CS 330 Homework
Funfun

1 Overview

Your responsibility in this homework is to explore functional programming, pattern matching, and higher-order functions. You will do this in the context of writing several utility functions and a few interactive programs in Haskell.

2 Dependencies

Download the Haskell platform and helper libraries on Ubuntu by entering the following at the command line:

```
sudo apt-get install haskell-platform
cabal install random-shuffle
```

In your source files, you will need to import a few packages:

```
import Data.List(foldl', scanl', group)
import GHC.Exts(sortWith)
import Data.Bits(testBit)
import System.Random.Shuffle(shuffleM)
import System.Environment(getArgs)
```

3 Requirements

To receive credit for this homework, you must satisfy these requirements:

1. Place all files in directory `<YOUR-REPOSITORY>/funfun`.
2. All code must run on a standard Linux machine. If you use another operating system, test your code on a Linux machine before submitting.
3. Complete the functions and files described in the following sections. Define each file's functions in a module of the same name. For example, in places all the functions of `Utilities.hs` in module `Utilities` by placing this line at the top of the file:

   ```
   module Utilities where
   ```

3.1 Utilities.hs

Write in `Utilities.hs` the following:

- Function `froto` to generate the consecutive pairs spanning a list. Accept any list and return a list of pairs. For example, `froto "ABCD"` → `[('A','B'),('B','C'),('C','D')]`. If a list contains fewer than 2 elements, return the empty list.

- Function `frotoByNearness` to generate a sorted list of consecutive pairs spanning a list of `Integers`. Write your function in point-free style by composing your `froto` function and the `sortWith` function. Sort such that the pair whose absolute difference is smallest appears first in the list. For example, `frotoByNearness [1, 5, -3, 2]` → `[(1,5),(-3,2),(5,-3)]`.

- Function `nearestPair` to yield from the list of `Integer`s given as a parameter the consecutive pair whose elements are closest. Use composition and implement your function in point-free style. For example, `frotoByNearness [1, 5, -3, 2]` → `(1,5)`.

- Function `mapButLast` that accepts a transformation function and a list, much like the builtin `map`, but it generates the new list by applying the function to all elements but the last. For example, `mapButLast (+1) [1..5]` → `[2, 3, 4, 5, 5]`.

- Function `implode` that accepts a separator `String` and a list of `Show`able items. It concatenates the items together with intervening separators. For example, `implode "," [1..3]` → `"1,2,3"`. Use function composition, `mapButLast`, and partial function application to write this in nearly point-free style. (You should only need to name the first parameter.)

- Function `normspace` that accepts a `String` parameter and returns a `String` like the parameter, but with all sequences of multiple spaces character reduced to one space character. For example, `normspace "the␣wee␣␣dog␣␣␣yipped"` → `"the␣wee␣dog␣yipped"`. Use function composition and the `group` function to write this in point-free style.

- Datatype `Direction` with nullary constructors `North`, `South`, `East`, and `West`. Make it ‘Show‘able using the ‘deriving‘ clause.

- Function `move1` that accepts an `Int` xy-coordinate pair and a `Direction` as parameters. Return a new pair offset from the parameter by the given `Direction`. For example, `move1 (10, 15) North` → `(10, 16)`. Use pattern matching on the `Direction` to simplify your implementation.

- Function `moveN` that accepts an `Int` xy-coordinate pair and a list of `Direction`s as parameters. Return a new pair representing the location that is reached by moving in the directions in the order they are given. Use `foldl'`. Write this in point-free style.

- Function `intermoves` that accepts an `Int` xy-coordinate pair and a list of `Direction`s as parameters. Return a list of pairs representing the locations that are reached by moving in the directions in the order they are given. Include the initial location. Use `scanl'`. Write this in point-free style.

3.2 Intervals.hs

Suppose you have a list of intervals representing ranges of integers that have been outlawed. The magnitude of the ranges is expansive, and some of the intervals overlap. How might one first the smallest number that is *free*, that has not been outlawed, without trying every possible number? Write these functions in file `Intervals.hs` to find such a number:

- Function `within` that accepts an `Integer` and an interval as a pair of `Integers`. It returns a `Bool`, which is `True` if and only if the first parameter lies inclusively within the interval. For example, `within 5 (3, 17)` → `True` and `within 998 (200, 300)` → `False`.

- Function `withins` that accepts an `Integer` and a list of intervals. It yields a list of all the intervals that contain the given number. Write this in nearly point-free style.
- Function `isFree` that accepts an `Integer` and a list of intervals. It yields a `Bool`, which is `True` if and only if the number doesn't fall within any of the given intervals. Write this in nearly point-free style.
- Function `freeBeyond` that accepts a list of intervals. It returns the first `Integer` that is beyond or greater than all of the intervals. Use function composition and write this in point-free style.
- Function `firstFree` that accepts an `Integer` and a list of intervals. It returns the smallest `Integer` greater than or equal to the parameter `Integer` that is not within any of the intervals.

3.3 MindReader.hs

Here's a trick a for you to amaze your friends. Imagine I'm dreaming up a number (it's 22, but you don't know that yet). You ask me the following questions:

```
What's your favorite upper bound? 31

Think of a number in [1, 31]. But don't tell me. I will read your mind!

[1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31]
Is your number in this list? y/[n]? n

[2,3,6,7,10,11,14,15,18,19,22,23,26,27,30,31]
Is your number in this list? y/[n]? y

[4,5,6,7,12,13,14,15,20,21,22,23,28,29,30,31]
Is your number in this list? y/[n]? y

[8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31]
Is your number in this list? y/[n]? n

[16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31]
Is your number in this list? y/[n]? y

Your number is 22.
```

Pay particular attention to the lists to which I responded yes. Their first elements were 2, 4, and 16. Observe that $2 + 4 + 16 = 22$. The magic here is that each list corresponds to a single binary digit. All the numbers in the first list have the binary representation `****1`, all in the second list have `***1*`, all in the third have `**1**`, all in the fourth have `*1***`, and all in the fifth have `1****`. A `y` response for a list enables the corresponding bit, while a `n` response disables the bit. Now, write the following functions in `MindReader.hs` to perform this trick for any range of numbers.

- Function `intsWithBit` that accepts two `Int`s: a maximum and a zero-based, little endian bit index. It yields a list of all `Int`s from 1 through the maximum that have the given bit enabled. For example, `intsWithBit 10 0` $\rightarrow$ `[1,3,5,7,9]` and `intsWithBit 10 1` $\rightarrow$ `[2,3,6,7,10]`.

- Function `binaryToDecimal` that accepts a `String` of 0s and 1s and yields the corresponding `Int`. For example, `binaryToDecimal "011001"` → 25. In an imperative language, we might write the conversion algorithm as follows:

  ```
  sum = 0
  for each bit, most significant to least
    sum *= 2
    if bit is enabled
      sum += 1
  return sum
  ```

  Haskell is not an imperative language, however. This is a job for `foldl'`. Use it.

- Function `decimalToBinary'` that accepts an `Int` and gives back its binary `String` representation, but in reverse. It also doesn't work for 0. Why this strangeness? Really, this function is just a helper. It's primary caller will be `decimalToBinary`, which will clean up its inadequacies. In an imperative language, we might write the conversion algorithm as follows:

  ```
  str = ""
  while n > 0
    if n is even
      str = "0" + str
    else (n must be odd)
      str = "1" + str
    n /= 2
  ```

  Reframe this algorithm recursively. When fed 0, return the empty `String`. Use guards to delineate your cases.

- Function `decimalToBinary` that accepts an `Int` (assumed positive) and gives back its proper binary `String`. If fed 0, return `"0"`. If fed anything else, return the reverse of the value yielded by `decimalToBinary'`.

- Function `nbits` that accepts an `Int` and yields the number of digits in its binary representation. For example, `nbits 100` → 7 and `nbits 8` → 4.

- Function `promptForBit` that accepts a maximum `Int` and an `Int` bit index. It returns an `IO Int`. It prints a blank line, prints the list of numbers with the given bit set, prompts the user to enter `y` or `n`, reads the user's input, and returns an 1 if the answer is `"y"` and 0 otherwise.

- Function `main` that returns an `IO ()`. It prints the upper bound prompt, reads the user's input, prints a blank line, prompts the user to think of a number in the specified range, prompts the player with the lists for each possible bit (use `mapM` to apply `promptForBit` to each possible index), assembles the responses into a binary string, prints a blank line, prints the guess message, and finally returns `()`.

3.4 BullsAndCows.hs

Let's consider another guess-a-number game: Bulls and Cows. You may have played Mastermind, which is similar. This time the computer picks the number and the player figures it out through the computer's feedback. Suppose the target number has four digits. After each guess, the player is told how many digits in the guess match the corresponding digits in the target. These are called *bulls*. For example, suppose the target is 1826 and the guess is 1872. This guess has 2 bulls. The player is also told how many digits in the guess do appear in the target, but are in the wrong spot. These are called *cows*. Our example guess contains one cow, the 2.

Consider this example run, which models what your output should look like:

```
Guess: 1267
Bulls: 0 | Cows: 0
Guess: 1269
Bulls: 1 | Cows: 0
Guess: 1289
Bulls: 1 | Cows: 0
Guess: 1239
Bulls: 1 | Cows: 1
Guess: 3129
Bulls: 1 | Cows: 1
Guess: 1329
Bulls: 2 | Cows: 0
Guess: 5329
Bulls: 2 | Cows: 0
Guess: 4329
Bulls: 2 | Cows: 1
Guess: 0349
Bulls: 4 | Cows: 0
```

Write the following functions to make this game happen:

- Function `bulls` that accepts the target and the guess as `String`s. (We use `String` rather than `Integer`, as leading 0s are significant in this game.) It returns a `String` of all the bulls in the guess, with their relative order preserved. For example, `bulls "1826" "1872"` $\rightarrow$ `"18"`. Use a one-line implementation that `zip`s the two parameter lists.

- Function `nonbulls` that accepts the target and the guess as `String`s. It returns a pair of `String`s. The second item of the pair is the `String` of digits in the guess that are not, with their relative order preserved. The first item is the `String` of corresponding digits in the target. For example, `nonbulls "1826" "1872"` $\rightarrow$ `("26", "72")`. Use a one-line implementation that initially `zip`s the two parameter lists, but `unzip`s them to produce the final result.

- Function `cows` that accepts the target and the guess as `String`s. It returns a `String` of all the cows in the guess, with their relative order preserved. For example, `cows "1826" "1872"` $\rightarrow$ `"2"`. Use functions `filter`, `elem`, and `nonbulls` to solve this.

- Function `score` that accepts the target and the guess as `String`s. It returns the number of bulls and cows in the guess as a pair of `Int`s.

- Function `game` that accepts the target number as a `String` and returns an `IO ()`. It prompts the player for a guess, reads the guess, scores it, and displays the feedback as shown in the example interaction above. If the score reports that every digit is a bull, return `()`. Otherwise, run another round of `game`.
- Function `randomTarget` that accepts an `Int` length and returns an `IO String`. Use `shuffleM` to shuffle the `String` of digits 0 through 9, and return the substring of the result to yield a `String` of the given length.
- Function `main` that generates a random target of the length specified as the first command-line parameter (accessed through `getArgs`) and runs a game with this target.

4 Later Week

To be eligible for later-week submission, you must successfully complete the functions of the `Utilities` module.

5 Submission

To submit your work for grading:

1. Run the grading script from your homework directory using `../specs/grade`.
2. Commit and push your work to your repository.
3. Verify that your solution is on Bitbucket by viewing your repository in a web browser.

A passing grading script does not guarantee you credit. Your grade is conditioned on a few things:

- You must meet the requirements described above. The grading script checks some of them, but not all.
- You must successfully submit your code to your repository. Expect to have issues with Git.
- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. Your instructor employs a vast repertoire of tools to sniff out academic dishonesty, including: drones, moles, and a piece of software called MOSS that rigorously compares your code to every other submission. You don't want to live in a world serviced by those who squeaked by through questionable means. For your future self, career, and family, do your own work.

The grading script allows you to signal your instructor when requirements are met. You only need to send an email if you qualified for later week submission and are resubmitting after the original deadline.