Csoportok

Funkcionális programozás
IP-18FUNPEG | 4

Imperatív programozás
IP-18IMPROGEG | 49

Funkcionális programozás
IP-18FUNPEG |

Vizsga - 2022.01.17.

Kategória: Vizsgafeladatok
Elérhető: 2022. 01. 17. 10:00
Pótolható határidő:
Végső határidő: 2022. 01. 17. 12:00
Kiírta:

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 feladatok tetszőleges sorrendben megoldhatóak. A pontozás szabályai a következők:

- Minden teszten átmenő megoldás ér teljes pontszámot.
- Funkcionálisan hibás (valamelyik teszteseten megbukó) megoldás nem ér pontot.
- Fordítási hibás vagy hiányzó megoldás esetén a teljes megoldás 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!

Feladatok

Hármasból szöveg (1 pont)

Definiáljuk azt a függvényt, ami egy rendezett hármasban kapott szövegeket sorban egymás után fűzi!

```
concatTripleString :: ([Char], [Char], [Char]) -> [Char]
```

```
concatTripleString ("","","") == ""
concatTripleString ("Hello"," ","there") == "Hello there"
concatTripleString ("General"," Ke","nobi") == "General Kenobi"
```

Egymás maradékai (1 pont)

Definiáljuk azt a függvényt, ami megadja a két paraméterként kapott érték egymással vett osztási maradékait rendezett párként egy `Just` konstruktorba csomagolva! Amennyiben valamelyik osztás nem végezhető el (`0` -val osztás), akkor az eredmény legyen `Nothing` .

```
mods :: Integral a => a -> a -> Maybe (a, a)
```

Csoportok

Funkcionális programozás
IP-18FUNPEG | 4

Imperatív programozás
IP-18IMPROGEG | 49

Funkcionális programozás
IP-18FUNPEG |

```
mods 5 5 == Just (0, 0)
mods 5 4 == Just (1, 4)
mods 4 5 == Just (4, 1)
mods 11 3 == Just (2, 3)
mods 34 0 == Nothing
mods 0 31 == Nothing
mods 0 0 == Nothing
```

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! Feltehető, hogy a lista véges, de a belső listák lehetnek végtelenek.

Segítség: Használhatunk mintaillesztést.

```
dropEmpties :: Eq a => [[a]] -> [[a]]
```

```
dropEmpties [] == []
dropEmpties [[1,2], [], [], []] == [[1,2]]
dropEmpties [[1,2], [], [], [], [1,5,2,1]] == [[1,2],[1,5,2,1]]
dropEmpties [[]] == []
length (dropEmpties [[1,2], [], [], [], [1..]]) == 2
```

Láncfűzés (2 pont)

Definiáld a `createChain` függvényt, amely létrehoz egy *n* hosszúságú láncot! A láncszemek mindegyike tartalmazza, hogy hányadik elem a sorban.

Megjegyzés: Használhatjuk a `show` függvényt.

```
createChain :: Integer -> String
```

```
createChain (-5) == ""
createChain 0 == ""
createChain 3 == "(1)(2)(3)"
createChain 14 == "(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)"
createChain 15 == "(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)"
```

Váltakozó kis- és nagybetűk (2 pont)

Írjuk meg az `aLtErNaTiNgCaPs` nevű függvényt, amely a paraméterül kapott `String` -et, úgy adja vissza, hogy a kis- és nagybetűk felváltva kövessék egymást, kezdjünk kisbetűvel, a szóköz karaktert is számoljuk bele.

Megjegyzés: Használhatjuk a `toUpper` és `toLower` függvényeket a `Data.Char` modulból.

```
aLtErNaTiNgCaPs :: String -> String
```

```
aLtErNaTiNgCaPs "spongebobsquarepants" == "sPoNgEbObSqUaRePaNtS"
aLtErNaTiNgCaPs "Haskell Is Cool" == "hAsKeLl iS CoOl"
take 25 (aLtErNaTiNgCaPs ['a' ..]) == "aBcDeFgHiJkLmNoPqRsTuVwXy"
```

Vizsgaeredmény (2 pont)

Egy feleletválasztós vizsgán minden feladatra `1` , `-1` , vagy `0` pontot lehet kapni aszerint, hogy a vizsgázó helyesen válaszol, hibásan válaszol, vagy nem válaszol az adott kérdésre. A vizsgázó eredményeit egy `Maybe Bool` (véges) lista tartalmazza. Definiáljuk azt a függvényt, amely ezen lista és a sikeres vizsgához szükséges minimum pontszám alapján megadja, hogy a vizsgázó sikeresen oldotta-e meg a feladatsort!

Ha a vizsgázó elérte a minimum pontszámot, sikeresen teljesítette a vizsgát.

Csoportok

Funkcionális programozás
IP-18FUNPEG | 4

Imperatív programozás
IP-18IMPROGEG | 49

Funkcionális programozás
IP-18FUNPEG |

```
result :: [Maybe Bool] -> Int -> Bool
```

```
result [Just True, Just False, Just True, Just True, Nothing] 1
result [Just True, Just False, Just True, Just True, Nothing] 2
result [Just True, Nothing, Just False, Nothing, Just True, Just True, Just True]
not $ result [Just True, Just False, Nothing, Just True, Just True] 3
result ((replicate 4 (Just True)) ++ replicate 3 Nothing ++ [Just False]) 2
not $ result [Just True, Just False, Nothing, Nothing, Just False, Just True, Jus
```

Maximum, ha... (2 pont)

Definiáld azt a függvényt, amely egy lista adott feltételt teljesítő elemei közül visszaadja a legnagyobbat! Ha nincs ilyen elem, akkor az eredmény legyen `Nothing`, különben az eredményt `Just` -ba csomagolva adja vissza a függvény.

Feltehető, hogy a kapott lista véges.

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

```
maximumIF (<5) [1..10] == Just 4
maximumIF even [1..11] == Just 10
maximumIF (<10) [] == Nothing
maximumIF not [True, False] == Just False
maximumIF (=='x') ['a'..'z'] == Just 'x'
maximumIF even [1,3..10] == Nothing
```

Aláhúzások kitöltése (2 pont)

Definiáld a `fillBlanks` függvényt, ami egy szövegben helyettesíti az aláhúzás karaktereket! A helyettesítést a második paraméterként kapott karakterlista alapján végezzük, az első aláhúzás az első karakterrel legyen helyettesítve, második a másodikkal stb.

Ha nincs elég karakter megadva, hogy az összes aláhúzást kicseréljük, akkor cseréljük ki amit lehet, a maradékot pedig hagyjuk meg `_` -nak.

```
fillBlanks :: String -> String -> String
```

```
lanks "_lma" "a" == "alma"
lanks "a_m_" "lam" == "alma"
lanks "__" "x" == "x_"
lanks "" "abc" == ""
lanks "" "" == ""
lanks "alma_" "" == "alma_"
lanks "alma" "xy" == "alma"
lanks "_askell" (repeat 'H') == "Haskell"
lanks "_ajnalban _askell _amar _elyes" (repeat 'H') == "Hajnalban Haskell Hamar Hely
55 (fillBlanks (cycle "_-") (cycle ['a'..'z'])) == "a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-
5 (fillBlanks (repeat '_') "xyz") == "xyz__"
52 (fillBlanks (concat (repeat "X_")) ['a'..'z']) == concat (map (\c -> 'X':[c]) ['a
54 (fillBlanks (concat (repeat "X_")) ['a'..'z']) == concat (map (\c -> 'X':[c]) ['a
```

Keverés (2 pont)

Definiáld a `riffleShuffle` függvényt, ami egy lista elemeit megkeveri!

Ezt a következő módon teszi: A listát a felénél elvágva két egyenlő részre bontjuk (páratlan hosszú lista esetén az első rész legyen a rövidebb), majd a két részből felváltva véve az elemeket előállítjuk a megkevert listát.

Feltehető, hogy a lista véges hosszú.

```
riffleShuffle :: [a] -> [a]
```

Csoportok

Funkcionális programozás
IP-18FUNPEG | 4

Imperatív programozás
IP-18IMPROGEG | 49

Funkcionális programozás
IP-18FUNPEG |

```
riffleShuffle [1..10] == [1,6,2,7,3,8,4,9,5,10]
iterate riffleShuffle [1..10] !! 6 == [1..10]
riffleShuffle [1,2,3,4,5] == [1,3,2,4,5]
riffleShuffle [] == []
riffleShuffle [5] == [5]
take 3 (iterate riffleShuffle [1..5]) == [[1,2,3,4,5],[1,3,2,4,5],[1,2,3,4,5]]
```

Elemek pozíciói (2 pont)

Definiáld azt a függvényt, amelyik megadja hogy egy érték melyik pozíciókon szerepel a listában! Amennyiben az elem legalább egyszer szerepel, úgy az eredmény listát egy `Just` konstruktorba csomagolva adja meg, ellenkező esetben az eredmény legyen `Nothing`. A lista indexelését `1`-től kezdjük. Feltehető, hogy a lista véges lesz.

```
getPositions :: Eq a => a -> [a] -> Maybe [Int]
```

```
ta szarka farka tarka." == Nothing
ta szarka farka tarka." == Just [3,5]
ta szarka farka tarka." == Just [13,16,19,22,26,29,32,35,38,41,45,48,51,54,57,64,67,
```

Alkalmazások (2 pont)

Adottak függvények és a függvényekhez tartozó bemenetek egy-egy listában. Alkalmazd mindegyik függvényt mindegyik bemenetre! Az első függvényt alkalmazzuk először az összes listaelemre, aztán a másodikat, stb. Tehát, ha a bemenetek listája végtelen, akkor az eredmény az első függvény értékei lesznek (hiszen sosem érünk a bemenetlista végére, ezt írja le az utolsó teszteset).

```
applies :: [a -> b] -> [a] -> [b]
```

```
2),(+1)] [1,2,3] == [2,4,6,2,3,4]
[1,2,3] == []
0)] [] == []
[] == ( [] :: [Integer] )
-" Alma"), (++" Barack")] ["Alma", "Szilva", "Barack"] == ["Alma Alma","Szilva Alma",
olies [(*i) | i <- [0..]] [5,10,15,20]) == [0,0,0,0,5,10,15,20,10,20,30,40,15,30,45,6
olies [(+5), (*0)] [-5..]) == [0..49]
```

Véges listák

Haskellben a listák hosszát költséges kiszámolni, és külön odafigyelést igényel a végtelen listák kezelése.

Adattípus definiálása (1 pont)

Definiáljuk a `FiniteList` típust az `Integer` típusú értékekből álló véges listák reprezentálására. A típus két adatkonstruktorral rendelkezzen:

- `Empty` : amely az üres listának felel meg,
- `NonEmpty` : melynek két paramétere van, a lista hosszát reprezentáló `Int` szám, valamint a véges lista elemeit tartalmazó `Integer` lista.

Kérjünk a fordítótól automatikus példányosítást a `Show` és `Eq` típusosztályokra!

Véges listává alakítás (1 pont)

Definiáljuk a `toFinite` függvényt, mely egy listát véges listává konvertál! Amennyiben a lista a megadott elemszámnál hosszabb, csak a megadott hossznak megfelelő számú elemet tároljuk le. Amennyiben a lista a megadott értéknél rövidebb, úgy a hossz értéket kompenzáljuk ennek megfelelően.

Csoportok

Funkcionális programozás
IP-18FUNPEG | 4

Imperatív programozás
IP-18IMPROGEG | 49

Funkcionális programozás
IP-18FUNPEG |

```
toFinite :: Int -> [Integer] -> FiniteList
```

```
toFinite 100 [1,31,12,2] == NonEmpty 4 [1,31,12,2]
toFinite 100 [1] == NonEmpty 1 [1]
toFinite 100 [] == Empty
toFinite 0 [1..] == Empty
toFinite 2 [1,2,3,4] == NonEmpty 2 [1,2]
toFinite 10 [1..] == NonEmpty 10 [1..10]
toFinite (-5) [1,9,8,4,10] == Empty
toFinite (-9) [] == Empty
toFinite (-12) [1..] == Empty
```

Véges listák összefűzése (2 pont)

Adjuk meg a véges listák összefűzésének műveletét! Feltehetjük, hogy a lista véges elemszámú és az értékek helyesen szerepelnek benne.

Segítség: Definiáljunk egy segédfüggvényt, amely két `FiniteList` -et tud összefűzni.

```
concatFL :: [FiniteList] -> FiniteList
```

```
mpty 2 [0,1],Empty,NonEmpty 1 [0],NonEmpty 4 [0,1,2,3]] == NonEmpty 18 [0,1,0,1,2,3,
```

Karakterláncok különbsége (3 pont)

Definiáljuk azt a függvényt, mely összehasonlítja két `String` azonos pozícióin található karaktereit! Különbözőség esetén egy listában visszaadja az első karakterlánc a másodiktól eltérő karaktereit `Just` -ba csomagolva! Ha a két `String` megegyezik az eredmény legyen `Nothing` A különbséget csak addig kell nézni, amíg az első lista el nem fogy.

```
difference :: String -> String -> Maybe String
```

A tesztesetek futtatásához szükséges a `Data.Maybe` modul importálása.

```
difference "alma" "alma" == Nothing
difference "alma" "asztal" == Just "lma"
difference "asztal" "alma" == Just "sztal"
difference "" "" == Nothing
difference "alma" "" == Just "alma"
difference "" "alma" == Nothing
difference "alma" "almafa" == Nothing
difference "abrakadabra" "abbrakadabra" == Just "rakadabra"
difference (reverse ('b':(replicate 20 'a'))) (replicate 21 'a') == Just "b"
difference ((replicate 10 'a') ++ ('b':(replicate 10 'a'))) (replicate 21 'a')  ==
difference (replicate 21 'a') (reverse ('b':(replicate 20 'a'))) == Just "a"
difference (replicate 20 'a') (reverse ('b':(replicate 20 'a'))) == Nothing
difference (replicate 21 'a') (repeat 'a') == Nothing
take 10 (fromJust (difference ['a'..] "alma")) == "bcdefghijk"
fromJust (difference "alma" ['a'..]) == "lma"
difference "alma" (cycle "alma") == Nothing
take 8 (fromJust (difference (cycle "alma") "alma")) == "almaalma"
```

Többség szerinti szűrés (3 pont)

Definiáld a `filterByMajority` függvényt, amely paraméterül kap egy logikai függvények listáját és egy elemeket listáját. Azokat az elemeket adja eredményül, amelyekre a függvények többsége teljesül, azaz több mint fele *igaz* értéket ad.

Csoportok

Funkcionális programozás
IP-18FUNPEG | 4

Imperatív programozás
IP-18IMPROGEG | 49

Funkcionális programozás
IP-18FUNPEG |

```
filterByMajority :: [(a -> Bool)] -> [a] -> [a]
```

```
filterByMajority [] [] == []
filterByMajority [] [1..10] == [1,2,3,4,5,6,7,8,9,10]
filterByMajority [even] [] == []
filterByMajority [even, (<5)] [1..20] == [2,4]
filterByMajority [even, odd] [1,2,3,4] == []
filterByMajority [id, (>False), (==True)] [True, False] == [True]
filterByMajority [(>n) | n<-[1..10]] [1..12] == [7,8,9,10,11,12]
take 10 (filterByMajority [even, (>2), (==3)] (cycle [0..4])) == [3,4,3,4,3,4,3,4,
```

Feltöltés

1 fájl feltöltése sikeres

Tallózás

Feltöltés

Megoldás

Letöltés

Név: vizsga(LA0YYO).zip
Feltöltés ideje: 2022. 01. 17. 11:52
Értékelés:
Státusz: Sikertelen tesztelés
Értékelte:
Megjegyzések:

Automatikus tesztelés eredményei

```
Elért pontszám: 21/30 pont.

Teszteken sikeresen átmenő definíciók: concatTripleString, mods, dropEmpties,
aLtErNaTiNgCaPs, riffleShuffle, result, getPositions, fillBlanks, applies, FiniteList,
toFinite, difference.

A következő konstansokat, függvényeket és adattípusokat nem találni a megoldásban:
createChain, maximumIF, concatFL, filterByMajority
Megbukott tesztek:
```

Mellékelt fájlok