Structuur van Computerprogramma's I
Examen Eerste Zit

Donderdag 18 januari 2018

Dit examen is **gesloten boek**. Neem voor **elke vraag** een volledig nieuw blad en schrijf op elk blad je naam, richting en het nummer van de vraag. Antwoorden in potlood worden niet aanvaard! Lees elke vraag **volledig** en **aandachtig** voor je aan je oplossing begint!

Veel succes!

1 Lijsten

```
(define transacties
  '(("WJ8-DCBQA-AO" BE -100)
    ("Q3D-POIBB-L8" NL -200)
    ("YWB-BLKEO-IO" US -3.1)
    ("JQB-CYO3P-CS" GB -142.2)
    ("WZL-UP2CD-9S" RU 7.3)
    ("8NB-L46V7-RP" CH 1500)
    ("TAB-1LGSU-RT" RU 2070)))
```

a) Donald heeft op de website van Deutsche Bank een lijst gedownload met daarin alle transacties die op zijn rekening uitgevoerd zijn. In deze lijst worden transacties voorgesteld door lijsten met 3 elementen; respectievelijk het rekeningnummer waar geld van komt of naar wordt overgeschreven, het land van oorsprong van deze rekening, en de hoeveelheid geld die is overgeschreven.

Donald, die helaas geen Scheme kan programmeren, vraagt nu aan jou om een speciale procedure (`bereken-saldos begin-saldo transacties`) te implementeren. Gegeven een initieel bedrag en een lijst van transacties, geeft de procedure een lijst met daarin het bedrag dat op zijn rekening stond na elke transactie. Het eerste element van deze lijst is dus het bedrag op Donald's rekening na de eerste transactie, het tweede element is het bedrag op zijn rekening na de eerste *en* de tweede transactie, enzoverder.

Hint: schrijf abstracties om je code netter te maken.

```
> (bereken-saldos 700 transacties)
(600 400 396.9 254.7 262.0 1762.0 3832.0)
```

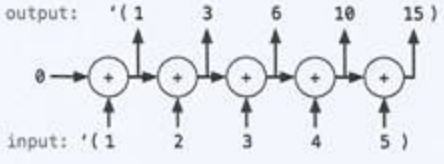
b) Levert je oplossing voor a) een recursief of iteratief proces op? Leg in je eigen woorden uit waarom je dit antwoord geeft (argumenteer bondig!). Schrijf de andere versie voor a) die *dezelfde* lijst teruggeeft.

c) Schrijf nu een procedure (`map-met-staat bewerking start-staat lijst`). Deze procedure is vergelijkbaar met de `map` procedure van Scheme, behalve dan dat de procedure die als argument wordt gegeven voor de parameter `bewerking` nu twee argumenten accepteert: een element van de lijst, en de *staat*. De procedure wordt dus toegepast op elk element van `lijst`, en het resultaat van de *vorige oproep*. Bij de eerste oproep, wordt de waarde van de parameter `start-staat` gebruikt als initiele waarde voor *staat*.

Bijvoorbeeld:

```
> (map-met-staat (lambda (staat el)
                   (+ staat el))
                 0
                 (list 1 2 3 4 5))
(1 3 6 10 15)
```

Het bovenstaande voorbeeld kan als volgt schematisch worden voorgesteld:

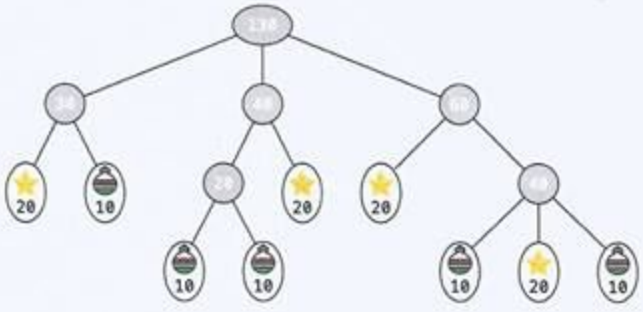


d) Gebruik nu de procedure `map-met-staat` om `bereken-saldos` te implementeren.

2 Bomen

Voor de "Dag van de Klant" wil Colruyt gratis kerstversiering aan zijn klanten geven. De ontwerper heeft gekozen om de kerstversiering, bestaande uit sterren en kerstballen, voor te stellen als een geneste lijst. In deze geneste lijst wordt per stuk decoratie bijgehouden wat het gewicht ervan is. Hieronder kan je een voorbeeld zien van zo'n geneste lijst, waarbij elke kerstbal 10 gram weegt, en elke ster 20 gram.

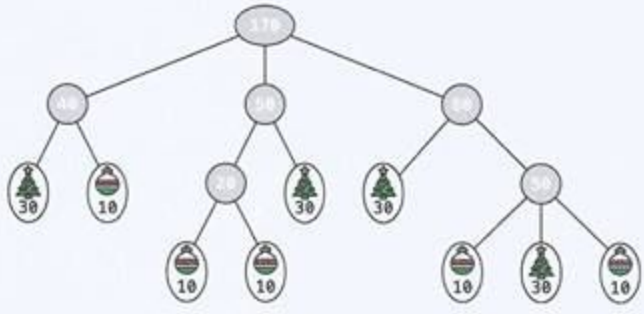
De ontwerper van de kerstversiering wil graag op elk moment weten hoe de gewichten verdeeld zijn in de boom. Daarom wil hij de geneste lijst omvormen tot een speciale boom, zoals je hieronder kan zien. In deze boom zijn de decoraties van de kerstversiering de bladeren van de boom. De knopen van de boom bevatten dan weer het totale gewicht van alle deelbomen onder de knop.



a) Schrijf nu een procedure `(vorm-om versiering)` die een geneste lijst van sterren en kerstballen omzet naar de hierboven beschreven boom. Hint: schrijf de nodige abstracties om de procedures vlot te implementeren.

```
> (define versiering
    '(((ster . 20) (bal . 10))
      (((bal . 10) (bal . 10)) (ster . 20))
      ((ster . 20) ((bal . 10) (ster . 20) (bal . 10)))))
> (vorm-om versiering)
(130 (30 ((ster . 20)) ((bal . 10))) (40 (20 ((bal . 10)) ((bal . 10))) ((ster . 20))) (60
((ster . 20)) (40 ((bal . 10)) ((ster . 20)) ((bal . 10)))))
```

b) Schrijf nu een procedure `(verander versiering oude-decoratie nieuwe-decoratie)`. Deze procedure neemt een boom als argument, en verandert elke decoratie die gelijk is aan `oude-decoratie` naar `nieuwe-decoratie`. Hou er rekening mee dat de gewichten in de boom ook moeten aangepast worden.




```
> (verander (vorm-om versiering) '(ster . 20) '(dennenboom . 30))
(170 (40 ((dennenboom . 30)) ((bal . 10))) (50 (20 ((bal . 10)) ((bal . 10))) ((dennenboom .
30))) (80 ((dennenboom . 30)) (50 ((bal . 10)) ((dennenboom . 30)) ((bal . 10)))))
```

3 Destructieve operatoren

```
(define transactions
  '(("WJ8-DCBQA-A0" BE -100)
    ("Q3D-P0IBB-L8" NL -200)
    ("YWB-BLKE0-I0" US -3.1)
    ("JQB-CY03P-CS" GB -142.2)
    ("WZL-UP2CD-9S" RU 7.3)
    ("8NB-L46V7-RP" CH 1500)
    ("TAB-1LGSU-RT" RU 2070)))
```

Als expert hacker wordt je door het Kremlin ingehuurd om Deutsche Bank te hacken. Het Kremlin is namelijk nogal bezorgd over Donald's rekening. Hierin staat duidelijk hoe Donald meerdere donaties heeft ontvangen van geheimzinnige bankrekeningen uit Rusland. Omdat het Kremlin niet wil dat mensen foute gedachten krijgen, vragen ze nu aan jou om zijn lijst van transacties aan te passen zodat de Russische bijdragen uit de lijst verdwijnen. Natuurlijk moet de balans van de rekening nog altijd correct zijn: als er een transactie verwijderd wordt die 100 euro op de rekening zet, moeten andere transacties aangepast worden zodat de som van transacties gelijk blijft. Verreken daarom dus het bedrag van een Russische transactie met de vorige transactie.

Let op: aangezien het gaat om een destructieve procedure mag je **geen enkele** nieuwe cons-cel aanmaken. Je mag hoogstens bestaande cons-cellen wijzigen of weglaten.

Implementeer de procedure `(change-transactions! transactions)` die een gegeven lijst van transacties destructief aanpast. Je mag er van uit gaan dat de eerste transactie niet van Russische afkomst is.

```
> (change-transactions! transactions)
done
> transactions
((WJ8-DCBQA-A0 BE -100) (Q3D-P0IBB-L8 NL -200) (YWB-BLKE0-I0 US -3.1) (JQB-CY03P-CS GB -134.9)
 (8NB-L46V7-RP CH 3570))
```

4 Objecten

a) Ontwerp een ADT `auto` dat je als object implementeert. Een auto heeft een batterij met een vaste capaciteit, gegeven in kilowattuur. Een auto weet hoeveel procent de batterij nog opgeladen is, en of hij al dan niet gekoppeld is aan een laadstation. Als de auto gekoppeld is aan een laadstation, kan de auto electriciteit, opnieuw gegeven in kilowattuur, onttrekken van dit station om de batterij terug op te laden.

Het koppelen en ontkoppelen van een auto aan een laadstation wordt altijd geinitieerd door een auto.

- De boodschap `charge` geeft terug hoeveel procent de auto opgeladen is.
- De boodschap `charge!` zal de auto volledig opladen, als die aan een laadstation gekoppeld is.
- De boodschap `koppel!` neemt een laadstation als argument en koppelt de auto er aan.
- De boodschap `ontkoppel!` zal de auto ontkoppelen van het laadstation.

b) Ontwerp een ADT `laadstation` dat je als object implementeert. Een laadstation weet op elk moment of het gekoppeld is aan een auto of niet, en hoeveel elektriciteit er in totaal al afgenomen is.

- De boodschap `withdraw!` zal een gegeven hoeveelheid stroom van het laadstation afnemen.
- De boodschap `koppel!` zal een wagen koppelen aan het laadstation.
- De boodschap `ontkoppel!` zal de wagen ontkoppelen van het laadstation.
- De boodschap `vrij?` zal teruggeven of het station al dan niet aan een auto gekoppeld is.

c) Ontwerp een ADT `laadpark` dat `n` laadstations beheert. Bij het aanmaken van een laadpark worden de bijhorende laadstations automatisch aangemaakt. Men kan gebruik maken van een laadpark door een auto het park te laten betreden. Bij het betreden van het laadpark, zal het laadpark de auto automatisch koppelen aan een vrij laadstation. Als er geen laadstation vrij is om de auto aan te koppelen, zal de `enter!` boodschap `#f` teruggeven.

- De boodschap `full?` zal teruggeven of het laadpark volzet is.
- De boodschap `enter!` kan gebruikt worden om een auto het laadpark te laten betreden.

e) Schrijf nu een scenario uit waarbij je een laadpark van grootte 2, en 3 auto's aanmaakt (capaciteit maakt niet uit). Probeer alle 3 auto's aan het laadpark te koppelen, deze op te laden, en 1 van de auto's daarna te ontkoppelen.

5 Stromen

Voor de volgende vragen mag je gebruik maken van de gekende stroom operatoren `cons-stream`, `head`, `tail`, `empty-stream?`, `map-stream`, `accumulate`, `stream-filter`, en `stream-append`. `the-empty-stream` kan je gebruiken om een lege stroom voor te stellen.

a) Schrijf een nieuwe stroom operator (`sliding-window stream n`) die, gegeven een oneindige stroom van elementen, een oneindige stroom van deelstromen teruggeeft. Elke deelstroom is een zogeheten *sliding window*. Dit wil zeggen dat de eerste deelstroom de eerste `n` elementen bevat van de input stroom, de volgende deelstroom bevat dan weer `n` elementen vanaf het tweede element van de input stroom, enzovoorts.

b) In beleggen is een *moving average* een gemiddelde prijs van een aandeel over een bepaalde periode. Je hebt een paar euro belegd in bitcoin en wil nu een stroom operator (`moving-average rates n`) implementeren die, gegeven een stroom van de dagelijkse bitcoin koers `rates`, een stroom produceert met de moving averages van de laatste `n` dagen.

Hint: implementeer ook een stroom operator die voor een gegeven eindige stroom het gemiddelde berekent.