Csoportok

Vizsga: 2024.06.21.

Kategória:
Vizsgafeladatok
Elérhető:
2024. 06. 21. 11:02
Pótolható határidő:
Végső határidő:
2024. 06. 21. 12:32
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 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!*

Visual Studio Code

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

Fordított hármasok (2 pont)

Definiáljunk egy függvényt, amely három listát kap paraméterül, majd ezek elemeiből rendezett hármasokat alkot párhuzamosan haladva a listán, de az elemek az eredményben a paraméterek típusa szerinti fordított sorrendben szerepeljenek. A függvény a legrövidebb lista hosszáig működjön.

```
reversedTriplets :: [a] -> [b] -> [c] -> [(c, b, a)]
```

```
reversedTriplets [1, 2, 3] ["A", "B", "C"] [True, False, True] == [(True, "A", 1), (False, "B", 2), (True, "C", 3)]
reversedTriplets [] [] [] == ([] :: [((), (), ())])
reversedTriplets [1, 2] ["A", "B", "C"] [True, False] == [(True, "A", 1), (False, "B", 2)]
null (reversedTriplets [] [2] [True, False])
null (reversedTriplets [2] [] [True, False])
null (reversedTriplets [True, False] [2] [])
null (reversedTriplets [True, False] [] [])
null (reversedTriplets [] [True] [])
null (reversedTriplets [] [] [True])
null (reversedTriplets [] [10..] [1..])
reversedTriplets [1..] [2..] [3..10] == [(3,2,1),(4,3,2),(5,4,3),(6,5,4),(7,6,5),(8,7,6),(9,8,7),(10,9,8)]
reversedTriplets [1..] ['a'..'z'] [(Just 5), Nothing, Nothing, (Just 10)] == [(Just 5,'a',1),(Nothing,'b',2),(Nothing,'c',3),(Just 10,'d',4)]
take 6 (reversedTriplets [0..] [10..] [100..]) == [(100,10,0),(101,11,1),(102,12,2),(103,13,3),(104,14,4),(105,15,5)]
```

Nullához képest (2 pont)

Definiáljuk a `byZero` függvényt, amely egy számokat tartalmazó listát kap. Az eredménye egy rendezett párokból álló lista, amelynek egyik tagja a listából való érték, a másik pedig a `0`. A rendezett páron belül az elemek sorrendje a következő szerint alakuljon:

- ha a szám kisebb, mint nulla, akkor az első pozícióra a szám, a másodikra pedig a `0` érték kerül,
- ellenkező esetben a `0` az első pozícióban, a másodikban pedig a szám szerepel.

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

```
byZero [] == []
byZero [0] == [(0,0)]
byZero [-1] == [(-1,0)]
byZero [1] == [(0,1)]
byZero [-1,1] == [(-1,0),(0,1)]
byZero [1,2 :: Integer,3] == [(0,1),(0,2 :: Integer),(0,3)]
byZero [0.00000001 :: Double] == [(0,0.00000001 :: Double)]
byZero [-11,2,-3,0] == [(-11,0),(0,2),(-3,0),(0,0)]
byZero [-110,-2,10,20,-3] == [(-110,0),(-2,0),(0,10),(0,20),(-3,0)]
take 11 (byZero [-5..]) == [(-5,0),(-4,0),(-3,0),(-2,0),(-1,0),(0,0),(0,1),(0,2),(0,3),(0,4),(0,5)]
```

Késések (2 pont)

Egy vasúttársaság pályakarbantartásokat végez a `100` -as számú vonalán, ami miatt azon a vonalon késések várhatóak. Egy listában kapjuk meg a vonatok adatait. Ezek az adatok a vonat vonalszáma, menetideje percekben és az érintett vonalak. Minden olyan vonat menetidejét növeljük meg `10` perccel, amelyek érintik a `100` -as vonalat (vagy a `100` -as vonal maga). A listákról feltehetők, hogy végesek (mind a külső, mind a belső).

Megjegyzés: A megoldáshoz definiálhatunk segédfüggvényt.

```
late :: [(Int,Int,[Int])] -> [(Int,Int,[Int])]
```

```
late [] == []
late [(1,50,[100])] == [(1,60,[100])]
late [(100,40,[101,111])] == [(100,50,[101,111])]
late [(5,23,[1,10,101]),(55,38,[101,125,200])] == [(5,23,[1,10,101]),(55,38,[101,125,200])]
late [(648,145,[100,101]),(548,160,[205,210,200])] == [(648,155,[100,101]),(548,160,[205,210,200])]
late [(648,145,[101,111]),(548,160,[205,210,200])] == [(648,145,[101,111]),(548,160,[205,210,200])]
late [(5,23,[1,10,100]),(55,38,[100,125,200])] == [(5,33,[1,10,100]),(55,48,[100,125,200])]
late [(99,123,[]),(100,60,[101,111]),(233,150,[30,100,87,39]),(33,33,[10,12,20,6])] == [(99,123,[]),(100,70,[101,111]),(233,160,[30,100,87,39]),(3
```

Furcsa összegzés (2 pont)

Definiáljunk egy függvényt, amely egy egészekből álló listát kapva összegzi azt olyan módon, hogy a páros indexűeket kivonja az összegből, a páratlan indexűeket pozitív hozzáadja az összeghez.
Az indexelést `0` -tól kezdjük. A listáról feltehető, hogy véges.

Példa: `strangeSum [9,-3,-6,10] == (-9) + (-3) - (-6) + 10 == 4`

```
strangeSum :: Integral a => [a] -> a
```

```
strangeSum [] == 0
strangeSum [1,2,3,4,5] == -3
strangeSum [5,4,3,2,1] == -3
strangeSum [1,-2,3,-4,5] == -15
strangeSum [1] == -1
strangeSum [1,1] == 0
strangeSum [3,9] == 6
strangeSum [10,20,7] == 3
strangeSum [9,-3,-6,10] == 4
strangeSum [10,40,-9,-7,20,16] == 28
```

Függvény `n` -szeres ismétlése (2 pont)

Definiáljunk egy magasabb rendű függvényt, amely paraméterül kap egy számot (`n`), egy függvényt (`f`) és egy értéket (`e`). Az eredményt az `f` függvény `e` értéken történő `n` -szeri ismételt alkalmazásával állítja elő. Ha a szám `0` -nál kisebb, akkor a függvény viselkedjen ugyanúgy, mintha a szám `0` lenne.

```
applyNTimes :: Integer -> (a -> a) -> a -> a
```

```
applyNTimes 3 (++ " hello") [] == " hello hello hello"
applyNTimes 2 (++ "ha") "" == "haha"
applyNTimes 10 (*2) 2 == 2048
applyNTimes 10 (`div` 2) 2048 == 2
applyNTimes 0 (*10) 2 == 2
applyNTimes 1 (*10) 3 == 30
applyNTimes 5 (+5) 6 == 31
applyNTimes (-2) (+5) 6 == 6
```

Római hadsereg

A római hadsereg szeretné számon tartani, hogy az egyes provinciákban milyen az ellátottság fegyverekből. Ehhez szükséges, hogy az egyes provinciákban mennyi katona állomásozik, illetve mennyi az ott rendelkezésre álló fegyver.

1. rész (2 pont)

Definiáljuk az `Arms` algebrai adattípust, amely a különböző fegyverek típusait és azok számának a nyilvántartására szolgál. Az adatkonstruktorok egy `Int` típusú adattaggal rendelkezzenek, amely segítségével megadható az adott fegyverből rendelkezésre álló mennyiség. A következő fegyvertípusokat adjuk meg: `Sword` , `Spear` , `Bow` .

Definiáljuk a `Province` algebrai adattípust, amely segítségével a provinciák típusát vezetjük be. Az egyszerűség kedvéért csak a következő provinciákat adjuk meg: `Pannonia` , `Dacia` , `Italia` , `Gallia` , `Germania` .

Az adatkonstruktoroknak egy adattagja legyen, egy `Int` típusú érték, amely a provinciában állomásozó katonák számát adja meg.

Kérjük meg a fordítót, hogy mindkét új típusunkra példányosítsa a `Show` és `Eq` típusosztályokat.

Adjuk meg az `armySize` függvényt, amely a provincián belül állomásozó sereg méretét adja meg.

```
armySize :: Province -> Int
```

```
armySize (Pannonia 400) == 400
armySize (Dacia 521) == 521
armySize (Italia 890) == 890
armySize (Gallia 987) == 987
armySize (Germania 120) == 120
```

Adjunk meg az `armsNum` függvényt, amely az egyes fegyverek esetén megadja azok számosságát.

```
armsNum :: Arms -> Int
```

```
armsNum (Sword 231) == 231
armsNum (Spear 12) == 12
armsNum (Bow 64) == 64
```

2. rész (3 pont)

A római hadsereg ellátási gondokkal küzd, emiatt a helyőrségi katonák közül nem mindenkinek jutott fegyver. Definiáljuk a `deficiency` függvényt, amely kiszűri azon provinciákat, amelyekben nincs elegendő fegyver a katonáknak. Az eredmény egy rendezett párokból álló lista legyen, ahol az első elem egy `Province` típusú érték, a második egy egész érték amely a hiányzó fegyverek számát adja meg. A provinciák száma és a fegyverek száma is véges.

```
deficiency :: [(Province, [Arms])] -> [(Province, Int)]
```

```
deficiency [] == []
deficiency [(Pannonia 952, [Sword 800, Spear 150]), (Italia 700, [Sword 500, Spear 150])] == [(Pannonia 952,2),(Italia 700,50)]
deficiency [(Pannonia 952, [Sword 800, Spear 150]), (Gallia 650, [Sword 500, Spear 150]), (Italia 700, [Sword 500, Spear 150])] == [(Pannonia 952,
deficiency [(Dacia 453, [Bow 32, Spear 42, Sword 380]),(Pannonia 952, [Sword 802, Spear 150]), (Italia 1500, [Spear 890, Sword 430, Bow 452]), (Ga
deficiency [(Dacia 453, [Bow 32, Spear 42, Sword 370]),(Pannonia 952, [Sword 802, Spear 150]), (Italia 1500, [Spear 650, Sword 430, Bow 419]), (Ga
```

Olvashatóbb számok (3 pont)

Definiáljuk a `decimalPlaces` függvényt, amely egy paraméterül kapott egész számot tesz olvashatóbbá úgy, hogy hátulról számítva 3 számjegyenként tagolja azt egy `.` beillesztésével. A függvény működjön negatív számokra is.

Segítség: Egy számot, a `show` függvény segítségével könnyedén `String` típusú értékké alakíthatunk.

```
decimalPlaces :: (Integral a, Show a) => a -> String
```

```
decimalPlaces 0 == "0"
decimalPlaces (-0) == "0"
decimalPlaces 1 == "1"
decimalPlaces 9 == "9"
decimalPlaces (-82) == "-82"
decimalPlaces 123 == "123"
decimalPlaces 630 == "630"
decimalPlaces 1234 == "1.234"
decimalPlaces 7348 == "7.348"
decimalPlaces 12345 == "12.345"
decimalPlaces 123456 == "123.456"
decimalPlaces 1234567 == "1.234.567"
decimalPlaces (-12) == "-12"
decimalPlaces (-123) == "-123"
decimalPlaces (-1234) == "-1.234"
decimalPlaces (-12345) == "-12.345"
decimalPlaces (-123456) == "-123.456"
decimalPlaces (-1234567) == "-1.234.567"
decimalPlaces 3456789123456789  == "3.456.789.123.456.789"
decimalPlaces 23456789123456789  == "23.456.789.123.456.789"
decimalPlaces 123456789123456789  == "123.456.789.123.456.789"
```

Git tároló

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

Útvonal:

`https://tms.inf.elte.hu/git/6253/vl2l98/w03712177aae343cd48b50d8b9`

Használat:

`git clone https://tms.inf.elte.hu/git/6253/vl2l98/w03712177aae343cd48b50d8b9`

Megoldás

Név:
Vizsga.zip
Feltöltés ideje:
2024. 06. 21. 12:21
Értékelés:
Státusz:
Sikertelen tesztelés
Feltöltések száma:
1

Értékelte:

Megjegyzések:

Automatikus tesztelés eredményei

#1

```
Elért pontszám: 4/18 pont.

Teszteken sikeresen átmenő definíciók: reversedTriplets, armsNum.

A következő konstansokat, függvényeket és adattípusokat nem találni a megoldásban: byZero, strangeSum, applyNTimes, deficiency,
decimalPlaces
Megbukott tesztek:

### late tesztjei:
##
## Kivétel:
Submission.hs:(26,1)-(29,93): Non-exhaustive patterns in function late

## Teszteset:
late [(5,23,[1,10,101]),(55,38,[101,125,200])] == [(5,23,[1,10,101]),(55,38,[101,125,200])]



##
## Kivétel:
Submission.hs:(26,1)-(29,93): Non-exhaustive patterns in function late

## Teszteset:
late [(648,145,[100,101]),(548,160,[205,210,200])] == [(648,155,[100,101]),(548,160,[205,210,200])]



##
## Kivétel:
Submission.hs:(26,1)-(29,93): Non-exhaustive patterns in function late

## Teszteset:
late [(648,145,[101,111]),(548,160,[205,210,200])] == [(648,145,[101,111]),(548,160,[205,210,200])]



##
## Kivétel:
Submission.hs:(26,1)-(29,93): Non-exhaustive patterns in function late

## Teszteset:
late [(5,23,[1,10,100]),(55,38,[100,125,200])] == [(5,33,[1,10,100]),(55,48,[100,125,200])]



##
## Kivétel:
Submission.hs:(26,1)-(29,93): Non-exhaustive patterns in function late

## Teszteset:
late [(99,123,[]),(100,60,[101,111]),(233,150,[30,100,87,39]),(33,33,[10,12,20,6])] == [(99,123,[]),(100,70,[101,111]),
(233,160,[30,100,87,39]),(33,33,[10,12,20,6])]
```

Mellékelt fájlok

justusadam.language-haskell-3.6.0.vsix