Functioneel Programmeren – opgaven practicum 2

Opgave 1. Definieer de volgende functies die precies zo werken als hun broertjes in Haskell: `myfilter`, `myfoldl`, `myfoldr`, `myzipWith`. Geef ook hun types.

Opgave 2. Van personen zijn opgeslagen in een database: naam, leeftijd, geslacht, woonplaats. Neem aan dat personen uniek zijn bepaald door hun naam.

Deelopgaven f, g hoeven maar op één van deze manieren te worden gemaakt.

a. Geef het type van zo'n database, waarbij de gegevens van één persoon in een 4-tupel worden opgeslagen. Maak zelf een voorbeeld database van dit type om onderstaande functies te kunnen testen.

b. Schrijf functies om de afzonderlijke gegevens van één persoon uit een tupel te kunnen halen.

c. Schrijf op *drie* verschillende manieren (met recursie, met lijstcomprehensie, en met hogere orde functies) een functie om de leeftijd van alle personen met n jaar op te hogen.

d. Schrijf op dezelfde drie manieren een functie die de namen oplevert van alle vrouwen tussen 30 en 40 jaar.

e. Schrijf een functie (één manier is voldoende) die de leeftijd van iemand (gegeven door zijn/haar naam) oplevert. De naam moet met hoofd- en/of kleine letters door elkaar geschreven kunnen worden (de Haskell module `Data.Char` kent de functies `toLower`, `toUpper`).

f. Sorteer het bestand op leeftijd.
Haskell kent een standaardfunctie `sort` (in de module `Data.List`) die ook voor tupels werkt (ga na hoe). Schrijf een functie die een tupel "swapt", en gebruik vervolgens hogere orde functies en functiecompositie.

Opgave 3.

a. De *zeef van Eratosthenes* is een techniek om de lijst van alle priemgetallen te genereren. Hij werkt als volgt: begin met de lijst van alle

natuurlijke getallen vanaf 2. Neem het eerste getal en verwijder uit de rest van de lijst alle getallen die deelbaar zijn door dit eerste getal. Herhaal deze procedure voor het getal dat vervolgens het eerste is, etc.

Schrijf een functie `zeef` die de zeef van Erathostenes implementeert. Schrijf vervolgens enkele functies die de functie `zeef` gebruiken:

(1) een functie die test of een gegeven getal `n` een priemgetal is,

(2) een functie die de eerste `n` priemgetallen oplevert,

(3) een functie die alle priemgetallen kleiner dan `n` oplevert.

Opmerking: deze functies maken wezenlijk gebruik van *oneindige lijsten* en van *lazy evaluation*.

b. Definieer een functie `delers` die de lijst van alle delers van een gegeven getal m oplevert. Definieer met behulp van deze functie opnieuw een functie die bepaalt of een getal een priemgetal is.

Opgave 4. Een drietal gehele getallen (a, b, c) heet een *pythagoreïsch drietal*, als $a^2 + b^2 = c^2$. Bekende voorbeelden van dergelijke drietallen zijn (3,4,5) en (5,12,13).

a. Schrijf een funktie `pyth` die alle pythagoreïsche drietallen onder een zeker maximum `n` genereert. Ga na of uw functie ook zou werken als u dit maximum `n` weg zou laten uit de definitie.

b. Schrijf een variant `pyth'` van `pyth` die alleen de lijst van "wezenlijk verschillende" drietallen oplevert. Bijvoorbeeld, (3,4,5) en (4,3,5) zijn hetzelfde drietal. Ook (3,4,5) en (6,8,10) zijn niet wezenlijk verschillend.

Opgave 5.

a. Een lijst van getallen is *stijgend* als elk volgend getal in de lijst groter is dan het voorgaande getal. Schrijf een functie `stijgend`, die bepaalt of een lijst stijgend is of niet.

b. Een lijst is *zwak stijgend* als elk volgend getal groter is dan het gemiddelde van alle voorafgaande elementen in de lijst. Schrijf een functie `zwakStijgend` die bepaalt of een lijst zwak stijgend is of niet.

Opgave 6. Een lijst `xs` heet een *deellijst* van een lijst `ys` als de elementen van `xs` aaneengesloten voorkomen in `ys`, terwijl `xs` een *sublijst* van `ys` is als de elementen van `xs` in dezelfde volgorde (maar niet per se aanaangesloten) ook in `ys` voorkomen. Merk op dat een deellijst dus ook een sublijst is, maar dat het omgekeerde niet hoeft te gelden.

a. Schrijf een funktie `deellijst` die nagaat of de eerste lijst deellijst is ven de tweede lijst.

b. Idem voor `sublijst`.

Opgave 7. Het *bubble sort* algoritme sorteert een lijst door er een aantal keren doorheen te gaan. Tijdens een gang door de lijst worden twee opeenvolgende elementen verwisseld als de eerste groter is dan de tweede. Na één doorgang staat dus het grootste element achteraan (het is naar achteren "gebubbeld"). Dit proces wordt herhaald, waarbij het laatste element na de vorige doorgang niet meer wordt beschouwd.

Schrijf een functie `bsort` die dit algortme implementeert. Deze functie kan gebruik maken van een hulpfunctie (`bubble`) die een enkele doorgang door de lijst implementeert.

Opgave 8. Het *min-max* algoritme neemt het minimum en het maximum uit de lijst en zet die vooraan en achteraan (respectievelijk). Dit proces wordt herhaald voor de lijst waaruit beide zijn verwijderd. Schrijf een functie `mmsort` die dit algoritme implementeert (de Haskell module `Data.List` kent een operatie `\\` die twee lijsten van elkaar aftrekt. Ga zelf na hoe die operatie precies werkt).

Opgave 9. Het algoritme *insertion sort* neemt bij het sorteren van een lijst de lege lijst als uitgangspunt en voegt de elementen van de te sorteren lijst één voor één op de juiste plaats toe. Schrijf de functie `isort` (de Haskell module `Data.List` kent al een functie `insert`). Maak gebruik van één van de `fold` functies.

Opgave 10. Het *merge sort* algoritme verdeelt de lijst in twee helften, sorteert beide helften (volgens dezelfde manier), en combineert (merget) beide gesorteerde helften zodanig dat het resultaat ook gesorteerd is. Schrijf de functie `msort`.

Opgave 11. Het *quick sort* algoritme verdeelt de lijst in elementen die kleiner zijn dan het eerste element, gelijk zijn aan het eerste element, en groter zijn dan het eerste element. Dit proces wordt herhaald voor de afzonderlijke lijsten. Schrijf de functie `qsort`.