CS_205 Declarative Programming 28 October 2019
Monika Seisenberger

Assignment 1: due 11 November 2019

Instructions:

1. Please note this coursework is to be solved in groups of 3. You may select/register your group via Blackboard until Monday 4 November before the lecture. If you have not chosen a group by then, you will be allocated to a group afterwards. No swap of groups is possible after 4th November.
2. Produce a single Haskell file with the solutions to all programming questions. The filename should be `<your group number>.hs`. Submit via the link in the assignment folder on blackboard, the latest attempt will be marked.
3. Marks will be awarded for both correct functionality and good code quality as well as for complying with the instructions. Please add the answers to additional questions as comments as requested.
4. By submitting this coursework electronically, you state that you fully understand and are complying with the University's policy on Academic Integrity and Academic Misconduct. The policy can be found at www.swansea.ac.uk/academic-services/academic-guide/assessment-issues/academic-integrity-academic-misconduct.

Question 1 a) Write two different functions `max1` and `max2` that compute the maximum of three numbers (without using built-in functions for computing the maximum). (Note: your two functions need to be computationally different, thus the same algorithm once implemented with an if-construction, once with a guarded equation, does not qualify.)

Run your programs with your student numbers (if you are only two in your group use 976543 as a third number), and include the Haskell query and your results as a comment into your solution.

b) The following demonstrates a method how you could check the correctness of your programs: Add the following code 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 =
     max1 x y z  == max2 x y z
```

In the command line then do the check by typing.

```
quickCheck checkcorrectness
```

Include the response of Haskell as a comment.

Additional question: can you think about possible weaknesses of this correctness check; how could it be improved? [Add your response as a comment.]

[10 marks]

Question 2 Pizzeria Luigi sells pizzas of an arbitrary size and with an arbitrary number of toppings. The owner Luigi wishes to have a program that allows to compute 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 are £0.002 per square centimetre and the costs for each topping £0.6 per topping Since Luigi also wants to make some profit, he multiplies the costs of a pizza by a factor 1.6. The result should be rounded to two digits. Can you help Luigi with a suitable Haskell function `luigi :: Float -> Float -> Float`?
Is Pizza Bambini (tomatoes, mozzarella, ham, salami, broccoli, mushrooms, 15 cm) more expensive than Pizza Famiglia (tomatoes, mozzarella, 32 cm)?

Add an answer to this question as a comment.

[7 marks]

Question 3 Write a function that counts how many digits occur in a given list of characters. Do this in different ways, using

(i) list comprehension

(ii) higher order functions

(iii) recursion

You may use the Haskell function `isDigit :: Char -> Bool` that tests whether or not a character is a digit. Store your results in `count1, count2, count3 :: Int`.

[7 marks]

Question 4

(a) In this part we will start off with a bit of nice string crunching! As a warm up, create a function `normaliseSpace :: String -> String`, that removes all double spaces in a string (and merges them into one single space).
E.g.: `normaliseSpace "  ab c  def  " = " ab c def "`

(b) Next, we will do something that makes parsing e.g. names easier. Write two functions, `normaliseFront :: String -> String` and `normaliseBack :: String -> String`, that remove preceding and tailed spaces, respectively. So

```
normaliseFront "    abc d  e   " =     "abc d  e   " and
normaliseBack  "    abc d  e   " = "    abc d  e"
```

For the remainder of this part the strings can be assumed to be finite.

(c) Great! Now we just put everything together. Write a function `normalise :: String -> String` that combines the effect of all previous functions, so that the input string is stripped of leading spaces, following spaces and inbetween never has any more but one consecutive space.

(d) This part is about generalising a bit what we have done before. First, we write some similar functions for the previous task, called prefix, substr and postfix, all with the type `String -> String -> Bool`. Their task is obvious from their name, so here are some examples of how they should work:

```
prefix    "abc" "abcdef" = True
prefix    "abc" "aabcaa" = False
substring "abc" "aabcaa" = True
```

```
substring "abc" "aabbcc" = False
postfix   "abc" "defabc" = True
postfix   "abc" "aaabcc" = False
```

For this task, all arguments are assumed to be finite. Furthermore it goes without saying that this time, no strings need to be normalised. (Do NOT use the library functions isPrefixOf, isSuffixOf or isInfixOf!)

(e) Splendid! Now we try for something more difficult. Have you ever written a mail to 200+ students and wanted to address each by name? We did. Write a function `substitute :: String -> String -> String -> String`,
such that `substitute xs ys zs` replaces the string `xs` in `zs` by `ys`,
e.g. `substitute "#name" "yourname" "Dear #name, you are awesome!"` yields `"Dear yourname, you are awesome!"`

Hints: `substitute [] _ _` is undefined. Always the "first match" is replaced, i.e. `substitute "aa" "b" "aaa"` yields `"ba"`. There can be more than one or no instances of the string to be replaced.

[10 marks]

Question 5 and 6 as well as general marks to be added tomorrow.