CSIS2258/COMP3258 Functional Programming 2014/15

Assignment 2 (Deadline: 11:59 AM HKT, March 27)

Code Correctness (10%)

Make sure that your code compiles and produces no warnings with GHC 7.8.3; that you did not change the provided names or types in the `Assignment2.hs` skeleton and **did not add extra import statements** on top of the provided ones. Recursive patterns should be defined using **list comprehensions or higher-order functions** (not plain recursion). Functions should terminate and not raise exceptions. Please include any extra code you wrote to test your code.

Code Style (10%)

Your code should be well-written (e.g. sensible names of functions or variables where appropriate), documented, well-formatted, and produce no warnings with HLint.

Submit your solution on Moodle in one file (use the attached `Assignment2.hs` skeleton) before the deadline (which is a cut-off time; no late submissions will be accepted).

Plagiarism

Please do this assignment on your own; if, for a small part of an exercise, you use something from the internet or were advised by your classmate, please mark and attribute the source in a comment. Do not use ***publicly accessible*** code sharing websites (pastebin, lpaste, fpcomplete, GitHub public repository…) for your assignment to avoid being suspected of plagiarism.

Exercise 1 (5%)

Write a function `tailsLC :: [a] -> [[a]]` that implements the Haskell `tails` function using list comprehension.
(The `tails` function returns all final segments of the argument, longest first. For example,

```
> tails "abc" == ["abc", "bc", "c",""])
```

Exercise 2 (5%)

Write a function `tailsHOF :: [a] -> [[a]]` that implements the Haskell `tails` function using higher-order functions.

Exercise 3 (10%)

```
type PhonePad = [(Int,String)]
```

represents a simple phone keyboard character layout, i.e. the first item of the pair represents the button number, the second item of the pair represents the characters under that button.

Using list comprehension, write a function `encodeLC :: PhonePad -> String -> String` that takes a phone keyboard character layout and a string, and returns a string which encodes phone button presses needed to write that string with that keyboard layout (i.e. as if you were to write that string on your phone). The function should be case insensitive and ignore characters that are not present in the keypad layout. For example,

```
usualPad :: PhonePad
usualPad = [ (1, ""), (2, "abc"), (3, "def"),
             (4, "ghi"), (5, "jkl"), (6, "mno"),
             (7, "pqrs"),(8, "tuv"), (9, "wxyz"),
                         (0, " ")]

> encodeLC usualPad "Hello World!"
"4433555555666096667775553"
```

HINT: You may want to define an auxiliary helper function to transform `PhonePad`.

Exercise 4 (10%)

Using higher-order functions, write a function `encodeHOF :: PhonePad -> String -> String` that implements the same behavior as the function from Exercise 3 (`encodeLC`).

Exercise 5 (20%)

Exercises 5 and 6 are implementing a new "Phone Number Method for Day-of-Week Calculation" that was invented by Hans Lachman on December 13, 2014, in Hong Kong: http://oz.ccnet.us/dayofweek/method14.txt

```
type Mnemonic = String

type Year = Int
type Month = Int
type Day = Int
type Date = (Year, Month, Day)
baseMnemonic :: Mnemonic
baseMnemonic = "IRS FLU FOX IRA"
```

(a) Write a function `monthToPhone :: Mnemonic -> PhonePad -> Month -> Int` that takes a mnemonic (a string that is easy to remember), a phone keyboard layout (as defined in Exercises 3 and 4), and a month (represented by Int), and returns the "month's phone number" defined as follows: given a month M and the mnemonic, select Mth letter (i.e. excluding whitespaces) from the mnemonic and return the phone button number on which it can be found (case insensitive). "For example, December is the 12th month, so select the mnemonic's 12th letter, "A", which is found on phone button 2.":

```
> monthToPhone baseMnemonic usualPad 12
2
```

NOTE: Feel free to use either list comprehensions or higher-order functions.

(b) Write a function `dowNumber :: Mnemonic -> PhonePad -> Date -> Float` that takes a mnemonic, a phone keyboard layout, and a date (a triple of integers, representing year, month, and a day) and returns the final calculated number defined as follows:

"Step 1: YYYY.MM * 1.25 + 0.97 = (drop the fraction, integer part is ZZZZ)
Step 2: ZZZZ + # + DD =
Step 3: / 7 ="

where "YYYY.MM" is the 4-digit year followed by a decimal point and the 2-digit month,”#" is the month’s phone number (as returned by `monthToPhone`) and DD is the day. E.g. April 15, 2000:

“2000.04 * 1.25 + 0.97 = 2501.02
2501 + F + 15 = 2519 ("F" is on phone button 3)
/ 7 = 359.85715”

```
> dowNumber baseMnemonic usualPad (2000,4,15)
359.85715
```

Exercise 6 (30%)

(a) Write a function `firstDecimalDigit :: Float -> Int`
 that takes a floating point number and returns the first digit after the decimal point.

```
> firstDecimalDigit 359.85715
8
```

(b)
```
dayPad :: PhonePad
dayPad = [ (1, "Monday"), (2, "Tuesday"), (3, ""),
            (4, "Wednesday"), (5, "Thursday"), (6, ""),
            (7, "Friday"),(8, "Saturday"), (9, ""),
                        (0, "Sunday")]

data PhoneDay = Monday | Tuesday | Wednesday | Thursday | Friday |
Saturday | Sunday
  deriving (Eq, Ord, Show, Read, Bounded)
```

`dayPad` defines a phone keypad visualization of the first decimal digit result.

`PhoneDay` is an algebraic data type that represents days of a week.

For `PhoneDay`, implement a custom instance of the `Enum` type class which uses the “day keypad” for the provided conversion operations. You can implement it using the auxiliary functions:

```
fromEnumCustom :: PhonePad -> PhoneDay -> Int
toEnumCustom :: PhonePad -> Int -> PhoneDay
```

If the read `Int` number is negative, its absolute value should be taken. If the number is larger than the largest button number in the provided `PhonePad`, divide the number by the the largest button number in the provided `PhonePad` and take the integer remainder of that division, e.g. for 10 and `dayPad` => the remainder of integer division 10 / 9 is 1 (i.e. `Monday`).

If everything is implemented correctly, you should be able to use the provided function `dayOfWeek :: Date -> PhoneDay` to get a correct day of a week between years 1901 and 2099.

```
> dayOfWeek (2000, 4, 15)
Saturday
```

NOTE: Feel free to use either list comprehensions or higher-order functions.
HINT: You may want to use operations defined by other type classes of `PhoneDay`.