Gequoteerde zitting Haskell: Personeelsmanager

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 oefeningen die via Toledo voor dit vak ter beschikking gesteld zijn. Je mag ook Hoogle en Hackage raadplegen. Ten slotte kan je E-Systant gebruiken om je oplossing te testen.
- In de map **1718_Gequoteerde/Haskell_woensdag** op Toledo vind je de bestanden `Personnel.hs` en `PersonnelTests.hs`. Ook de indienmodule staat daar.
  - **Download en open** het bestand `Personnel.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 `Personnel.hs` worden reeds een aantal modules 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 `error "not implemented"` met jouw implementatie. Het is natuurlijk ook altijd toegestaan om extra (hulp)functies te schrijven.
  - Je kan je oplossing testen m.b.v. `PersonnelTests.hs`. Dit doe je door in de map waarin de twee `.hs` bestanden staan het volgende commando uit te voeren:

    ```
    runhaskell PersonnelTests.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 `Personnel.hs` in via Toledo. Je mag E-Systant gebruiken om je oplossing te testen, maar je moet uiteindelijk via Toledo indienen. Enkel de Toledo submissie wordt beoordeeld.

Personeelsmanager

Tom is de CEO van een belangrijke multinational. Een van zijn aandachtspunten bij het runnen van het bedrijf is dat het aannemen van nieuw personeel volledig objectief moet verlopen. Hij vraagt jou dan ook om een automatische personeelsmanager te ontwikkelen. Deze applicatie moet het huidige personeelsbestand bijhouden en beslissen welke nieuwe mensen aangenomen worden.

Het bedrijf bestaat uit 3 departementen: 1. Het ICT departement, voor de programmeurs van het bedrijf. 2. Het HR departement, dat instaat voor conflicten tussen personeelsleden. 3. Het Cleaning departement, dat ervoor zorgt dat het gebouw altijd in perfecte staat is.

Opdracht 1: Personeel

Opdracht 1a: Schrijf een ADT `Employee` om een werknemer voor te stellen. Je dient onderscheid te maken tussen 3 verschillende soorten werknemers, overeenkomend met de 3 verschillende departementen. Elke werknemer moet een uniek ID nummer toegekend krijgen (`Int`), een naam hebben (`String`) en een maandelijks loon (`Int`). Verder moeten een aantal extra attributen bijgehouden worden, afhankelijk van het departement waarin de werknemer werkt.

Deze departementen beoordelen hun personeel op verschillende punten. De personeelsmanager moet dan ook een onderscheid maken tussen hun werknemers. Voor programmeurs en HR werknemers moet het aantal werkuren van de voorbije week bijgehouden worden (`Int`). Bovendien moet voor HR werknemers ook het aantal opgeloste conflicten (`Int`) opgeslagen worden. Ten slotte is Tom niet geïntereseerd in het aantal werkuren van het Cleaning personeel, maar enkel in het aantal lokalen dat ze onderhouden (`Int`).
Hint: Gebruik voor deze opdracht 1 datatype, met meerdere constructors. Code duplicatie bij het schrijven van dit datatype is geen probleem.

Opdracht 1b: Schrijf een instance voor het `Employee` data type voor de `Show` klasse, zodat personeelsleden op de volgende manier geprint kunnen worden (afhankelijk van hun departement):
Programmeur:

```
1 --- Tom --- 2000 euro/month --- 50 hours
```

HR:

```
1 --- Tom --- 2000 euro/month --- 50 hours --- 2 conflicts
```

Cleaning:

```
1 --- Tom --- 2000 euro/month --- 5 rooms
```

Hint: Wanneer je de `Show` instantie voor `Employee` geschreven hebt, dien je de `deriving (Show)` weg te halen bij je definitie van `Employee`.

Opdracht 1c: Schrijf een functie `createEmployee :: EmployeeInput -> Int -> Employee`, die een nieuwe `Employee` construeert. De functie krijgt een input (`EmployeeInput`, wat bestaat uit een tuple met een naam (`String`), een departement (`Department`) en een maandelijks loon (`Int`)) en een nieuw ID nummer (`Int`). Nieuwe werknemers starten uiteraard met 0 gewerkte uren, 0 opgeloste conflicten en 0 toegewezen lokalen om te onderhouden.

Merk op: Het `EmployeeInput` type en het `Department` data type zijn al gedefinieerd in de template en hoef je dus niet meer zelf te schrijven.

Opdracht 1d: Schrijf een functie `createInitEmployees :: [EmployeeInput] -> EmployeeRecord`, die gegeven een lijst van input tuples (`EmployeeInput`) een `EmployeeRecord` construeert. `EmployeeRecord` is al voor gedefinieerd in de template en bestaat uit een lijst van `Employee`s. De ID nummers beginnen op 1 voor de eerste werknemer in `[EmployeeInput]` en lopen dan verder op.

```
Main> createInitEmployees example_employees
[1 --- Tony --- 5000 euro/month --- 0 hours
,2 --- Bruce --- 2000 euro/month --- 0 hours
,3 --- Nick --- 2000 euro/month --- 0 hours --- 0 conflicts
,4 --- Phil --- 1500 euro/month --- 0 hours --- 0 conflicts
,5 --- Steve --- 1500 euro/month --- 0 rooms]
```

Merk op: Om de leesbaarheid te verbeteren is het voorbeeld hierboven manueel over meerdere lijnen opgedeeld.

Opdracht 2: Aanwerven

Tom heeft voor elk departement een aantal gewenste kwaliteiten uitgeschreven voor nieuwe werknemers. Aan alle nieuwe kandidaatwerknemers is gevraagd om hun belangrijkste kwaliteiten op te sommen. De applicatie neemt dan de kandidaat aan die aan het grootste aantal van de gewenste kwaliteiten voor het gegeven departement voldoet. Indien twee kandidaten aan evenveel kwaliteiten voldoen, wordt de kandidaat met het laagste gevraagde loon aangenomen.

Opdracht 2a: Schrijf een functie `getMatchingPercentage :: Department -> [Requirement] -> Candidate -> Int`, die voor een gegeven departement (`Department`), lijst van requirements (`Requirement`) en een kandidaat (`Candidate`) berekent aan hoeveel procent van de gewenste kwaliteiten de kandidaat voldoet.

Zowel `Requirement` als `Candidate` zijn voorgedefinieerd. `Requirement` is gedefinieerd als een tuple `(Department, String)`, waarin het tweede argument een gewenste kwaliteit is voor het gegeven departement. `Candidate` is gedefinieerd als een tuple `(String, [String], Int)`, waarbij het eerste argument de naam is van de kandidaat, het tweede een lijst van skills en het laatste argument het gevraagde maandelijkse loon.

Het resultaat dient altijd naar beneden afgerond te worden naar een `Int` waarde.

```
Main> example_requirements
[(ICT,"Haskell")
```

```
,(ICT,"Prolog")
,(ICT,"Git")
,(HR,"PeopleSkills")
,(HR,"Connections")
,(Cleaning,"Experience")
,(Cleaning,"Motivation")]

Main> getMatchingPercentage ICT example_requirements
      ("Peter", ["Haskell", "Git", "Motivation"], 1000)
66

Main> getMatchingPercentage HR example_requirements
      ("Peter", ["Haskell", "Git", "Motivation"], 1000)
0

Main> getMatchingPercentage Cleaning example_requirements
      ("Peter", ["Haskell", "Git", "Motivation"], 1000)
50
```

Opdracht 2b: Schrijf een functie `sortCandidates :: Department -> [Requirement] -> [Candidate] -> [Candidate]`, die een gegeven lijst van kandidaten sorteert van goed naar slecht. Zoals eerder vermeld, komen de kandidaten die aan het meeste van de gewenste kwaliteiten voldoen vooraan. Indien twee kandidaten gelijk zijn, dient de kandidaat met het laagste gevraagde loon eerst te staan.

Hint: Je kan **sortBy** gebruiken en hoeft dus zeker geen eigen sorteer algoritme te implementeren.

```
Main> sortCandidates ICT example_requirements example_candidates
[("Peter",["Haskell","Git","Motivation"],1000)
,("MaryJane",["Prolog","Connections","Looks"],1500)
,("Ben",["Haskell","PeopleSkills","Connections","Experience","Wisdom"],5000)
,("May",["PeopleSkills","Experience","Motivation"],2000)
,("Harry",["Connections","Motivation","Money"],8000)]

Main> sortCandidates HR example_requirements example_candidates
[("Ben",["Haskell","PeopleSkills","Connections","Experience","Wisdom"],5000)
,("MaryJane",["Prolog","Connections","Looks"],1500)
,("May",["PeopleSkills","Experience","Motivation"],2000)
,("Harry",["Connections","Motivation","Money"],8000)
,("Peter",["Haskell","Git","Motivation"],1000)]

Main> sortCandidates Cleaning example_requirements example_candidates
[("May",["PeopleSkills","Experience","Motivation"],2000)
,("Peter",["Haskell","Git","Motivation"],1000)
```

```
,("Ben",["Haskell","PeopleSkills","Connections","Experience","Wisdom"],5000)
,("Harry",["Connections","Motivation","Money"],8000)
,("MaryJane",["Prolog","Connections","Looks"],1500)]
```

Merk op: Om de leesbaarheid te verbeteren is het voorbeeld hierboven manueel over meerdere lijnen opgedeeld.

Opdracht 2c: Schrijf een functie `hireCandidate :: Department -> Int -> [Requirement] -> [Candidate] -> Maybe Candidate` dat, gegeven een departement (`Departement`), een maximaal budget (`Int`), een lijst van requirements (`Requirement`), en een lijst kandidaten (`Candidate`), de beste kandidaat teruggeeft indien deze binnen het budget valt. Als de beste kandidaat een hoger maandelijks loon vraagt dan het budget toelaat, of de gegeven lijst van kandidaten leeg is, wordt er niemand aangenomen en geeft de functie `Nothing` terug.

```
Main> hireCandidate ICT 2000 example_requirements example_candidates
Just ("Peter",["Haskell","Git","Motivation"],1000)


Main> hireCandidate ICT 500 example_requirements example_candidates
Nothing
```

Opdracht 2d: Schrijf een functie `executeHire :: Departement -> Int -> [Requirement] -> [Candidate] -> EmployeeRecord -> EmployeeRecord` die de functie uit de vorige opgave gebruikt om het gegeven personeelsbestand (`EmployeeRecord`) te updaten met het nieuwe personeelslid. Indien `hireCandidate Nothing` teruggeeft, mag het gegeven personeelsbestand niet aangepast worden. Nieuwe personeelsleden dienen achteraan toegevoegd te worden in het personeelsbestand.

```
Main> executeHire ICT 2000 example_requirements example_candidates
      (createInitEmployees example_employees)
[1 --- Tony --- 5000 euro/month --- 0 hours
,2 --- Bruce --- 2000 euro/month --- 0 hours
,3 --- Nick --- 2000 euro/month --- 0 hours --- 0 conflicts
,4 --- Phil --- 1500 euro/month --- 0 hours --- 0 conflicts
,5 --- Steve --- 1500 euro/month --- 0 rooms
,6 --- Peter --- 1000 euro/month --- 0 hours]


Main> executeHire ICT 500 example_requirements example_candidates
      (createInitEmployees example_employees)
[1 --- Tony --- 5000 euro/month --- 0 hours
,2 --- Bruce --- 2000 euro/month --- 0 hours
,3 --- Nick --- 2000 euro/month --- 0 hours --- 0 conflicts
,4 --- Phil --- 1500 euro/month --- 0 hours --- 0 conflicts
,5 --- Steve --- 1500 euro/month --- 0 rooms]
```

Merk op: Om de leesbaarheid te verbeteren is het voorbeeld hierboven manueel over meerdere lijnen opgedeeld.

Opdracht 3: Interactieve Personeel Manager

Ten slotte vraagt Tom je om de applicatie interactief te maken. Hij geeft je dit voorbeeld over hoe hij de tool wil gebruiken:

```
Main> mainManager
Welcome to the Personnel Register
Enter the department where to hire:
 ICT
Enter the budget:
 2000
Resulting personnel register:
+-------------------------------------------------
| 1 --- Tony --- 5000 euro/month --- 0 hours
| 2 --- Bruce --- 2000 euro/month --- 0 hours
| 3 --- Nick --- 2000 euro/month --- 0 hours --- 0 conflicts
| 4 --- Phil --- 1500 euro/month --- 0 hours --- 0 conflicts
| 5 --- Steve --- 1500 euro/month --- 0 rooms
| 6 --- Peter --- 1000 euro/month --- 0 hours
+-------------------------------------------------
```

Opdracht 3a: Schrijf een functie `readDepartment :: String -> Maybe Department`, dat een `String` omzet naar een `Department`. De mogelijke strings zijn `"ICT"`, `"HR"` of `"Cleaning"`. Wanneer een andere string als input gegeven wordt, dient de functie `Nothing` terug te geven.

```
Main> readDepartment "ICT"
Just ICT

Main> readDepartment "Kitchen"
Nothing
```

Opdracht 3b: Schrijf een functie `prettyPrintEmployeeRecord :: EmployeeRecord -> IO ()`, dat een `EmployeeRecord` naar het scherm print in een kader, alsvolgt:

```
Main> prettyPrintEmployeeRecord $ createInitEmployees example_employees
+-------------------------------------------------
| 1 --- Tony --- 5000 euro/month --- 0 hours
| 2 --- Bruce --- 2000 euro/month --- 0 hours
| 3 --- Nick --- 2000 euro/month --- 0 hours --- 0 conflicts
| 4 --- Phil --- 1500 euro/month --- 0 hours --- 0 conflicts
```

```
| 5 --- Steve --- 1500 euro/month --- 0 rooms
+--------------------------------------------------
```

Wanneer het gegeven personeelsbestand leeg is, dient `EMPTY` geprint te worden:

```
Main> prettyPrintEmployeeRecord $ createInitEmployees []
+--------------------------------------------------
| EMPTY
+--------------------------------------------------
```

Hint: Gebruik hiervoor de `Show` instantie die je in de eerste opgave geschreven hebt. Je moet enkel nog de twee horizontale lijnen (een plus en 50 streepjes) en de verticale lijntjes aan het begin van de lijn toe te voegen.

Opdracht 3c: Schrijf de functie `mainBase :: [Requirement] -> [Candidate] -> EmployeeRecord -> IO ()`, die Tom's gegeven voorbeeld implementeert. Deze functie wordt aangeroepen met de juiste argumenten door de `mainManager :: IO ()` functie. De functie vraagt de gebruiker om een departement (`String`) en een budget. De functie gaat dan (indien het past binnen het budget) een nieuwe werknemer toevoegen. Ten slotte wordt het resulterende personeel bestand op het beeldscherm geoutput, m.b.v. de pretty printer die je in de vorige opgave gedefinieerd hebt.

Indien de gebruiker geen geldige string geeft wanneer je om het departement of het budget vraagt, print je een fout melding en dient de functie van vooraf aan te beginnen. Dit wordt weergegeven in het volgende voorbeeld:

```
Main> mainManager
Welcome to the Personnel Register
Enter the department where to hire:
Kitchen
Please try again

Welcome to the Personnel Register
Enter the department where to hire:
ICT
Enter the budget:
2000
Personnel register:
+------------------------------------------------
| 1 --- Tony --- 5000 euro/month --- 0 hours
| 2 --- Bruce --- 2000 euro/month --- 0 hours
| 3 --- Nick --- 2000 euro/month --- 0 hours --- 0 conflicts
| 4 --- Phil --- 1500 euro/month --- 0 hours --- 0 conflicts
| 5 --- Steve --- 1500 euro/month --- 0 rooms
| 6 --- Peter --- 1000 euro/month --- 0 hours
+------------------------------------------------
```

Hint: Je kan gebruik maken van `readMaybe :: String -> Maybe a` om het gegeven budget in te lezen.