universität innsbruck

Functional Programming | WS 2019/2020 | LVA 703024+703025

Exercises 4 | Deadline: October 30, 2019, 6am

- Please write all your Haskell functions from this exercise sheet into a single .hs-file and upload it in OLAT.
- You can use a template .hs-file that is provided on the proseminar page.
- The file should compile with ghci.
- Once the file has been uploaded, it cannot be changed or resubmitted!

Exercise 4.1 *Booleans* **3 p.**

Write code for the function `boolFun :: Bool -> Bool -> Bool -> Bool` as specified by

x	y	z	boolFun x y z
False	False	False	False
False	False	True	False
False	True	False	True
False	True	True	True
True	False	False	False
True	False	True	True
True	True	False	True
True	True	True	False

in three different ways:

1. using `if then else` and `&&`, `||`, `not`; (1 point)
2. as before but without using `if then else`; (1 point)
3. using pattern matching. (1 point)

Exercise 4.2 *Enumerations* **4 p.**

1. As of the 2019 elections the Austrian National Council houses 5 parties. In total there are 183 members of parliament (MPs), which are distributed as follows: Oevp - 71 MPs, Spoe - 40 MPs, Fpoe - 31 MPs, Gruene - 26 MPs and Neos - 15 MPs.

   Since no party holds more than half of the seats, there is need to form a coalition between at least two parties.

   Write a function `coalition :: Party -> Party -> Bool` that takes two parties and returns whether they can form a government together or not (purely mathematically, by holding at least 92 seats).

   As intermediate steps, define an enumeration type `Party` and a function `mps :: Party -> Integer`.

   (2 points)

2. Define a type `Season` with constructors for the four seasons and write a Haskell function `daysInSeason` that returns the number of days in a given season. Make sure to also write down the type signature. (Assume the following durations: spring - 93, summer 94, fall - 90, winter - 89.)
   (a) Using pattern matching (1 point)
   (b) Using if-then-else and an `Eq instance` for `Season` (1 point)

Exercise 4.3 *Polymorphism* **3 p.**

1. Write a function `threeEqual` taking three arguments of the same, arbitrary type, which returns `True` if and only if all of them are equal. Also write down the type signature. (If you have done Exercise 4.2, check if this function works with `Season`) (1 point)

2. Write two different functions `foo` and `bar` of type `a -> a -> a`. Here, different means that for some input values `x` and `y` the result of `foo x y` is different from the result of `bar x y`. (1 point)

3. Is there a difference between the type signatures `a -> b -> a` and `c -> a -> c`? Give a short explanation in terms of how these types can be instantiated. (1 point)

4. Challenges to think about: Is it possible to define a Haskell-function of type `a -> a -> b`? How many functions are there of type `a -> a`? (0 points)