UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 — FUNCTIONAL PROGRAMMING CLASS EXAM

due 4pm Tuesday 25 October 2022

INSTRUCTIONS TO CANDIDATES

- Unless otherwise stated, you may define any number of helper functions. 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. The functions are labeled as *basic functions* and *library functions*, which is important in some questions.

Unlike other tutorial coursework, you should not consult other students or Piazza when answering the class exam.

You will not receive credit for your coursework unless you attend the corresponding tutorial session. Please email kendal.reid@ed.ac.uk if you cannot join your assigned tutorial.

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 Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.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

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

```
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 '*' = "*****"
```

Figure 2: Library functions

1. (a) Write a function `f :: String -> Int` that returns the sum of the character codes (computed using `ord :: Char -> Int`) of the letters in string, ignoring non-letters. Use `isAlpha :: Char -> Bool` to determine if a character is a letter or not. For example:

```
f "" == 0
f "Ha5k3ll 15 gr8" == 709
f "allLetters" == 1052
f "7*6 = 42" == 0
f "1 More Example!!!" == 1119
```

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

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

(c) Write a third function `h :: String -> Int` that behaves identically to `f`, this time using *basic functions* and *higher-order functions*.

(d) Write a QuickCheck property `prop_fgh` to confirm that `f`, `g`, and `h` behave identically.

2. (a) Write a function `c :: String -> String -> Bool` that takes two strings and returns `True` if the characters in corresponding positions in the strings are the same, when both are letters.


```
c "Julian" "Lon" == False
c "7*6" "5!" == True
c "potat0e5" "p*tatoes" == True
c "" "p*tat+e!" == True
c "ptate" "p*tat+e!" == False
```


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 library functions.

(c) Write a QuickCheck property `prop_cd` to confirm that `c` and `d` behave identically.