Gequoteerde zitting Haskell: My Own Programming Language

NAAM: RICHTING:

Enkele praktische afspraken

- Je krijgt twee uur om deze opdracht **individueel** op te lossen.
- Je raadpleegt enkel de Haskell slides (eventueel in afgedrukte vorm met handgeschreven nota's) en oefening die via Toledo voor dit vak ter beschikking gesteld zijn. Je mag ook de manuals vermeld op Toledo en eventueel Hoogle raadplegen.
- In de map **1415_Gequoteerde/Haskell_donderdag** op Toledo vind je de bestanden `MOPL.hs` en `MOPLtest.hs`. Ook de indienmodule staat daar.
  - **Download en open** het bestand `MOPL.hs`, hierin staat reeds een template voor je oplossing.
  - Als eerste vul je bovenaan je naam, studentennummer en richting in.

    ```
    -- Jan Jansen
    -- r0123456
    -- master cw
    ```

  - Bovenaan in het bestand worden reeds een aantal functies geïmporteerd die je waarschijnlijk nodig zal hebben. Zoek hun werking op indien je ze nog niet kent. Je mag ook **extra functies en types importeren**!
  - Voor elke opdracht zijn een aantal functies reeds gedefinieerd met een bijbehorende typesignatuur. Deze typesignatuur mag niet gewijzigd worden. Vervang telkens `undefined` met jouw implementatie. Je mag argumenten voor het gelijkheidsteken zetten, maar de meeste van de functies kunnen *point-free* (zonder het expliciet benoemen van de argumenten) geschreven worden. Het is natuurlijk ook altijd toegestaan om extra (hulp)functies te schrijven.
  - Je kan je oplossing testen m.b.v. `MOPLtest.hs`. Dit doe je door in de map waarin de twee `.hs` bestanden staan het volgende commando uit te voeren:

    ```
    runhaskell MOPLtest.hs
    ```

    N.B. dat alle testen slagen betekent niet per se dat je programma helemaal correct is of dat je het maximum van de punten verdient.
  - Na twee uur of wanneer je klaar bent, dien je het bestand `MOPL.hs` bestand in via Toledo.

Oefening

In deze oefening zullen we een nieuwe programmeertaal ontwerpen: My Own Programming Language (MOPL). De programmeertaal is heel simpel en er zijn slechts twee soorten statements beschikbaar, toekenningen van variabelen, en print statements. We zullen deze taal eerst uitwerken voor assignments, en daarna uitbreiden met print statements.

We stellen een programma voor als een lijst van statements. Dit is een voorbeeld van een geldig programma:

```
program :: [Statement]
program = [
    assign  "a" (intTerm 8), --a = 8
    printTerm (plus (varTerm "a") (intTerm (-5))), --print(a-5)
    assign "b" (plus (varTerm "a") (intTerm 2)), --b = a+2
    assign "a" (plus (varTerm "a") (varTerm "b")), -- a = a+b
    printTerm (varTerm "a") -- print(a)
    ]
```

Als dit programma uitgevoerd wordt, wordt de variabele `a` eerst gelijkgesteld aan 8, vervolgens wordt `a-5` geprint, dus het programma print `3`. Daarna wordt `b` gelijkgesteld aan `a+2`, dus `b` wordt 10, daarna wordt `a` gelijkgesteld aan `a+b`, dus `a` wordt 18, en daarna wordt `a` geprint, dus het programma print `18`.

```
Main> execute program
3
18
```

We bouwen de programmeertaal stap per stap op. Laat je niet ontmoedigen door het grote aantal opdrachten, het zijn allemaal opdrachten die op enkele lijntjes kunnen opgelost worden.

Opdracht 1: Datatypes voor MOPL
Schrijf de datatypes `Term` en `Statement`. Een statement is een assignment van een variable aan een term, of een printstatement van een term. Een term is ofwel een variabele (voorgesteld door een String), een integer, ofwel een toepassing van een een binaire operatie op 2 termen. In deze taal ondersteunen we optelling, aftrekking en vermenigvuldiging, maar zorg dat je elke binaire functie over de integers kunt representeren zonder je datatype aan te passen.

Opdracht 2: Hulpfuncties voor MOPL
Om makkelijk te kunnen werken met de datatypes van opdracht 1, is het goed om functies te maken zodat we makkelijk programma's kunnen schrijven. Implementeer volgende functies[1]:

```
assign :: String -> Term -> Statement
printTerm :: Term -> Statement
intTerm :: Int -> Term
varTerm :: String -> Term
plus :: Term -> Term -> Term
times :: Term -> Term -> Term
minus :: Term -> Term -> Term
```

Opdracht 3: De state van een MOPL-programma
Een programma dat uitgevoerd wordt, heeft op ieder ogenblik een bepaalde state, namelijk de binding tussen de variabelen, en hun waarde. We zullen ene state voorstellen door middel van een lijst van koppels `[(String,Int)]`.

- Schrijf een functie `valueOf :: [(String,Int)] -> String -> Int`, die voor een gegeven String, de eerste integer uit de lijst haalt die gekoppeld is met die string. Je mag er vanuit gaan dat er altijd zo een koppel bestaat.

  Enkele voorbeelden:

[1] Hint: bij goedgekozen datatypes zou de implementatie van elk van deze functies minder dan 20 tekens moeten bevatten.

```
Main> valueOf [("a",5),("b",9),("c",10)] "b"
9
Main> valueOf [("a",1),("a",2),("a",3)] "a"
1
```

- Schrijf een functie `insertS :: String -> Int -> [(String,Int)] -> [(String,Int)]`, die een tupel toevoegt aan/vervangt in de state. Het nieuwe koppel wordt sowieso toegevoegd aan de state, en als er al koppels aanwezig zijn met dezelfde string, dan worden deze verwijderd. De volgorde van de koppels in de resulterende lijst is niet van belang.

  Enkele voorbeelden:

```
Main> insertS "a" 1 []
[("a",1)]
Main> insertS "b" 3 [("a",5),("b",9),("c",10)]
[("a",5),("b",3),("c",10)]
```

Opdracht 4: De evaluatie van termen
Als opstapje naar het uitvoeren van een programma, moeten we eerst termen kunnen evalueren aan de hand van een state. Schrijf de functie `evalTerm :: [(String,Int)] -> Term -> Int`, die de waarde van een term berekent aan de hand van de huidige state.

```
Main> evalTerm [] (intTerm 5)
5
Main> evalTerm [("a",6)] (varTerm "a")
6
Main> evalTerm [("a",6)] (times (varTerm "a") (intTerm 2))
12
```

Opdracht 5: De uitvoering van een assignment
Schrijf de functie `execAssign :: String -> Term -> [(String,Int)] -> [(String,Int)]`, die de state van een programma voor een statement, omzet naar de state van een programma na het statement.

```
Main> execAssign "b" (intTerm 6) []
[("b",6)]
Main> execAssign "b" (minus (varTerm "b") (varTerm "a") ) [("a",3),("b",8)]
[("a",3),("b",5)]
```

Opdracht 6: De uitvoering van lijst van assignments
Schrijf de functie `execPure :: [(String,Int)] -> [Statement] -> [(String,Int)]`, die de state van een programma voor een lijst van statements, omzet naar de state van een programma na deze statements, door al deze statements na elkaar uit te voeren. De print statements hebben geen effect op de state, dus deze kun je volledig negeren.

Opdracht 7: De volledige taal
Schrijf de functie `execute :: [Statement] -> IO ()` die een programma volledig uitvoert zoals in de introductie van de opgave, startende vanaf de lege state.