CS316 2014/5, Worksheet 2

This practical is designed to help you to understand what you have been learning in the lectures about tuple types and lists. In particular, it is expected that many of your answers will use list comprehensions and the functions `map`, `filter`.

The practical comes in two sections. The first section contains a series of questions to get you started while the questions in the second section combine to form a larger application.

Download the file Prac2.hs and edit the file to contain your answers. Since this is the first time you will have written substantial functions of your own, I have decided to help you out by giving the occasional hint. You may work in groups of upto 3, but your answers can only be marked in the lab on Monday 20 October. All members of your group need to be present and your answers must be complete at the beginning of the lab. Half the marks will be for code correctness and half for a SHORT explanation of why your code is corect.

1 Section A

Question 1: Evaluate the lists `l1,l2,l3,l4` and write some other list expressions which involve the functions `take, drop, zip, reverse`. Describe in words what these functions do by writing comments in the file prac2.hs

Question 2: Write a function `addup :: Int -> Int` which takes a number `n` as input and returns `0+1+2+..+n`. Your function should only use list comprehensions and the inbuilt function `sum`. Use the function `addup` to define another function `tri :: Int -> [Int]` which takes as input a number `n` and returns the list

```
[addup 0, addup 1, addup 2, .., addup n]
```

In both cases, if the input is negative, an appropriate error message should be given. [*Hint: For the second part, of this question, you are being asked to apply a certain function to every element of the list* `[1 .. n]`].

Question 3: In the lectures you have seen the problem of summing the first `n` square numbers. Write a Haskell function `lssquares::Int->Int` which uses list comprehensions to sum the first `n` square numbers. [*Hint: Start with the list* `[1`

`.. n]`, *turn it into the list* `[1*1, .., n*n]` *and then apply the function* `sum` *to the result*].

Question 4: You saw the following function in the lectures which calculates a list of factors of a number. Rewrite the `factors` function by using the function `filter` instead of list comprehensions

```
factors :: Int -> [Int]
factors n = [ x | x <- [1 .. n], n `mod` x ==0 ]
```

[*Hint: First, define a function* `isFactorOfn :: Int -> Bool` *which takes a number as input and returns* `True` *if the input is a factor of* `n`]

2 Section 2: Building a Football Table

In this section you will write a larger application which will take as input the results of a series of football matches and will return as output the associated football table. Infact I have written the functions to sort the table and print it out — you have to define the rest of the functions. Dont worry if you don't know anything about football as everything will be explained. Also if you get stuck on one part of the question, you can test if your answers to the later questions are correct by running them on the various test data which are supplied.

The football matches take place between teams in the following league

```
league = ["Newcastle", "Sheffield", "Sunderland","Liverpool"]
```

A football match consists of a home team, the number of goals they scored, an away team, and the number of goals the away team scored. These types are modelled as follows:

```
type Match = ((Team,Goals),(Team,Goals))
type Team = String
type Goals = Int
type Points = Int
```

Notice that we are modelling a match as a pair type, the first component of which is also a pair type and so is the second component. For instance, in a

recent match, Newcastle played at home against Sheffield and won 8-0. This would be represented as the value

```
((Newcastle,8), (Sheffield,0)) ::  Match
```

In the file prac2.hs there is a list of matches called `scores` which you should look at. This list will act as test data for the remaining questions.

Question 5: Each team taking part in a match scores a number of points as follows:

- If the home team scores more goals than the away team, then the home team scores 3 points and the away team scores 0 points
- If the home team scores the same number of goals as the away team, then the home team scores 1 point and the away team scores 1 point
- If the home team scores less goals than the away team, then the home team scores 0 points and the away team scores 3 points

Write a function `points :: Match -> ((Team,Points),(Team,Points))` which takes a match as input and returns the teams together with the points scored by the team.[*Hint: Since the input of the function* `points` *is a pair type you want to use a pattern for the input and not a variable. Also there are various conditions to be checked so your function should be defined using guards*]

Question 6: Write a function `homeresults :: [Match] -> [(Team,Points)]` which takes a list of matches as input and returns a list of pairs containing the home team of a match and the number of points scored by the team in the match. The result of running the function `homeresults` on the input `scores` is included in the file as `homescores`.[*Hint: Apply the function* `points` *to every match in the input. You will then have to apply another function to every element of the resulting list*].

Similarly write a function `awayresults :: [Match] -> [(Team,Points)]` which takes a list of matches as input and returns a list of pairs containing the home team of a match and the number of points scored by the team in the match.

Finally, write a function `results :: [Match] -> [(Team,Points)]` which takes a list of matches as input and returns a list consisting of the home results and the away results joined together into one list. Applying the function `results` to the input `scores` should give the list `resultScores` in the file prac2.hs

Question 7: Write a function `performance :: [(Team,Points)] -> Team -> Points` which takes a list of results and a team as inputs and returns the total number of points scored by that team as a consequence of those results. [*Hint: Use a list comprehension to answer this question. Take the input list and keep only those elements whose first component is the team we are looking for. Form a list containing only the second component of those elements we are keeping. Finally, apply the function* `sum` *to this list*]

Question 8: Write a function `collect :: [(Team,Points)]->[(Team,Points)]` which returns a list consisting of each team in the league and the number of points scored. *Hint: Use a list comprehension for this question. For each team in the list* `league`*, use your answer to question 7 to calculate the number of points that team has scored and put these points, together with the team name, into a list*].

I've written the functions required to sort the table into order and print the results. Once you have finished question 8, uncomment the remaining functions and type

```
showTable scores
```

and you should get the following league table. Proving what we knew all along. That Newcastle United are the greatest football team! By the way, imagine writing this program using Java!

```
Newcastle       14
Sheffield       13
Liverpool       7
Sunderland      0
```