Csoportok

Minta vizsga

Kategória:
Órai munkák
Végső határidő:
5/31/2025, 11:59 PM (Beadva, Lejár 6 nap múlva)
Próbálkozások száma:
Korlátlan
Kiírta:
Erdei Zsófia

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 érvényes vizsgához!

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, ezért 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

Verseny (2 pont)

Egy ügyességi versenyen időre kell egy akadályokkal teli pályán végigjutni, minél kevesebb hibával. A pontokat úgy számolják, hogy `100` -ból kivonják az idő felét (alsó egészrész) és a hibapontokat. Egy listában meg van adva a versenyzők neve, ideje és hibapontjai egy rendezett hármasban. Adjuk meg ki hány pontot ért el.

A listában ne szerepeljen az, aki:

- `100` hibapontot kapott, mert ez azt jelenti, hogy kizárásra került a versenyből,
- az összesítés után, `0` vagy negatív eredménye lenne.

```
points :: Integral a => [(String, a, a)] -> [(String, a)]
```

```
points [("Tomi",68,2),("Kati",75,10),("Imre",84,0)] == [("Tomi",64),("Kati",53),("Imre",58)]
points [("Laci",52,100)] == []
points [("Laci",52,87)] == []
points [("Samu",57,10),("Saci",52,6),("Geri",68,100)] == [("Samu",62),("Saci",68)]
points [("Tomi",68,100),("Kati",75,100),("Imre",84,100)] == []
take 10 (points (cycle [("Samu",57,10),("Saci",52,6),("Geri",68,100)])) == [("Samu",62),("Saci",68),("Samu",62),("Saci",68),("Samu",62),("Saci",68
```

Ryuk almái (2 pont)

Definiáljunk egy függvényt, amely megadja, hány almát tudtunk leszedni Ryuk-nak egy almás kertből. A kertben az összes almát egy `(Bool, Int)` rendezett pár reprezentálja, ezen elemek listája a fát, a fák listája pedig a kertet jelentik. A pár első komponense az alma érettségére vonatkozik (igaz, ha érett, különben hamis), második pedig azt adja meg, hogy az alma milyen magasan van a fán.

```
-- ne feledjük bemásolni a modulba
type Apple = (Bool, Int)
type Tree = [Apple]
type Garden = [Tree]
```

Amikor leszedjük az almákat két dolgot kell szem előtt tartanunk:

- Csak érett almákat szedünk le. (Az almát reprezentáló tuple első eleme `True`)
- Csak **3** méter magasságig érjük el az almákat, az annál magasabban lévőket nem tudjuk leszedni.

Megjegyzés: A kertről feltehető, hogy véges.

```
ryuksApples :: Garden -> Int
```

```
ryuksApples [] == 0
ryuksApples [[],[],[]] == 0
ryuksApples [[(True,3)]] == 1
ryuksApples [[(True,3), (False, 2)], [(True, 4)]] == 1
ryuksApples [[(True,6), (False, 2)], [(True, 4)]] == 0
ryuksApples [[(True,1),(True,2)],[(True,3)],[]] == 3
ryuksApples [[(True,1),(True,2),(True,4)],[(True,3)],[(True,4),(True,0)]] == 4
ryuksApples [[(False,1),(True,2),(False,4)],[(True,3)],[(True,4),(False,0)]] == 2
```

Szöveg a szövegben (2 pont)

Definiáljunk egy függvényt, amely két szöveget kap paraméterül és eldönti, hogy az első minden eleme megtalálható-e a másodikban. A tartalmazás alatt azt értjük, hogy az első szöveg minden eleme ugyan abban a sorrendben benne van a második szövegben. A karaktereknek nem kell közvetlen egymás mellett lenniük, azaz a karakterek közé egyéb karakterek is keveredhetnek a második szövegben.

Megjegyzés: Feltehetjük, hogy az első szöveg véges.

```
doesContain :: String -> String -> Bool
```

```
doesContain "" "" == True
doesContain "" "a" == True
doesContain "a" "" == False
doesContain "a" "a" == True
doesContain "a" "aa" == True
doesContain "aa" "aa" == True
doesContain "aa" " a  a" == True
doesContain "aa" " a  a " == True
doesContain "hero" "the quick brown fox jumps over the lazy dog" == True
doesContain "quick" "the quick brown fox jumps over the lazy dog" == True
doesContain "log" "the quick brown fox jumps over the lazy dog" == True
doesContain "elf" "the quick brown fox jumps over the lazy dog" == False
doesContain "alma" "_a_l_m_a_" == True
doesContain "alma" "a_l_m" == False
doesContain "alma" "a_l_a_m" == False
doesContain "alma" "wxalermmmma" == True
doesContain "szilva" (cycle "s f z l k j i l m m k v a j h") == True
```

Barbie (2 pont)

Barbie nagyon szereti a divatot és rengeteg különböző színű és stílusú szoknyája van. Barbie a szoknyákat egy listában tárolja, ahol minden szoknyát egy szó reprezentál, ami a ruhadarab színét adja meg. A tesztesetekben a szoknyák színeit magyarul, kisbetűkkel és ékezetek nélkül írtuk meg. Például: rozsaszin, fekete. Barbie a páros sorszámú szoknyáit szereti a legjobban, de nagyon szereti a rózsaszíneket is. A feketéket viszont ki nem állhatja. Ha nem talál kedvére való szoknyát, akkor pedig farmert vesz fel.

Definiáljunk egy függvényt, amely segít Barbie-nak megtalálni az első páros indexű, nem fekete szoknyát, vagy az első rózsaszín szoknyát. A függvény a választott szoknya színével térjen vissza. Amennyiben nincs a feltételnek megfelelő szoknya, úgy Barbie kénytelen farmert felvenni.

Megjegyzés: A lista indexelését `1` -től kezdjük.

```
barbie :: [String] -> String
```

```
barbie [] == "farmer"
barbie ["zold"] == "farmer"
barbie ["fekete"] == "farmer"
barbie ["rozsaszin"] == "rozsaszin"
barbie ["rozsaszin", "fekete"] == "rozsaszin"
barbie ["fekete", "rozsaszin"] == "rozsaszin"
barbie ["rozsaszin", "feher"] == "rozsaszin"
barbie ["kek", "fekete"] == "farmer"
barbie ["kek", "feher"] == "feher"
barbie ["kek", "fekete", "piros", "zold"] == "zold"
barbie ["kek", "fekete", "piros", "fekete", "rozsaszin"] == "rozsaszin"
barbie ["kek", "fekete", "piros", "fekete", "sarga", "zold", "rozsaszin"] == "zold"
barbie ["kek", "fekete", "piros", "fekete"] == "farmer"
barbie (cycle ["kek", "fekete", "fekete"]) == "kek"
```

Első teljesülő predikátum (2 pont)

Definiáljuk a `firstValid` függvényt, amely egy listányi predikátumot és egy, a függvényeknek megfelelő típusú értéket kap paraméterül. Adjuk vissza az első igazat adó predikátumnak az indexét!

Megjegyzés: Az indexelést `0` -tól kezdjük.

```
firstValid :: [a -> Bool] -> a -> Maybe Int
```

```
firstValid [(>2), (<3)] 1 == Just 1
firstValid [(>2), (<1)] 2 == Nothing
firstValid [(>1)] 0 == Nothing
firstValid ((>3) : repeat (const False)) 4 == Just 0
firstValid ((<3) : (>4) : (==4) : repeat (const False)) 4 == Just 2
```

Alkalmazott szűrés (2 pont)

Írj egy olyan magasabbrendű függvényt, amely egy predikátumot, egy függvényt, és két listát kap paraméterül. Ha a predikátum igazat ad a két lista ugyan azon indexű elemére, abban az esetben alkalmazzuk a két elemre a függvényünket.

```
combineListsIf :: (a -> b -> Bool) -> (a -> b -> c) -> [a] -> [b] -> [c]
```

```
combineListsIf undefined undefined [1, 2, 3] [] == []
combineListsIf (\x y -> y > x) (\x y -> x) [1, 2, 3] [4, 5, 6] == [1, 2, 3]
combineListsIf (\x y -> x `elem` y) (\x y -> x : y) ['a','b','c'] ["alma", "bálna", "terasz"] == ["aalma", "bbálna"]
take 15 (combineListsIf (\x y -> y > x) (\x y -> y) [1..] [4..]) == [4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]
```

Tömegközlekedés (3 pont)

Algebrai adatszerkezet

Egy város tömegközlekedését buszok és villamosok szolgálják ki. Sok járat van, minden járat sok, de véges számú megállóval rendelkezik, illetve egy megállóban akár több járat is megállhat. Definiáld a `Line` algebrai adattípust, amellyel a járatokat adja meg. A típusnak két konstruktora legyen, melyek az alábbiak:

- `Tram :: Integer -> [String] -> Line`
- `Bus :: Integer -> [String] -> Line`

Mindkét tömegközlekedési eszköz esetében az `Integer` a járatszámot, a `[String]` pedig a járat megállóinak a neveit tartalmazza. Kérjük meg a fordítót, hogy példányosítsa az `Eq` és `Show` típusosztályokat az új típusunkra.

Melyik busz áll meg az adott megállóban

Definiáld a `whichBusStop` függvényt, amely megadja hogy a tömegközlekedési hálózat mely **busz**járata(i) áll(nak) meg egy adott megállóban!

```
whichBusStop :: String -> [Line] -> [Integer]
```

```
whichBusStop "Alma utca" [] == []
whichBusStop "József utca" [Bus 111 ["Alma utca", "Károly utca", "József út", "Halom utca", "Mária utca"], Tram 10 ["József út", "Karinthy utca",
whichBusStop "Károly utca" [Tram 20 ["Alma utca", "Károly utca", "József út", "Halom utca", "Mária utca"], Tram 10 ["József út", "Karinthy utca",
whichBusStop "Mária utca" [Bus 111 ["Alma utca", "Károly utca", "József út", "Halom utca", "Mária utca"], Bus 10 ["József út", "Karinthy utca", "R
take 20 (whichBusStop "Nemes út" (Bus 5 ["József út", "Karinthy utca", "Róbert utca", "Templom tér", "Nemes út", "Őrház"] : Bus 10 ["József út", "
```

Színházi helyfoglalás (3 pont)

A barátainkkal színházba szeretnénk menni egy este. A feladat az, hogy meghatározzuk, van-e elegendő egymás melletti szabad hely egy adott sorban a társaságunk számára. Definiáljunk a `isReservable` függvényt, amely megnézi egy sorban, hogy van-e elegendő egymás melletti szabad hely hogy mindenki elférjen. A függvény paraméterül kapja azt, hogy hány székre lenne szükség folytonosan egy nemnegatív egész értékként. A második paraméterül kapott szövegben az `x` reprezentálja a szabad, `o` a foglalt helyet.

Segítség: Használjunk segédfüggvényt, esetleg nézzük meg a `Data.List` modul `isPrefixOf` függvényét.

Megjegyzés: Feltehetjük, hogy a String-ben csak `'x'` és `'o'` karakterek szerepelnek.

```
isReservable :: Int -> String -> Bool
```

```
isReservable 0 "" == True
isReservable 4 "" == False
isReservable 3 "oxxxx" == True
isReservable 2 "xxxxo" == True
isReservable 4 "ooooxxxo" == True
isReservable 0 "ooooxoo" == True
isReservable 2 "ooooxoo" == True
isReservable 3 "oooxoo" == False
isReservable 1 "oooxoo" == True
isReservable 4 "oooxoo" == False
isReservable 7 "oooxoxxxoooxxxxxxxxxo" == True
isReservable 7 (cycle "oooxoxxxoooxxxxxxxxxo") == True -- biztosan van benne
isReservable 7 (cycle "ooxoooxooooooxoooooxxxxxxxxxooxoxxxooooxxxxxxxxxo") == True -- biztosan van benne
```

Feltöltés

Tallózás

Feltöltés

Megoldás

Név:
solution.zip
Feltöltés ideje:
5/21/2025, 3:56 PM
Értékelés:
Státusz:
Sikertelen tesztelés
Feltöltések száma:
1
Értékelte:
Megjegyzések:

Automatikus tesztelés eredményei

#1

```
Elért pontszám: 18/18 pont.

Teszteken sikeresen átmenő definíciók: points, ryuksApples, doesContain, barbie, firstValid, combineListsIf, whichBusStop,
isReservable.

Minden teszt sikeresen lefutott.
```