5DV086

Lab 1 - P-hus

Submission v1.0

Teacher: Henrik Björklund

Tutor: Oscar Kamf

Gustaf Söderlund - et14gsd - (gusa0038@cs.umu.se)

February 10, 2022

Contents

List of Figures

1 Introduction

Syftet med laborationsuppgiften är att lära sig och förstå funktionell programmering. Det programmeringsspråk som uppgiften är skriven i är Haskell.

Programmet P-hus är ett program som är skrivet i haskell och har som uppgift att ta in en lista med bilars registreringsnummer, om de har åkt in eller ut ur parkeringshuset och vilken tid de kom/lämnade som indata. Programmet ska använda detta data och ta fram ett resultat som innehåller vilken bil som spenderat längst tid i parkeringshuset följt av en lista med alla andra bilar och deras spenderade tid i parkeringshuset.

2 Systembeskrivning

Programmet kan delas upp i 4 sektioner. Den första sektionen är Main funktionen *phus*. Det är denna funktion som bestämmer strukturen på hur indata och utdata av programmet kommer att se ut. Signaturen är definierad från specifikationen för laborationsuppgiften och har följande struktur:

```
phus :: [(String, Bool, (Integer, Integer))]
-> (String, [(String, (Integer, Integer))])
```

Den andra sektionen av programmet är inläsning av data. Funktionen *getVisitedCars* använder rekursion för att skapa en lista med dem bilar som har åkt in och ut från parkeringshuset. I rekursionen går funktionen igenom alla bilar i indata och kontrollerar om bilen har åkt in eller ut i parkeringshuset. För att kontrollera

om en bil har åkt in eller ut från parkeringshuset används funktionen *carInHouse* som kontrollerar om en Bool är satt till sann eller falsk. Bool variabeln är sann om bilen har åkt in i parkeringshuset och falsk om bilen har lämnat.

Har en bil åkt in i parkeringshuset används funktionen *carParkedTime* som tar fram bilens registreringsnummer med hjälp av funktionen *regName*. För att ta reda på hur länge bilen har varit parkerad används funktionen *getTime* som appliceras på när bilen kom in i huset och när den lämnade och för att sedan göra en beräkning på total tid i garaget under en vistelse används *timeSpent*.

Efter *getVisitedCars* har körts existerar nu en lista som innehåller alla bilar som har besökt parkeringshuset och hur länge respektive bil stod parkerad. I listan med bilar och parkeringstider kommer det existera dubbletter eftersom vissa bilar kan ha eventuellt besökt parkeringshuset ett flertal gånger under ett dygn. I den tredje sektionen av programmet kommer dessa dubbletter att slås samman.

Den tredje sektionen av programmet börjar med att slå samman alla dubbletter av bilar som har besökt parkeringshuset flera gånger under ett dygn. Det görs med hjälp av funktionen *mergeDuplicates*, signaturen för funktionen är följande

```
mergeDuplicates :: [PTime] -> [(String, [Time])]
```

Om det existerar dubbletter i listan av bilar kommer *mergeDuplicates* att slå samman de bilar som har samma registreringsnummer och skapa en lista av tupplar som innehåller tiderna för besöket i parkeringshuset. Efter *mergeDuplicates* slagit samman alla dubbletter behöver programmet slå samman listan med tuppler som innehåller innehåller tiderna för varje besök. Det görs med hjälp av funktionen *mergeTime* som rekursivt itererar igenom listan av bilar och använder funktionen

sumTime på alla element. *sumTime* kommer att returnera bilens namn och totala tiden som en bil spenderat i parkeringshuset. För att enkelt beräkna den totala tiden som en bil har spenderat i parkeringshuset konverteras tiden som är given i en tupple med timme och minuter till minuter det görs med hjälp av funktionen *convertToMinutes*. Då kommer det att existera en lista av Integers istället för en lista av tupplar med integers. Haskells inbyggda funktion *sum* appliceras på listan av Integers för att beräkna totala tiden. När totala tiden är beräknad behöver en konvertering göras tillbaka till tupplar av integers det konverteras med hjälp av funktionen *convertBack* som använder divison och modulus för att utföra beräkningen korrekt.

I den fjärde sektionen av programmet tas den bil som befunnit sig längst tid i parkeringshuset tas fram. Funktionen *getLongestParkedCar* med signaturen

```
getLongestParkedCar :: [PTime] -> (String, [PTime])
```

tar ut den bil som spenderat längst tid i parkeringshuset från listan av bilar. För att ta fram namnet på den bil som spenderat längst tid används funktionen *getCarName* som rekursivt itererar igenom listan av bilar. Vid varje iteration kontrolleras om den nuvarande bilen har kortare tid än nästa. Detta fortsätter tills listan är tom och den sista bilen som finns kvar i listan är bilen som spenderat längst tid i parkeringshuset.

3 Användarhandledning

För att använda programmet behöver man först installera GHCI på sin maskin. Det går att installera via www.haskell.org/ghc/. När GHCI är installerat startar man en terminal och navigerar fram till den mapp som innehåller filen Phus.hs. Därefter skriver man in kommandot *GHCI* i terminalen följt av kommandot *:l Phus.hs*. Progammet kommer att kompileras och är redo att användas. I figur 1 presenteras hur terminalen kommer att se ut efter kompilering.

```
itchy:~/programsprak> ghci
GHCi, version 8.8.4: https://www.haskell.org/ghc/  :? for help
Prelude> :l Phus.hs
[1 of 2] Compiling Register         ( Register.hs, interpreted )
[2 of 2] Compiling Phus             ( Phus.hs, interpreted )
Ok, two modules loaded.
*Phus>
```

Figure 1: Kompilering av programmet Phus.hs

För att använda programmet behöver användaren skriva kommandot *phus* följt av ett inputdata. Inputdata kan anges direkt i terminalen som en lista på följande struktur

$$[(String, Bool, (Integer, Integer))] \tag{1}$$

alternativt använda sig av ett program som t.ex. Register.hs som har färdig data att använda genom att ange ett alias, en demonstration av hur man anger inputdata som alias istället för en lista åskådliggörs i figur 2.

```
*Phus> phus day0
("DGH739",[("CBN395",(1,42)),("DGH739",(3,15)),("EYI469",(2,50)),("JGK835",(3,2)),("QYU034",(2,0)),("VCM345",(1,25))])
```

Figure 2: En körning av programmet Phus.hs med inputdata *day0*.

4 Testning

Programmet är konstruerat genom att dela upp problem i mindre delar som inläsning, slå samman dubbletter och hitta bilen som varit i Parkeringshuset längst. Där varje del delades upp i andra mindre funktioner för att lösa varje delproblem.

Varje funktion testades genom att manuellt ange indata och kontrollera dess utdata. Därefter testades varje funktion i Main funktionen där utdata kontrollerades igen tillsammans med de fördefinierade dagarna från Register.hs som indata.

5 Diskussion

Att utföra denna laborationsuppgift i ett språk som C eller Java hade nog varit enklare tänkte jag först skriva, men det kan bero på att jag är mer van i dem imperativa/objekorienterade språken. Det var första gången jag skrev ett funktionellt språk som Haskell. Till en början var det väldigt svårt att sätta sig in i tankesättet som man var tvungen att använda. Det jag tyckte var svårast var faktiskt att börja och komma igång. När man väl hade skrivit ett par funktioner och läst några guider blev det lättare att förstå tankesättet.

Laborationsspecifikationen var väldigt bra måste jag säga. Jag fick en snabb förståelse på vad den gick ut på och vad som behövdes göras. Väldigt bra med Register.hs filen att man kunde ta reda på vad utdata skulle vara för något beroende på vilken dag man tog.

6 källkod

```haskell
module Phus where

--Imports

import Data.List
import Data.Map
import Data.Ord
import qualified Data.Map as Map

--Type declatarions
type Parking' = (String,Bool,(Integer,Integer))
type PTime' = (String,(Integer,Integer))
type Time' = (Integer,Integer)

--main
phus :: [(String, Bool, (Integer, Integer))] -> (String,
 ↪  [(String, (Integer, Integer))])
phus day =
 ↪  getLongestParkedCar(mergeTime(mergeDuplicates(getVisitedCars
 ↪  day)))

-------------------------------------------------
-- Get the car that has parked the longest time -
-------------------------------------------------

```

```haskell
--Function used to make the signarute correct.
getLongestParkedCar :: [PTime'] -> (String, [PTime'])
getLongestParkedCar listOfCars = (getCarName ("",(0,0))
    listOfCars , listOfCars)

-- Calculate wich car has spent most time in the garage.
getCarName :: PTime' -> [PTime'] -> String
getCarName car [] = regName' car
getCarName car (head:tail)  = if( getTime' car < getTime' head
    )
                                then getCarName head tail
                              else getCarName car tail

-- Get the total time spent in garage. Helps getCar func to
    decide wich
-- car that has spent most time in the garage
getTime' :: PTime' -> Integer
getTime' (_,(h,m)) =  h*60 + m

-- get the car regname and print it.
regName' :: PTime' -> String
regName' (name, _) =  name


-----------------------------------------
-- Merge the duplicates and merge the time -
```

```haskell
-----------------------------------------

--merge the duplicates in the list.
mergeDuplicates :: [PTime'] -> [(String, [Time'])]
mergeDuplicates carList = toList $ fromListWith (++) [(k, [v])|
    (k , v) <- carList]

--merge the times and make correct signature
mergeTime :: [(String, [Time'])] -> [PTime']
mergeTime [] = []
mergeTime (head:tail) = sumTime head : mergeTime tail

-- merges the list of timestamps to total time.
sumTime :: (String, [Time']) -> PTime'
sumTime(carname, time) = (carname, convertBack (sum
    (convertToMinutes time)))

-- Convert the minutes to (Hour,Minutes) using divison and
    modulus.
convertBack :: Integer -> Time'
convertBack time =  (div time 60, time `mod` 60)

-- Convert the list of tuples to minutes to easier calculate
    the total time.
convertToMinutes :: [Time'] -> [Integer]
convertToMinutes [] = []
```

```haskell
convertToMinutes ((h,m):tail) = (h * 60 + m) : convertToMinutes tail


-------------------------------------
-- Get all the cars and their time -
-------------------------------------


-- check how long car has been parked using recursion
-- The list that is returned will have duplicates.
getVisitedCars :: [Parking'] -> [PTime']
getVisitedCars [] = []
getVisitedCars (head:tail) =
                        if(carInHouse head)
                            then carParkedTime head tail : getVisitedCars tail
                        else getVisitedCars tail

-- Get the car name and the time the car spent in the garage during the visit.
carParkedTime :: Parking' -> [Parking'] -> PTime'
carParkedTime car (head:tail) =
                        if (carInHouse car == False)
                            then ("",(0,0))
                        else
```

```
91                                  if(regName car == regName head)
92                                      then (regName car ,timeSpent
                                        ↪  (getTime car, getTime
                                        ↪  head))
93                                  else carParkedTime car tail
94
95  -- get the timespent in phus for one visit.
96  timeSpent :: (Time',Time') -> (Integer,Integer)
97  timeSpent ((a,b),(c,d)) =
98                              if( d-b < 0)
99                                  then (c-a-1, d-b+60)
100                             else (c-a, d-b)
101
102 -- is car still in house? Use the bool to check.
103 --True = car entered house, false = left house
104 carInHouse :: Parking' -> Bool
105 carInHouse (_,bool,_) = bool
106
107 -- get the car regname
108 regName :: Parking' -> String
109 regName (name, _, _) = name
110
111 -- get time for a car.
112 getTime :: Parking' -> Time'
113 getTime (_,_,time) = time
```

7 Referenser

http://learnyouahaskell.com - Gratis pdf med guider och information om haskell som använts under projektets gång.