CS-205 coursework: due November 18th 2022, 11am

1. Submission for this coursework will be individual; detailed submission instructions will be given on Canvas next week.

2. Automated tests will be used to award marks; you should scrupulously follow instructions. In particular, the types of your functions and their names should match exactly what is in this document (case included).

3. Feel free to indicate any further relevant information in a comment at the beginning of the file (please keep it as simple as possible).

4. You are allowed to use any function that is shipped with `ghc`. If you feel some function is so basic it should come with a standard Haskell distribution, recall that you can look up functions by types or names on `https://hoggle.haskell.org`. You are free to add import statements at the top of your file to access those functions (like `import Data.Char` to access `isDigit`).

5. By submitting 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 `https://myuni.swansea.ac.uk/academic-life/academic-misconduct`. The consequences of committing academic misconduct can be extremely serious and may have a profound effect on your results and/or further progression. The penalties range from a written reprimand to cancellation of all of your marks and withdrawal from the University.

6. There is a total of 40 marks to be gained by answering the questions. 5 additional marks will be given for adherence to the submission rules and 5 marks for good style, so there is a total of 50 marks to be earned.

Question 1. A pizzeria sells pizzas of an arbitrary size and with an arbitrary number of toppings. The owner, Vittorio, 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, as well as the number of sauces added. The pizza base costs £0.002 per cm^2 and the costs for each topping are £0.001 per cm^2. Each sauce costs £0.55 Since Vittorio also wants to make some profit, he multiplies the costs of a pizza by a factor 1.5. Additionally, if the customer requires a delivery, Vittorio additionally charges £0.9 per miles between his shop and the delivery location to compensate the deliveryman. Can you help Vittorio with a suitable well-structured Haskell function

```haskell
pizzaPricing :: Float -> Int -> Int -> Float -> Float
```

where the first input is the pizza diameter, the second one is the number of toppings, the third one is the number of sauces and the last one indicates the distance to the delivery point (0 if the customer comes to pick up their pizza)? The computed result should be truncated after two decimal digits, e.g. if you would have a result of 6.5999 by applying the above rules, then the resulting price should be 6.59.

[8 marks]

Question 2. A list function Write a function

```haskell
howManyAboveAverage :: [Double] -> Int
```

that counts how many numbers in the input list are greater or equal than their average. For instance, the average of `[2,3,5,5]` is `3.75`, so `howManyAboveAverage [2,3,5,5]` should be equal to `2`.

[5 marks]

Question 3. Representation and evaluating arithmetic expressions The goal of this exercise is to look at how to convert between different representations for arithmetic expression involving `+`, `*`, `-`, `/` and nonnegative integers.

Consider the following datatype to represent such expressions in a tree-like manner.

```
data BinOp = Plus | Minus | Times | Div
  deriving (Show, Eq, Enum)
data Expr = Const Int | Op BinOp Expr Expr
  deriving (Show, Eq)
```

One example of a value of type `Expr` representing $\frac{4+2*33}{2}$ would be

```
sampleExpr :: Expr
sampleExpr = BinOp Div (BinOp Plus (Const 4) (BinOp Times (Const 2) (Const 33))) (Const 2)
```

We encourage you to define more examples to test your answers.

a) Write a function

   ```
   expr2String :: Expr -> String
   ```

   that prints out an arithmetic expression as a string of characters. For instance, `expr2String sampleExpr` should evaluate to something like `"(4 + (2 * 33)) / 2"` or `"(4 + 2 * 33) / 2"` but certainly not to `"4 + 2 * 33 / 2"`.

   To get full marks, you should have a solution which is correct and uses a minimal amount of parentheses, while still allowing to pase back the correct `Expr` unambiguously, assuming that we follow the usual rules for the precedence of the operators (i.e., `+`, `-`, `*`, `/` is the list of operators with increasing priority). To guide you, we provide the code of a function

   ```
   string2Expr :: String -> Maybe Expr
   ```

   that attempts to create a `Expr` from a `String` in the on Canvas in a file names `string2Expr.hs`. Your function should be an inverse of that, in the sense that we should have for every `e :: Expr`

   ```
   string2Expr (expr2String e) == Just e
   ```

   [8 marks]

b) Say that an `Expr` is left-balanced if it contains no subexpression of the shape `Op op l (Op op l' r')` for `op` being `Plus` or `Times`. All expressions can be rewritten to an equivalent left-balanced one containing the same constants because $+$ and $-$ are *associative*, meaning that we have that $(x+y)+z = x+(y+z)$ and $(x*y)*z$ (but `-` and `/` aren't)!

   Write a function

   ```
   rewriteAssoc :: Expr -> Expr
   ```

   that turns every expression into an equivalent left-balanced one with the same constants. **[7 marks]**

Question 4. *Countdown* is a British[1] TV game show that is still airing nowadays. In one of the games, the contestants are given a list of numbers, as well as a target number, and attempt to reach the target number by applying the basic arithmetic operations `+`, `*`, `-` and `/` using the numbers they are given. There are some natural restrictions in how they can use those numbers:

- Negative or fractional numbers are not allowed at any stage of a computation. So for instance, $(3-5)+10$ and $5/2+3/2$ are never valid answers.
- All numbers given to reach the target must be used at most once in the computation. For instance, if the list $5, 5, 4, 3, 2$ is given, $5+5-4$ and $3*2$ are valid answers to reach 6, but $5-4+3+4-2$ is not.

To limit the search space, let us only consider expressions including `+` and `*` for this question.

Write a function

[1] Actually a French export; not sure where else TV shows with the same conceit air.

```
countdownAllSolutions :: [Int] -> Int -> [Expr]
```

that takes as input a list of numbers, a target number, and lists all of the arithmetical expression that a contestant could use to compute the target from the given number. Your expression should follow the rules, i.e., no subexpression should evaluate to a negative number.

You should not hesitate to write auxiliary functions for this exercise. In particular, one useful functions would be a functions

```
allSplits :: [Int] -> [([Int],[Int])]
```

that would enumerate all possible ways of splitting a list of elements in two distinct sublists containing the same elements. If you do not manage to get a full solution, you may get partial marks for implementing this.

We expect that you would be able to run your solution in reasonable time for lists of 5 numbers. If you manage to go significally beyond by programming this efficiently (which would require using more advanced methods than what we have taught at the release of the coursework, namely how to do *dynamic programming* or *memoization* in Haskell) *and* integrate all four arithmetic operations in your results, we may award bonus marks.

Partial marks would be awarded if you output at least one solution whenever it exists and the empty list when there is none.

[12 marks]