universität innsbruck

Functional Programming	SS 2020	LVA 703025
Exercise Sheet 5, 10 points		Deadline: Monday, April 20, 2020, 10am

- Watch the lecture of week 5 and study part 3 of the slides.[1].
- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- You can use the template .hs-file that is provided on the proseminar page[2].
- Your .hs-file should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 5.1 *Type Classes* **3 p.**

This exercise should show you how types and type classes can be used to model data. Consider the following data types and the following type class:

```
data City = Innsbruck | Munich | Graz | Gramais
data Country = Austria | Germany
data State = Tyrol | Bavaria | Styria

class Info a where
  population :: a -> Integer
  area :: a -> Integer
```

1. Make the types `City`, `Country` and `State` instances of the class `Info`, cf. slides 15 – 16 of part 3, using the following data: [3]

Location	Type	Population	Area in km^2
Innsbruck	City	132110	105
Munich	City	1471508	105
Graz	City	288806	127
Gramais	City	41	32
Bavaria	State	13076721	70550
Tyrol	State	754705	12640
Styria	State	1243052	16401
Germany	Country	83019203	357578
Austria	Country	8858775	83878

(1 point)

2. Define the following functions:

```
inhabitantsPerSkm :: (Info a) => a -> Integer
hasMoreArea :: (Info a, Info b) => a -> b -> Bool
hasMoreInhabitants :: (Info a, Info b) => a -> b -> Bool
```

[1] http://cl-informatik.uibk.ac.at/teaching/ws19/fp/slides/03x1.pdf
[2] http://cl-informatik.uibk.ac.at/teaching/ss20/fp/index.php#exercises
[3] Data taken from https://de.wikipedia.org/

The function `inhabitantsPerSkm` should return the number of inhabitants per km^2 rounded down. `hasMoreArea x y` should return `True` if `x` has a larger area then `y`, otherwise it should return `False`. `hasMoreInhabitants x y` should return `True` if `x` has more inhabitants then `y`, otherwise it should return `False`.

Note that the notation `(Info a, Info b) =>` means that `a` may only be instantiated by a type that is an instance of type-class `Info`, and additionally `b` can only be instantiated by instances of `Info`. (1 point)

3. Make the types `Country` and `State` instances of the following type class:

```
class Located a where
  isIn :: City -> a -> Bool
```

`isIn x y` should return `True` if city `x` is in state or country `y`, otherwise `False`. For example, `isIn Gramais Tyrol` should evaluate to `True`. (1 point)

Exercise 5.2 *Pattern Matching* 2 p.

```
fun_1 True True   = False
fun_1 True False  = False
fun_1 False True  = True
fun_1 False False = False

fun_2 True  _     = False
fun_2 False y     = y

fun_3 x     True  = not x
fun_3 _     False = False

fun_4 True  True  = False
fun_4 False True  = True
fun_4 _     False = False
```

We say that a binary Haskell function f is equal to g w.r.t. a set of inputs I, if and only if for all inputs $x \in I$ and $y \in I$ the equality $f\ x\ y = g\ x\ y$ is satisfied.

1. Consider the Haskell programs above. Which of the functions `fun_2`,...,`fun_4` are equal to `fun_1` w.r.t. inputs {`True`, `False`}? For each inequality, provide some input which distinguishes the functions. (1 point)
2. Which of the functions `fun_2`,...,`fun_4` are equal to `fun_1` w.r.t. inputs {`True`, `False`, $\bot$}? (As usual, $\bot$ is a special value that represents non-termination or abnormal termination via errors.) For each inequality, provide some input which distinguishes the functions. You should solve this part without invoking GHCi, but instead understand pattern matching as it is explained on slides 10 – 11 in part 3. (1 point)

Exercise 5.3 *Strings* 5 p.

Codes [4] map characters to strings of bits. Texts are then encoded by concatenating the codes of its characters (cf. substitution ciphers [5]). Suppose the following code, coding the first four letters of the alphabet as bit strings:

```
code :: Char -> String
code 'a' = "01"
code 'b' = "1"
code 'c' = "001"
code 'd' = "0001"
```

See slides 54 – 57 of part 3 of the lecture for functions on lists. Remember: A `String` in Haskell is a synonym for `[Char]`, i.e. a list of characters.

1. Write a function `encode :: String -> String` which encodes a string which only contains characters `'a'`, `'b'`, `'c'`, and `'d'` by a string of bits, based on the function `code`. For instance, encoding `"aba"` should yield `"01101"`. (1 point)

[4] https://en.wikipedia.org/wiki/Code
[5] https://en.wikipedia.org/wiki/Substitution_cipher

2. Write functions `isPrefix, neitherPrefix :: String -> String -> Bool` checking whether the first string is a prefix of the second, respectively whether neither string is a prefix of the other. Here a string is a prefix of any string obtained from it by appending. For instance, `"hell"` is a prefix of `"hello world"` (append `"o world"`) but `"ell"` and `"world"` are not. Note that the empty string `""` is a prefix of any string (append that string), and that any string is a prefix of itself (append the empty string). (1 point)

3. Write a function `decode :: String -> String` decoding bit strings into the original string of characters. For instance, decoding `"01101"` should yield `"aba"`. Do something sensible for strings that cannot be decoded such as `"0000"`.

   You may make use of the functions `take,drop :: Int -> [a] -> [a]` from the standard Prelude, which take resp. drop the given number of characters from a `String`, and `length` that yields the length of a list. The functions `isPrefix` from the previous item may be useful.
   Hint: since `Char` is an instance of `Enum`, `succ` can be used to enumerate them. (2 points)

4. A good encoding should be *injective*: different strings should have different encodings. For that it is sufficient that no code of a character is a prefix of the code of another character, a so-called *prefix* code.[6] Write a function `isPrefixCode :: Bool` checking that this property indeed holds for the function `code`, by checking `neitherPrefix` holds for all combinations of codes of different characters; since we code 4 characters, you should check the property for $\frac{4*(4-1)}{2} = 6$ combinations.

   Your implementation should be independent of the concrete codes for the 4 characters. E.g., changing the last line of `code` to `code 'd' = "00"`, `isPrefixCode` should return `False`. (1 point)

[6] `https://en.wikipedia.org/wiki/Prefix_code`