UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 — FUNCTIONAL PROGRAMMING CLASS TEST

due 4pm Wednesday 4 November 2020

INSTRUCTIONS TO CANDIDATES

- 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.

Unlike tutorials, you should not consult other students or Piazza when answering the class exam, and CodeGrade is not available to help you check your answers.

Good Scholarly Practice: Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

```
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 :: String -> Bool` that returns true if given a string where every alphabetic character is upper case. For example:

```
f "" == True
f "..." == True
f "nope" == False
f "FAKE NEWS" == True
f "This is a Tweet." == False
f "THIS IS A TWEET!" == True
```

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

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

   (c) Write a QuickCheck property `prop_fg` to confirm that `f` and `g` behave identically.

2. (a) Write a function `c :: String -> Bool` that takes a string and returns true if the string contains two adjacent characters that are identical.

```
c "" == False
c "s" == False
c "ss" == True
c "Misisipi" == False
c "Mississippi" == True
c "single-dash" == False
c "double--dash" == True
```

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

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

   (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.