UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

Date: Monday 24th October 2016
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 define any number of helper functions and use any function from the standard prelude, including the libraries Char and List. 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        and, or :: [Bool] -> Bool
sum [1.0,2.0,3.0] = 6.0                    and [True,False,True] = False
product [1,2,3,4] = 24                     or [True,False,True] = True

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

concat :: [[a]] -> [a]                     (++) :: [a] -> [a] -> [a]
concat ["go","od","bye"]  =  "goodbye"     "good" ++ "bye" = "goodbye"

(!!) :: [a] -> Int -> a                    length :: [a] -> Int
[9,7,5] !! 1  =  7                         length [9,7,5]  =  3

head :: [a] -> a                           tail :: [a] -> [a]
head "goodbye" = ’g’                       tail "goodbye" = "oodbye"

init :: [a] -> [a]                         last :: [a] -> a
init "goodbye" = "goodby"                  last "goodbye" = ’e’

takeWhile :: (a->Bool) -> [a] -> [a]       take :: Int -> [a] -> [a]
takeWhile isLower "goodBye" = "good"       take 4 "goodbye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]       drop :: Int -> [a] -> [a]
dropWhile isLower "goodBye" = "Bye"        drop 4 "goodbye" = "bye"

elem :: (Eq a) => a -> [a] -> Bool         replicate :: Int -> a -> [a]
elem ’d’ "goodbye" = True                  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) Let's call a number "vowelly" if it is less than 100 and starts with a vowel when spelled out in English. Define a function `vowelly :: Int -> Bool` that returns `True` for vowelly numbers and `False` for all other numbers. For example:

```
vowelly 6 = False (six)          vowelly 11 = True (eleven)
vowelly 15 = False (fifteen)     vowelly 83 = True (eighty-three)
```

Hint: There are only 14 vowelly numbers. Don't write a function that converts numbers to English and then tests if the first character is a vowel!

[*15 marks*]

(b) Using `vowelly`, define a function `count :: [Int] -> Int` that counts how many vowelly numbers a list contains. For example:

```
count [22,11,34,17,52,26,13,40] = 1      count [] = 0
count [21,64,32,11,4] = 1                count [8,83,4,8] = 3
```

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

[*20 marks*]

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

[*20 marks*]

2. (a) Write a function `c :: Char -> String -> String` that replaces every second character in a string, starting with the first, with the given character.

The result string should have the same length as the argument string and should begin with the given character, followed by the second character of the string, followed by the given character, followed by the fourth character of the string, followed again by the given character, and so on. For example:

```
c '.' "abcdefg"    =   ".b.d.f."
c '.' "abcd"       =   ".b.d"
c '.' ""           =   ""
c '.' "a"          =   "."
```

Use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

[*20 marks*]

(b) Write a second function `d :: Char -> String -> String` that behaves identically to `c`, this time using *basic functions* and *recursion*, but not list comprehension or other library functions.

[*20 marks*]

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

[*5 marks*]