UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

Date: Monday 23rd October 2017
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
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) Write a function `f :: [Int] -> Int` that computes the sum of the squares of those numbers in a list that are divisible by 3 but not by 5. For example:

```
f [] = 0
f [9,-3] = 90
f [0,30,2,7] = 0
f [-6,15,2,1,3] = 45
```

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

[*20 marks*]

(b) Write a second function `g :: [Int] -> Int` that behaves identically to `f`, this time using *basic functions* and *recursion*, but not list comprehension or other library functions.

[*20 marks*]

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

[*5 marks*]

2. (a) We say that an integer x is *much smaller than* an integer y if either $x \geq 0$ and y is more than twice as large as x, or $x < 0$ and y is larger than $x/2$.

Define a function `mst :: Int -> Int -> Bool` that returns `True` if its first argument is much smaller than its second argument and `False` otherwise. For example:

```
mst (-10) (-5) == False          mst 7 14   == False
mst (-10) (-4) == True           mst 7 15   == True
mst (-2) 3     == True
```

[*15 marks*]

(b) Define a function `ordered :: [Int] -> Bool` that returns `True` if the integers in its argument list are in ascending order according to `mst`, and `False` otherwise. For example:

```
ordered [] = True
ordered [-4,-1,3,1,9] = False
ordered [-4,-1,1,3,9] = True
ordered [-4,-1,1,2,9] = False
```

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

[*20 marks*]

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

[*20 marks*]