Csoportok

5. házi feladat

Kategória:
Házi feladatok
Végső határidő:
3/23/2025, 11:59 PM (Beadva)
Próbálkozások száma:
Korlátlan
Kiírta:
Erdei Zsófia

Leírás:

Mintaillesztés, egyszerű rekurzió

0. Modul

Definiálj egy modult `Hazi5` néven!

1. Hegy

Hozzunk létre egy félhegyet `mountain` néven, amit jobbról lehet megmászni bal felfelé. A hegynek a szélessége `1` -től `n` -ig `1` -esével nő fentről lefelé. A hegynek egy `String` -nek kell lennie, új sorokkal elválasztva az egyes szélességeket. Új sort a `'\n'` karakterrel lehet írni. A hegy álljon `'#'` karakterekből. Az egyszerűség kedvéért lehet úgy is csinálni, hogy van `'\n'` a `String` végén, az ügyesebbeknek pedig lehet úgy is, hogy szép legyen és nincs `'\n'` a `String` végén. A megoldásban elágazás nem használható! A megoldásban használj rekurziót! Ne használj listagenerátort!

A paraméterül kapott számról feltehető, hogy nemnegatív. Jelenlegi eszközökkel még nem lehet lekezelni a negatív számokat.

```
mountain :: Integral i => i -> String
```

```
-- példa
mountain 5 == "#\n##\n###\n####\n#####\n" || mountain 5 == "#\n##\n###\n####\n#####"
-- konzolra kiírni a putStrLn függvénnyel lehet, ekkor a mountain 5 példára a következőt kell látni:
#
##
###
####
#####
```

2. 'a' karakterek egy szövegben

Definiáld a `countAChars` nevű függvényt, amely megszámolja egy szövegben található `'a'` karaktereket. Használj rekurziót! Ne használj listagenerátort, egyenlőség vizsgálatot és elágazást, használj **mintaillesztést**!

A szövegről feltehető, hogy véges!

```
countAChars :: Num i => String -> i
```

3. Lucas-sorozat

Számítsd ki a Lucas-sorozat `n.` elemét rekurzívan! A sorozat nulladik eleme a `2` , első az `1` , az `n.` eleme pedig az előző kettőnek az összege (mint a Fibonacci-sorozatban). Feltehető, hogy nem kap a függvény negatív számot paraméterül.

Figyelem: Egy adott definíció esetén a `30.` elemet már nagyon feltűnően nagyon lassan fogja kiszámolni, ezért azt javaslom, hogy arrafelé ne tesztelgessetek. Létezik annál jobb definíció is, de azt a mostani eszközökkel még nem lehet megcsinálni.

```
lucas :: (Integral a, Num b) => a -> b
```

4. Hasznos függvény a jövőben

Definiáld a `longerThan` függvényt, amely megmondja, hogy listának több eleme van-e, mint egy megadott szám, és amely kikerüli a `length lista > n` defektjét végtelen listákon. Ugye a lista ha végtelen, akkor sosem kapjuk vissza azt, hogy `True` annak ellenére, hogy a végtelen lista elemszáma nyilván nagyobb bármilyen egész számnál. Úgy szeretnénk megírni ezt a függvényt, hogy ilyen esetben is válaszoljon.

Használj rekurziót és mintaillesztést! Ne használj elágazást, illetve a `length` függvényt se!

Minden bemenetre tudnia kell válaszolni! (Tehát üres lista, végtelen lista, pozitív szám, negatív szám, 0 és ezek bármilyen típushelyes kombinációjára.)

```
longerThan :: Integral i => [a] -> i -> Bool
```

5. Kiegészítés

Definiáld a `format` függvényt, amely kiegészít szóközökkel egy szöveget megadott szélességűre (ami egy tetszőleges egész szám)! Ha netán a megadott szélesség kisebb lenne, mint a kapott szöveg, akkor ne vágj le a szövegből! Negatív szélességekkel, kiegészítésekkel nem kell foglalkozni, feltehető, hogy nem lesz olyan.

Használj mintaillesztést és rekurziót! Ne használj egyenlőségvizsgálatot és elágazást, magasabb rendű függvényeket, továbbá ne használd a `replicate` és `genericReplicate` függvényeket sem!

Emlékeztető: Végtelen `String` -re is kell megoldást adnia.

```
format :: Integral i => i -> String -> String
```

Segítség: A megoldáshoz bőven elég csak a `(:)` függvény, mintaillesztés és rekurzió megfelelő használata.

6. Összefésülés

Definiáld a `merge` függvényt, amely két listát összefésül. Az első elem az első lista első eleme legyen, a második elem a második lista első eleme, majd így váltakozva következzenek az elemek. Ha az egyik listából elfogynak az elemek, akkor a másik lista maradék elemét fűzd az eredmény végéhez.

```
merge :: [a] -> [a] -> [a]
```

Tesztek

```
mountain 3 == "#\n##\n###\n" || mountain 3 == "#\n##\n###"
mountain 0 == ""
mountain 40 == "#\n##\n###\n####\n#####\n######\n#######\n########\n#########\n##########\n###########\n############\n#############\n##############
countAChars "alma" == 2
countAChars "Én elmentem a vásárba a fél pénzemmel." == 3
countAChars "Ebben nincs." == 0
countAChars "" == 0
countAChars "A nagy A betű nem számít." == 1
lucas (0 :: Int) == (2 :: Double)
lucas 1 == (1 :: Float)
lucas (5 :: Integer) == (11 :: Int)
lucas 7 == 29
lucas 8 == 47
lucas (20 :: Int) == (15127 :: Integer)
[1..] `longerThan` 1000
not ([] `longerThan` 0)
not (replicate 50 'a' `longerThan` 51)
[1..50] `longerThan` 25
[] `longerThan` (-10)
[0,2..] `longerThan` (-1)
[0,2..] `longerThan` (-100)
[1] `longerThan` (-1)
[[1,2],[4,9,10],[1,0]] `longerThan` (-3)
format (10 :: Int) "alma" == "alma      "
format (4 :: Integer) "" == "    "
format (5 :: Integer) "szilva" == "szilva"
format (0 :: Int) "" == ""
format (0 :: Integer) "barack" == "barack"
take 61 (format (60 :: Integer) (repeat 'a')) == replicate 61 'a'
merge [1,3,5,7,9] [2,4,6,8] == [1..9]
merge [] [1,10,12] == [1,10,12]
merge [1,10,12] [] == [1,10,12]
null (merge [] [])
merge [1,2,3,4,5] [6,7] == [1,6,2,7,3,4,5]
merge "ab" "cdefgh" == "acbdefgh"
take 25 (merge "szilvafa" (repeat 'b')) == "sbzbiblbvbabfbabbbbbbbbbb"
```

Git tároló

Útvonal:

https://tms.inf.elte.hu/git/8208/vl2l98/w479f59cb39b91327d4a818f85

Használat:

git clone https://tms.inf.elte.hu/git/8208/vl2l98/w479f59cb39b91327d4a818f85

Megoldás

Név:
solution.zip
Feltöltés ideje:
3/22/2025, 7:08 PM
Értékelés:
Státusz:
Elfogadva
Feltöltések száma:
4
Értékelte:
Szávó Tamás
Megjegyzések:

Automatikus tesztelés eredményei

#1