universität innsbruck

Functional Programming	WS 2019/2020	LVA 703024+703025

Exercises 7 — Deadline: November 20, 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 7.1 *Non-Recursive Data-Types* **5 p.**

1. Define two non-recursive datatypes `Polar` and `Cart` for coordinates in the polar coordinate system [1] and the cartesian coordinate system [2]. Think about choosing useful type synonyms for the components of our coordinates.

   Write a function `createPolar` and its type signature, which takes a radius and an angle in degrees and returns a polar coordinate with the angle in radians. (1 point)

2. Implement the functions `cart2Tuple` and `polar2Tuple`, that convert a `Cart` into a tuple and a `Polar` into a tuple. Add the corresponding type signature. (1 point)

3. Define two conversion functions `polar2Cart :: Polar -> Cart` and `cart2Polar :: Cart -> Polar` between the coordinate systems above.

   To convert the polar coordinates (r, φ) to cartesian coordinates (x, y) use:

   $$x = r \cdot \cos\varphi \qquad\qquad y = r \cdot \sin\varphi$$

   The cartesian coordinates (x, y) can be converted into the polar coordinates (r, φ) as follows:

   $$r = \sqrt{x^2 + y^2} \qquad\qquad \varphi = \begin{cases} \arccos(\frac{x}{r}) & \text{if } y \geq 0 \text{ and } r \neq 0 \\ -\arccos(\frac{x}{r}) & \text{if } y < 0 \\ d & \text{if } r = 0 \end{cases}$$

   where you should define a sensible value for d.

   Think about the mathematical definitions above and find suitable implementations in Haskell. For the calculation of φ use guarded equations.

   Some useful functions: `cos`, `acos`, `sin`, `sqrt`, `round`, `pi`.

   Hint: Be aware that all trigonometric functions work with radians. (2 points)

4. Write for the types `Cart` and `Polar` a `Show` instance so that `show` produces, given cartesian coordinates `1`, `1` or polar coordinates `(sqrt 2)`, `45` (in degrees), for both representations the same `String` `"(1.0,1.0)"`. The presented coordinates should be rounded to 1 decimal after the comma. (1 point)

[1] https://en.wikipedia.org/wiki/Polar_coordinate_system
[2] https://en.wikipedia.org/wiki/Cartesian_coordinate_system

Exercise 7.2 *Strings* **5 p.**

Codes [3] map characters to strings of bits. Texts are then encoded by concatenating the codes of its characters (cf. substitution ciphers [4]). 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"
```

For solving the exercises below, do *not* yet use pattern matching on lists. You may only use the functions on lists given in the lecture (slides 54 – 57 of part 3).

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)

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.[5] 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)

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