

CSIS2258/COMP3258 Functional Programming 2014/15

Assignment 1 (Deadline: 11:59 AM HKT, February 27)

Exercise 1 (1pt)



Write a function `ryanGosling :: Integer -> String` that given an integer number, it returns a string as follows:

- 1) If the number is divisible by 3, it returns "Ryan".
- 2) If the number is divisible by 5, it returns "Gosling".
- 3) If the number is divisible by both 3 and 5, it returns "Ryan Gosling".
- 4) In other cases, it returns the string representation of the given number's *absolute value*.

e.g.

```
*Assignment1> ryanGosling 3
"Ryan"
*Assignment1> ryanGosling 7
"7"
*Assignment1> ryanGosling 15
"Ryan Gosling"
```


Exercise 2 (1pt)

Write a QuickCheck property function `prop_ryanGosling :: Integer -> Bool` that tests the function implemented in Exercise 1 as follows: for any integer, the `ryanGosling` function returns the same result as when applied with *the negated value* of that integer.

Exercise 3 (1pt)

Write a recursive function `recRyanGosling :: [Integer] -> String` that takes a list of integers and returns a string composed follows:

- 1) Every integer in the list is transformed using the `ryanGosling` function defined in Exercise 1.
- 2) Each of these transformed integers are on a newline.
- 3) The string ends with the following substring: "will not eat cereal".

e.g. (you can use `putStrLn` for a nicer printout) 

```
*Assignment1> recRyanGosling [1..10]
"1\n2\nRyan\n4\nGosling\nRyan\n7\n8\nRyan\nGosling\nwill not eat cereal"
```

Exercise 4 (1pt)

Write a QuickCheck property function `prop_recRyanGosling :: Integer -> [Integer] -> Property` that tests the function implemented in Exercise 3 as follows: for any **positive** integer **N** and list of integers **L**, the output of the function `recRyanGosling` applied with **L** is the same as when applied with **L** where the first **N** integers are negated.

e.g. for 3 and [1,2,3,4,5]

```
recRyanGosling [1,2,3,4,5] == recRyanGosling [-1,-2,-3,4,5]
```

HINT: You may want to define an auxiliary recursive helper function.

Exercise 5 (2pt)



Write a function `bracketBalancer :: String -> Bool` that takes a string and checks whether all round brackets are **matched** and **closed** in it.

e.g.

```
*Assignment1> bracketBalancer "(This should return True.)"
True
*Assignment1> bracketBalancer "(This should return False: :-)"
False
*Assignment1> bracketBalancer "(This should also return False: :-)" (-:"
False
```

HINT: You may want to define auxiliary helper functions (some of them may be recursive).

Exercise 6 (3pt)

Write a function `waysToReturnMoney :: Integer -> [Integer] -> Integer` that takes an amount of money to return (e.g. HKD4), the list of different denominations (e.g. HKD1 and HKD2 coins) and gives the number of different ways you can return the given amount of money with those denominations. For the amount of 0, assume there is 1 way to return the money. Also, assume that the amount will not be negative and that the list of denominations contains unique and positive money denominations.

e.g. for the amount HKD4, with HKD1 and HKD2 coins, you have 3 distinct ways:

- 1) HKD1, HKD1, HKD1, HKD1
- 2) HKD2, HKD2
- 3) HKD2, HKD1, HKD1

```
*Assignment1> waysToReturnMoney 4 [1,2]
3
```

HINT: Think of the base cases. There are 2 recursive calls. You may want to define auxiliary helper functions.

NOTE: You may find that for large numbers and longer lists, the computation takes a very long time. If you like to experiment more, you can also implement an optimized version

`waysToReturnMoneyDP` which utilizes dynamic programming to speed up the computation.

Code Correctness (1pt)

Make sure that your code compiles, produces no warnings with GHC 7.8.3, all your defined QuickCheck properties are satisfied, that recursive functions terminate and that your definitions are using **plain recursive functions** (and not list comprehension or library higher-order functions). Please include any extra code you wrote to test your code.

Code Style (1pt)

Your code should be well-written (e.g. sensible names of functions or variables where appropriate), documented, and produce no warnings with HLint (apart from possibly the ones suggesting the use of library higher-order functions).

Submit your solution on Moodle in one file (use the attached `Assignment1.hs` skeleton).

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.