Infix kiértékelő

Ha ez a házi megvan kb. 40-45 percen belül, akkor a 4. zh megoldásával semmilyen fajta probléma nem lehet!

0. Modul

Definiálj egy `Hazi9` nevű modult!

1. Matematikai kifejezések

1.1. Saját típus

Definiáld az `Infix` típust, amellyel matematikai kifejezéseket lehet jelölni!

Az alábbi konstruktorokra van szükség:

- `Value :: a -> Infix a`, ez fog egy konstans értéket jelölni, tárolni.
- `Operation :: (a -> a -> a) -> Infix a`, ez tartalmazza a műveletet, amelyet két értéken lehet elvégezni. Tegyük fel, hogy az összes ilyen művelet balra köt. Ennek a kiértékelésnél lesz jelentőssége.
- `OpeningBracket :: Infix a`, ez jelöli a nyitó zárójelet.
- `ClosingBracket :: Infix a`, ez jelöli a csukó zárójelet.

Megjegyzés: Észrevehetitek, hogy semmilyen `deriving` nem fog működni rajta, próbáljátok ki nyugodtan. Saját módon `instance`-ok készíthetőek rájuk, de ellenőrizni fogom, hogy totálisak-e az azokban megadott függvények. (A feladatok megoldásának szempontjából egyébként teljesen felesleges bármilyen `instance` rá.)

1.2. Kifejezés

Definiálj egy típusszinonímát `InfixExpression` névvel, amely csak az `[Infix a]` típust rövidíti.

1.3. Helyesen zárójelezett kifejezés

Például egy `InfixExpression Int` típusú értéket a következőképpen kell elképzelni:

- `[Value 1, Operation (+), Value 2] ==> 1 + 2`
- `[OpeningBracket, Value 1, Operation (*), Value 4, ClosingBracket, Operation (+), Value 6] ==> (1 * 4) + 6`
- `[Value 1, Operation (+), OpeningBracket, OpeningBracket, Value 3, Operation (^), Value 2, ClosingBracket, Operation (^), Value 4, ClosingBracket] ==> 1 + ((3 ^ 2) ^ 4)`
- `[Value 1, Operation (+), OpeningBracket, OpeningBracket, Value 3, Operation (^), Value 2, ClosingBracket, Operation (^), Value 4] ==> 1 + ((3 ^ 2) ^ 4 -- helytelenül zárójelezett kifejezés`

Definiáld az `isCorrectlyParenthesised :: InfixExpression a -> Bool` függvényt, amely ellenőrzi, hogy minden `OpeningBracket`-hez tartozik-e annak megfelelő `ClosingBracket`, azaz a kifejezés helyesen zárójelezett-e. Akkor lesz helyesen zárójelezett egy kifejezés, ha minden nyitó zárójelnek megvan a záró zárójel párja, amely a nyitó **után** szerepel.

Az nem érdekes, hogy esetleg magának a felírt kifejezésnek nincs feltétlen jelentése, annak ellenőrzése nem ennek a függvénynek a feladata.

1.4. A kifejezés kiértékelése

Definiáld az `evaluateInfixExpression :: InfixExpression a -> Maybe a` függvényt, amely kiértékeli a listaként elkódolt kifejezést. Minden művelet balra kötő, balról jobbra felé haladva kell elvégezni mindet. A zárójelekben lévő kifejezések elsőbbséget élveznek, azokat úgy együtt szükséges kiértékelni. Ha a kifejezés nem egy érvényes infix kifejezés, akkor az eredmény legyen `Nothing`. A kifejezésről feltehető, hogy nem végtelen, így a lista sem az (ettől függetlenül felesleges a `!!` és `length` függvényeket használni).

Sok segítség:

1. Hasonló a módszer, mint a zárójel ellenőrzésénél, csak több mindenre kell figyelni.
2. Az `1.1.`-ben meg volt adva, hogy az összes művelet **balra** köt, emiatt milyen konstrukciót érdemes használni?
3. Érdemes segédfüggvényt használni külön a zárójeles kifejezések kiszámítására. Egyszerűbb az ellenőrzést megint simán rekurzívan csinálni, mint külön ellenőrizni az `isCorrectlyParenthesised` függvénnyel, hiszen a zárójeleknek most egy kicsit több funkciójuk van, műveleti sorrendet határoznak meg.
4. A `case ... of` konstrukció tud segíteni a zárójeles kifejezések kiértékelésében a fő függvényben.
5. A zárójeles kifejezés kiértékeléséhez egyszerűbb a teljes maradék kifejezést átadni, majd visszaadni eredményként a még feldolgozandót, mintsem csak a záró zárójelet keresgélni.
6. Elég csak a szabályos esetekkel foglalkozni, a többit lekezelni a _ mintával és `Nothing`-ot visszaadni.
7. Mik a szabályos esetek? Mikor milyen érték, szimbólum után mi állhat?
8. Pl. mivel kezdődhet a kifejezés? Csak értékkel vagy nyitó zárójellel; ha bármi más, az gyanús.

Tesztek

```
isCorrectlyParenthesised []
isCorrectlyParenthesised [Value 1, Operation (+), Value 2]
isCorrectlyParenthesised [OpeningBracket, Value 1, Operation (*), Value 4, ClosingBracket, Operation (+),
isCorrectlyParenthesised [Value 1, Operation (+), OpeningBracket, OpeningBracket, Value 3, Operation (^),
not (isCorrectlyParenthesised [Value 1, Operation (+), OpeningBracket, OpeningBracket, Value 3, Operation
not (isCorrectlyParenthesised [ClosingBracket, OpeningBracket])
isCorrectlyParenthesised [OpeningBracket, ClosingBracket]
isCorrectlyParenthesised [Value 2, OpeningBracket, Value 3, OpeningBracket, Operation mod, Operation div,
not (isCorrectlyParenthesised [Operation (*), OpeningBracket, Value (-10), OpeningBracket, ClosingBracket
not (isCorrectlyParenthesised [Operation (*), OpeningBracket, Value (-10), OpeningBracket, ClosingBracket
not (isCorrectlyParenthesised [OpeningBracket])
not (isCorrectlyParenthesised [ClosingBracket])
case evaluateInfixExpression [] of Nothing -> True; _ -> False
case evaluateInfixExpression [Operation (+), Value 10, Value 15] of Nothing -> True; _ -> False
case evaluateInfixExpression [Value 10, Value 20, Operation (*), Value 3] of Nothing -> True; _ -> False
case evaluateInfixExpression [OpeningBracket] of Nothing -> True; _ -> False
case evaluateInfixExpression [ClosingBracket] of Nothing -> True; _ -> False
case evaluateInfixExpression [OpeningBracket,ClosingBracket] of Nothing -> True; _ -> False
case evaluateInfixExpression [Value 10, Operation (*), Value 20, Operation (+), OpeningBracket, ClosingBr
case evaluateInfixExpression [Value 10, Operation (*), Value 20, Operation (+), OpeningBracket, Value 10,
case evaluateInfixExpression [Value 10, Operation (*), Value 20, Operation (+), OpeningBracket, Value 10,
case evaluateInfixExpression [Value 10, Operation (*), Value 20, Operation (+), OpeningBracket, Value 10,
case evaluateInfixExpression [Value "alma", Operation (++), Value "szilva", Operation (zipWith min), Valu
case evaluateInfixExpression [Value 10, Operation (+), Value 20, Operation (*), OpeningBracket, Value 10,
case evaluateInfixExpression [OpeningBracket, OpeningBracket, Value 10, Operation (*), OpeningBracket, Va
case evaluateInfixExpression [Value 5, Operation (+), OpeningBracket, Value 10, Operation (+), OpeningBra
```