UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

Date: Tuesday 27th October 2015
Duration: 35 minutes

INFORMATICS 1 — FUNCTIONAL PROGRAMMING CLASS TEST

INSTRUCTIONS TO CANDIDATES

- **ALL QUESTIONS ARE COMPULSORY.**
- **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS.** Take note of this in allocating time to questions.
- WRITE YOUR ANSWERS ON THE EXAM PAPER ITSELF. Write as legibly as possible.
- In the answer to any part of any question, you may use any function specified in an earlier part of that question. You may do this whether or not you actually provided a definition for the earlier part; nor will you be penalized in a later part if your answer to an earlier part is incorrect.
- Unless otherwise stated, you may use any function from the standard prelude, including the libraries Char, List, and Maybe. You need not write import declarations.
- As an aid to memory, some functions from the standard prelude that you may wish to use are listed on the next page. You need not use all the functions.

PLEASE INSERT YOUR NAME AND MATRICULATION NUMBER IN THE SPACE BELOW:

MATRICULATION NUMBER	NAME

```
div, mod :: Integral a => a -> a -> a
even, odd :: Integral a => a -> Bool
(+), (*), (-), (/) :: Num a => a -> a -> a
(<), (<=), (>), (>=) :: Ord => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
max, min :: Ord a => a -> a -> a
isAlpha, isAlphaNum, isLower, isUpper, isDigit :: Char -> Bool
toLower, toUpper :: Char -> Char
ord :: Char -> Int
chr :: Int -> Char
```

Figure 1: Basic functions

```
sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24

maximum, minimum :: (Ord a) => [a] -> a
maximum [3,1,4,2]  =  4
minimum [3,1,4,2]  =  1

concat :: [[a]] -> [a]
concat ["go","od","bye"]  =  "goodbye"

(!!) :: [a] -> Int -> a
[9,7,5] !! 1  =  7

head :: [a] -> a
head "goodbye" = 'g'

init :: [a] -> [a]
init "goodbye" = "goodby"

takeWhile :: (a->Bool) -> [a] -> [a]
takeWhile isLower "goodBye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]
dropWhile isLower "goodBye" = "Bye"

elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye" = True

and, or :: [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True

reverse :: [a] -> [a]
reverse "goodbye" = "eybdoog"

(++) :: [a] -> [a] -> [a]
"good" ++ "bye" = "goodbye"

length :: [a] -> Int
length [9,7,5]  =  3

tail :: [a] -> [a]
tail "goodbye" = "oodbye"

last :: [a] -> a
last "goodbye" = 'e'

take :: Int -> [a] -> [a]
take 4 "goodbye" = "good"

drop :: Int -> [a] -> [a]
drop 4 "goodbye" = "bye"

replicate :: Int -> a -> [a]
replicate 5 '*' = "*****"

zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3,4] [1,4,9] = [(1,1),(2,4),(3,9)]
```

Figure 2: Library functions

1. (a) Define a function `count :: String -> Int` that counts the number of characters in a string that are either upper case or digits. For example:

```
count "Inf1-FP" = 4        count "" = 0
count "none here" = 0      count "5HOU7" = 5
```

Your definition may use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

[*20 marks*]

(b) Define another function, `countRec :: String -> Int`, that behaves identically to `count`, this time using *basic functions* and *recursion*, but not list comprehension or library functions.

[*20 marks*]

(c) Write a QuickCheck property `prop_count` to confirm that `count` and `countRec` behave identically. Give the type signature of `prop_count` and its definition.

[*5 marks*]

2. Take any integer $n > 0$. If n is even, divide it by 2. If n is odd, multiply it by 3 and add 1. Repeat. The *Collatz conjecture* (which has several other names) is that no matter what number you start with, you will eventually reach 1.

   This question is about testing lists of integers to see if they obey the rules above, ignoring the question of whether or not the sequence eventually reaches 1.

   (a) Define a function `isNext :: Int -> Int -> Bool` that, given two numbers, checks to see if the second number is the one that comes immediately after the first number according to the rules above. For example:

```
isNext 4 2 = True          isNext 8 3 = False
isNext 5 16 = True         isNext 3 12 = False
```

   [*15 marks*]

   (b) Using `isNext`, define a function `collatz :: [Int] -> Bool` that checks if each pair of consecutive numbers in a list follows the rules above. For example:

```
collatz [22,11,34,17,52,26,13,40] = True      collatz [] = True
collatz [21,64,32,8,4] = False          collatz [4,2,1,4] = True
```

   Your definition may use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

   [*20 marks*]

   (c) Define another function `collatzRec :: [Int] -> Bool` that behaves identically to `collatz`, this time using *basic functions* and *recursion*, but not list comprehension or library functions.

   [*20 marks*]