UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Tuesday 25 August 2009

09:30 to 11:30

Convener: M O'Boyle
External Examiner: R Irving

INSTRUCTIONS TO CANDIDATES

1. **Note that ALL QUESTIONS ARE COMPULSORY.**
2. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.**

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

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.

As an aid to memory, basic functions are listed in Figure 1 and library functions are listed in Figure 2. You will not need all the functions listed.

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

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"

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

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

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'

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

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

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

Figure 2: Library functions

1. (a) Write a function `f :: String -> Bool` that takes a string, and returns true if every digit in the string is even. Any characters in the string that are not digits should be ignored. For example,

```
f "246"       ==  True
f "2467"      ==  False
f "x4y2z"     ==  True
f "abc12"     ==  False
```

Your definition may use basic functions, *list comprehension*, and *library functions*, but not recursion. [*12 marks*]

(b) Write a second function `g :: String -> Bool` that behaves like `f`, this time using basic functions and *recursion*, but not list comprehension or other library functions. [*12 marks*]

(c) Write a third function `h :: String -> Bool` that also behaves like `f`, this time using one or more of the following higher-order library functions.

```
map    :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr  :: (a -> b -> b) -> b -> [a] -> b
```

You may use basic functions, but not list comprehensions, other library functions, or recursion. [*12 marks*]

2. (a) Write a function `p :: Int -> Int -> Int` that takes two integers and counts the even numbers between them. For example,

```
p 3 7  ==  2
p 7 8  ==  0
p 8 0  ==  3
```

The first returns 2 because 4,5,6 are the numbers between 3 and 7, and 4 and 6 are even. The second returns 0 because there are no numbers between 7 and 8. The third returns 3 because 7,6,5,4,3,2,1 are the numbers between 8 and 0, and 6, 4, and 2 are even. [*12 marks*]

(b) Using `p`, write a function `r :: [Int] -> Int` that takes a list of at least two integers, in which no two adjacent elements are equal, and returns the largest count of even numbers between any two adjacent elements. For example,

```
r [3,7,8,0]  ==  3
```

This returns 3 because there are 2 even numbers between 3 and 7, none between 7 and 8, and 3 between 8 and 0, and 3 is the largest of 2, 0, and 3. Your definition may use basic functions, *list comprehension* and *library functions*, but not recursion. [*12 marks*]

(c) Again using `p`, write a second function `s :: [Int] -> Int` that behaves like `r`, this time using basic functions and *recursion*, but not list comprehension or other library functions. [*12 marks*]

3. (a) Write a function `t :: String -> Int` that takes a list of digits and returns the corresponding numerical value of the list in reverse order. For example,

```
t "1234"  ==  4321
t "526"   ==  625
```

Your definition may use basic functions, *list comprehension*, and *library functions*, but not recursion. Do *not* use the library function `read`. [*14 marks*]

(b) Write a second function `u :: String -> Int` that behaves like `t`, this time using basic functions and *recursion*, but not list comprehension or library functions. You may wish to use auxiliary functions in your definition. [*14 marks*]