CS_205 Declarative Programming 26 October 2020
Ulrich Berger, Monika Seisenberger

Assignment 1: Due 9 November 2020

1. This coursework will be submitted in **groups of three** (at most two groups may have a size of two). As a group, you are asked to get together, discuss your ideas, plan the solutions, compare your solutions, etc. Everyone needs to be able to explain the main ideas in the submission.
2. Please register in groups of three on CANVAS by Thursday 29 October.
3. Please submit **exactly one Haskell file for the whole coursework** and add the answers to additional questions as comments. The file must be named `<your groupnumber>.hs` (for example `12.hs`).
4. The university takes plagiarism very seriously. With the submission you take responsibility that this is the group's own solution, and that the group has implemented the code, rather than someone having taken it from somewhere else.
5. To avoid confusion please **put your group name, your names and student numbers at the beginning of the file**. If somebody has not contributed to, say, one question in your solution, you can state this at the beginning of the file, and they will not be awarded the marks for it. However, please keep this simple.
6. The required Haskell functions **must be named as prescribed in the questions**.
7. **The submitted file must compile**. If you cannot get certain functions to work, then comment them out and (very briefly) explain the problem.

Rules for awarding marks:

- Full marks are awarded for a solution that is correct, complete, well-structured, makes good use of functional programming concepts, and, in case of a complex solution, contains explanations of the overall strategy and the roles of the helper functions.
- Up to 10 extra marks may be awarded for writing a well-structured document (for example, make it clearly visible where a new question starts) and complying with the rules set out above (1 file, correctly named, group name and authors names at the beginning, use of prescribed function names, etc.)

Question 1. a) Write a Haskell function `average` that computes the average of three numbers. (Note: For simplicity you may use input type `Float`.) Add another function that computes how many of these three numbers are above the average. Give two solutions for this function, called `howManyAboveAverage1` and `howManyAboveAverage2`, which are different in nature, that is, implement different algorithms (note that expressing the same algorithm with if-then-else and guarded equations does *not* count as two different algorithms).

Run your programs with your student numbers and include the result as copy into your solution.

Now, do the same question with a list of numbers instead of three numbers and call it `howManyAboveAverageL`. (Only one implementation of howManyAboveAverage needed). You need to be careful when using floats and integers in an expression as Haskell does not do automatic type conversion. (Please include the test and its Haskell response as a comment.)

[20 marks]

b) Optional: Use the following method to check the correctness of your programs: Add to the top of your file

```
import Test.QuickCheck
```

and write a function as below (please complete its type) that compares the two implementations of your howManyAboveAverage functions:

```
checkCorrectness x y z =
     howManyAboveAverage1 x y z == howManyAboveAverage2 x y z
```

In the command-line then do the check by typing.

```
quickCheck checkCorrectness
```

Include the response of Haskell as a comment.

Question 2. Define a function `split` that takes a function `p :: a -> Bool` and a list `xs :: [a]` as arguments and produces a tuple of two lists, one with the elements `x` in `xs` for which `p x` is `True`; the other with the elements `x` in `xs` for which `p x` is `False`. Give three different solutions for this, called `split1`, `split2`, `split3`, using list comprehension, higher order functions, and recursion. Give an interesting example where you compare your implementations using the digits of your student numbers.

Add the results of the runs as a comment to the file.

[10 marks]

Question 3. Pizzeria Alfredo sells pizzas of arbitrary sizes and numbers of toppings. The owner Alfredo wishes to have a program that computes the selling price of a pizza depending on its size (given by its diameter in cm) and the number of toppings. The pizza base costs £0.001 per cm^2 and the cost for each topping is £0.002 per cm^2. Since Alfredo also wants to make some profit, he multiplies the cost of a pizza by a factor of 1.6. Can you Alfredo with a suitable Haskell function? Call your solution function `alfredo`.

Is Pizza Bambini (tomatoes, mozzarella, ham, salami, broccoli, mushrooms, 14 cm) more expensive than Pizza Famiglia (tomatoes, mozzarella, 32 cm)?

Use your program `alfredo` to define a Boolean, called `pizzaCompare`, that answers this question. Include the Haskell output as a comment.

[10 marks]

Question 4. Starting from the programs

```
divides :: Integer -> Integer -> Bool
divides x y = y `mod` x == 0

prime :: Integer -> Bool
prime n = n > 1 &&  and [not(divides x n) | x <- [2..(n-1)]]

allPrimes :: [Integer]
allPrimes = [x | x<- [2..], prime x]
```

define a function `allPrimesBetween:: Integer -> Integer -> [Integer]` that produces all prime numbers between two bounds.

Next, produce an infinite list `primeTest :: [Bool]` such that the nth position of the list is `True` if n is prime, and `False` otherwise. Then, produce an infinite list `primeTestPairs` such that the nth position is the pair (n, b) where the value b is `True` if n is prime and `False` otherwise. Evaluate parts of these lists using the Haskell function `take` and include the results as a comment.

Finally, define a function `primeTwins` that, given n, computes how many prime twins are amongst the first n prime numbers. Use this to answer the questions: How many prime twins are there amongst the first 20 prime numbers; how many are there amongst the first 2000 prime numbers?

[20 marks]

Question 5. (Phonetic Search) The following is a programming test for job applicants provided by a leading UK company: Your program must read a list of surnames from a file `surnames.txt` one per line. The command-line argument to your program will be also a list of surnames. For each of these surnames you must print out all of the names from `surnames.txt` that match those from the command-line. If the command-line arguments were "Smith" and "Jones", and each matched three names in the input data, then you should print out (with the appropriate punctuation as shown):

```
Smith: Smith, Smyth, Smythe, Smid, Schmidt
Jones: Jonas, Johns, Saunas
```

The matching algorithm is as follows:

1. All non-alphabetic characters are ignored
2. Word case is not significant
3. After the first letter, any of the following letters are discarded: A, E, I, H, O, U, W, Y.
4. The following sets of letters are considered equivalent
   - A, E, I, O, U
   - C, J, K, Q, S, X, Y, Z
   - B, F, P, V, W
   - D, T
   - M, N
   - All others have no equivalent
5. Any consecutive occurrences of equivalent letters (after discarding letters in step 3) are considered as a single occurrence

The rules should be applied in that order. So, given a file `surnames.txt` that contains the following data:

```
Smith
Smyth
Smythe
Smid
Schmidt
Smithers
Jonas
Johns
Johnson
Macdonald
Nest O'Malett
Chen
```

```
Gan
Ericsson
Erikson
Saunas
Van Damme
```

Running your program from the command-line like this

```
phonetic ["Jones", "Winton"]
```

Should print out:

```
Jones: Jonas, Johns, Saunas
...
```

The file `surnames.txt` is available on Canvas. Please download it and use it when running the program.

[30 marks]

Question 6 Write a well-structured document complying with the rules as described at the beginning.

[10 marks]

Overall available: [100 marks]