenIndexes [1,2,3,4,5] [6,7,8,9,10] == [2,7,4,9]
keepEvenIndexes ["alma", "körte", "barack", "szilva"] ["citrom", "narancs",
keepEvenIndexes [10,20,30,40] [50,60,70] == [20,60,40]
keepEvenIndexes [1,2,3] [] == [2]
keepEvenIndexes [] [] == []
keepEvenIndexes [] [2.3,5.6,2.4,7.8] == [5.6,7.8]
keepEvenIndexes [True] [False] == []
take 11 (keepEvenIndexes [1..] [1,3..]) == [2,3,4,7,6,11,8,15,10,19,12]
take 14 (keepEvenIndexes [1..] [10,20,30,40]) == [2,20,4,40,6,8,10,12,14,16,
take 8 (keepEvenIndexes [9,7,3,2] [10..]) == [7,11,2,13,15,17,19,21]
```

Növekvő lista (2 pont)

Definiáljuk az **isIncreasing** nevű függvényt, amely egy listáról eldönti, hogy növekvő sorrendben tartalmazza-e az elemeket. **Nem** megengedett az egyenlőség, azaz szigorú monotonitást várunk el.

```
isIncreasing :: (Ord a) => [a] -> Bool
```

```
isIncreasing []
isIncreasing [1]
isIncreasing ["abc"]
isIncreasing [2,3]
not (isIncreasing [2,2])
not (isIncreasing [3,2])
isIncreasing ['a','b','c']
isIncreasing [Just (-1),Just 1]
not (isIncreasing [1,2,3,2,5,6,7])
isIncreasing [-3,-2,-1]
not (isIncreasing [x | y <- [1..], x <- [1..y]])
```

Összegek (2 pont)

Egy számokat tartalmazó listában döntsük el, hogy az első negatív érték előtt lévő számok összege a nagyobb, vagy az azt követőeké. A listáról feltehető, hogy véges.

Megjegyzés: a listában biztosan lesz egy negatív szám! Továbbá ha az első negatív előtt vagy után nem szerepel szám, az összeg 0-nak tekintendő.

```
sumLargerBefore :: (Ord a, Num a) => [a] -> Bool
```

```
sumLargerBefore [1,-1]
not (sumLargerBefore [-1,1])
not (sumLargerBefore [1,-1,1])
not (sumLargerBefore [1,3,-1,1,5])
sumLargerBefore [1,3,-1,1,-4,5]
sumLargerBefore [7,1,3,-1,2,5]
not (sumLargerBefore [-3])
sumLargerBefore [-3,-4,3]
```

Tuple összegzés (2 pont)

Adjuk össze egy számpárokat tartalmazó lista elemeit úgy, hogy az eredmény egy pár legyen. Az eredmény első kompanse a listában található párok első komponenseinek összege, míg a második a párok második komponenseinek összege legyen. A listáról feltehető, hogy véges.

```
sumTuples :: Num a => [(a, a)] -> (a, a)
```

```
sumTuples [] == (0,0)
sumTuples [(1,2), (1,2)] == (2,4)
sumTuples [(1,2), (1,1), (10,11)] == (12,14)
sumTuples (replicate 10 (1,1)) == (10,10)
sumTuples [(3,2), (1,3), (-4,5), (9, -20)] == (9,-10)
```

Élő fák (2 pont)

Adott egy lista, amely rendezett párokban fákról tartalmaz információkat. A rendezett pár első eleme a fa korát, a második pedig a fajtáját határozza meg. A fa fajtája egy `Just` konstruktorban adott szöveggént, amennyiben a fa élő, illetve `Nothing`, ha a fa már kiszáradt.

Definiáljuk azt a függvényt, amely megnöveli eggyel az élő fák korát (az első komponens a `tuple`-ban), és kihagyja a listából azokat, amelyek már nem élnek (`tuple` második eleme).

```
aliveTrees :: [(Int, Maybe String)] -> [(Int, Maybe String)]
```

```
aliveTrees [(5, Just "Birch"), (1, Nothing)] == [(6, Just "Birch")]
aliveTrees [] == []
aliveTrees [(0, Just "Birch"), (1, Nothing)] == [(1, Just "Birch")]
aliveTrees [(5, Just "Birch"), (2, Just "Oak"), (3, Just "Oak"), (3, Just "Birch")] == [(6, Just "Birch"), (3, Just "Oak"), (4, Just "Oak")]
take 10 (aliveTrees (zip [0..] (cycle [Just "Oak", Nothing, Just "Birch", Just "Oak"]))) == [(6, Just "Birch"), (3, Just "Oak"), (4, Just "Oak"), (5, Just "Oak"), (6, Just "Oak"), (7, Just "Oak"), (8, Just "Oak"), (9, Just "Oak"), (10, Just "Oak"), (11, Just "Oak")]
```

Részlisták (2 pont)

Definiáljuk a `separate` függvényt, amely egy listából megadott hosszúságú részlistákat képez. A függvény első paramétere az előállítandó részlisták hossza, a második pedig a kezdőelemek "indexeinek távolsága" az eredeti listában. Azaz, a második paraméter többszörösei adják meg, hogy az eredeti listából mely elemek lesznek azok, amelyek részlisták kezdő elemei lesznek. Amennyiben a paraméterül kapott lista üres, úgy a függvény eredménye legyen üres lista.

Példa: ha a `separate 2 3 [1,2,3,4,5,6,7,8,9,10]` függvényhívás tekintjük, akkor:

- a szeletek hossza **2**,
- a részsorozatok kezdőelemeinek indexének távolsága **3**.

A szeletek ennek megfelelően: `[1,2]`, `[4,5]`, `[7,8]` és `[10]`

Megjegyzések:

- Ha két szám bármelyike vagy akár mindkettő negatív, akkor kezeljük úgy, mintha 0 lenne.

```
separate :: Int -> Int -> [a] -> [[a]]
```

```
separate 4 1 [] == []
separate 2 2 [2,3,4] == [[2,3],[4]]
separate 2 3 [1,2,3,4,5,6,7,8,9,10] == [[1,2],[4,5],[7,8],[10]]
take 4 (separate 3 4 [1..]) == [[1,2,3],[5,6,7],[9,10,11],[13,14,15]]
separate 2 1 ["alma","krumpli","paprika"] == [["alma","krumpli"],["krumpli","paprika"]]
separate 2 5 [2] == [[2]]
take 3 (separate 0 3 [2..]) == [[],[],[[]]]
take 3 (separate 2 1 [0,5..]) == [[0,5],[5,10],[10,15]]
separate 2 4 "krumplisteszta" == ["kr","pl","te","ta"]
take 3 (separate 2 0 [2,3,4]) == [[2,3],[2,3],[2,3]]
```

Horgászverseny (3 pont)

A MOHOSZ által szervezett feeder verseny egyik fordulójában elért eredményeket szeretnénk összesíteni. Ehhez bevezetjük az adatok reprezentálására szolgáló adatszerkezetet és az összesítő függvényt is definiáljuk.

Hal reprezentálása

- Add meg a **Suly** típusszinonimát, amely a kifogott hal súlyát fogja megadni grammban (**Integer**).
- Definiáld a **Hal** adattípust, aminek négy konstruktora van : **Keszeg**, **Harcsa**, **Karasz**, **Ponty** :: **Suly** -> **Hal** . Kérd meg a fordítót, hogy példányosítsa az **Eq** és **Show** típusosztályokat az új típusra.

Horgász hely reprezentálása

Vezesd be a szák **Szak** típuszinonimát, amely a halak listáját fogja reprezentálni.

Definiáld a **Horgaszhely** adattípust, aminek egyetlen konstruktora a **Versenyző**.
A konstruktornak az alábbi adattagjai legyenek:

- **String** - amelyben a horgász nevét lehet megadni,
- **Int** - a versenypályán a horgászhely sorszámát adja meg,
- **Szak** - ami a horgász által kifogott halak listája.

Kérd meg a fordítót, hogy példányosítsa az `Eq` és `Show` típusosztályokat az új típusra.

Vezesd be a **Verseny** típuszinonímát, amely a **Horgaszhely** -ek listája lesz.

```
(Ponty 5312) == (Ponty 5312)
(Karasz 5312) /= (Ponty 5312)
(Ponty 5312) /= (Ponty 5311)
(Keszeg 120) == (Keszeg 120)
(Keszeg 120) /= (Keszeg 129)
(Keszeg 120) /= (Karasz 120)
(Keszeg 1230) == (Keszeg 1230)
(Keszeg 1230) /= (Harcza 11230)
(Karasz 652) == (Karasz 652)
(Harcza 35456) == (Harcza 35456)
(Versenyzo "A" 1 [(Ponty 5432)]) == (Versenyzo "A" 1 [(Ponty 5432)])
(Versenyzo "A" 1 [(Keszeg 5432)]) /= (Versenyzo "A" 1 [(Ponty 5432)])
(Versenyzo "A" 1 [(Ponty 5432)]) /= (Versenyzo "B" 1 [(Ponty 5432)])
(Versenyzo "A" 1 [(Ponty 5432)]) /= (Versenyzo "A" 2 [(Ponty 5432)])
(Versenyzo "B" 23 [(Harcza 11123)]) == (Versenyzo "B" 23 [(Harcza 11123)])
```

Eredmények összesítése

Definiáld a **summarize** függvényt, ami rendezett hármasok listájaként adja meg, hogy az adott versenyző összesen hány gramm halat fogott a verseny alatt. A rendezett hármasok a horgász nevét, a horgászhely számát és a kifogott halak mennyiségét adják meg. A verseny eredményébe **nem számítanak bele a harcsák**, így ezeket figyelmen kívül kell hagyni! A versenyről feltehető, hogy véges! Továbbá feltehető, hogy az összes versenyző különböző.

```
summarize :: Verseny -> [(String, Int, Suly)]
```

```

summarize [] == []
summarize [(Versenyzo "A" 1 [(Keszeg 5), (Harcsa 7)])] == [("A",1,5)]
summarize [(Versenyzo "A" 1 [(Keszeg 5), (Harcsa 7)]),(Versenyzo "B" 2 [(Karasz 8), (Ponty 11)])] == [("A",1,5),("B",2,19)]
summarize [(Versenyzo "A" 3 [(Keszeg 5), (Harcsa 7)]),(Versenyzo "B" 1 [(Karasz 8), (Ponty 11)])] == [("A",3,16),("B",1,19)]
summarize [(Versenyzo "C" 4 [(Harcsa 5), (Ponty 3)]),(Versenyzo "B" 5 [(Keszeg 9), (Karasz 8)])] == [("C",4,23),("B",5,45)]
summarize [(Versenyzo "AB" 4 []),(Versenyzo "AC" 5 [(Ponty 5)]),(Versenyzo "BC" 6 [(Keszeg 9)])] == [("AB",4,0),("AC",5,25),("BC",6,54)]
summarize [(Versenyzo "AAA" 1 []), (Versenyzo "AAB" 0 []), (Versenyzo "AAC" 0 [])] == []
summarize [(Versenyzo "A" 1 [Keszeg 5, Harcsa 7, Keszeg 9, Karasz 8, Ponty 11])] == [("A",1,35)]
summarize [(Versenyzo "A" 1 [Keszeg 5, Ponty 11, Harcsa 7, Keszeg 9, Karasz 8])] == [("A",1,35)]

```

Páros számjegyek (3 pont)

Definiáljunk egy függvényt, amely megmondja egy számról, hogy hány páros számjegy található benne.

Segítség:

- A feladat megoldásában a **div** és **mod** függvények segítenek. Ne feledjük, hogy 10-es számrendszerben dolgozunk. Például: $324 = 3 * 10^2 + 2 * 10^1 + 4 * 10^0 = 3 * 100 + 2 * 10 + 4$

```
evenDigits :: (Integral a) => a -> Int
```

```
evenDigits 0 == 1
evenDigits 1 == 0
evenDigits 11 == 0
evenDigits 4 == 1
evenDigits 24 == 2
evenDigits 43217843 == 4
evenDigits (-245) == 2
evenDigits 1234567890 == 5
evenDigits (-1234567890) == 5
```

Git tároló

A jelszóvédett feladatok esetében a 'git push' nem használható.

Útvonal:

https://tms.inf.elte.hu/git/6226/w5t3fd/wd82bf46507bfe154a9be50548

Használat:

git clone https://tms.inf.elte.hu/git/6226/w5t3fd/wd82bf46507bfe154a9be50548

Megoldás

 Letöltés

Név: vizsga_vegleges.zip
Feltöltés ideje: 2024. 06. 04. 12:28
Értékelés: 13
Státusz: Elfogadva
Feltöltések száma: 3
Értékelte: Erdei Zsófia
Megjegyzések:

Automatikus tesztelés eredményei



#1

Elért pontszám: 13/18 pont.

Teszteken sikeresen átmenő definíciók: keepEvenIndexes, isIncreasing, sumLargerBefore, sumTuples, aliveTrees, summarize.

A következő konstansokat, függvényeket és adattípusokat nem találni a megoldásban:
separate, evenDigits
Megbukott tesztek:

Mellékelt fájlok

 justusadam.language-haskell-3.6.0.vsix

