UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Monday 8 December 2008

14:30 to 16:30

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
(<), (<=), (>), (>=) :: 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
```

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 :: [Int] -> Int` that takes a list of integers, and returns the largest integer in the list between `0` and `100` (inclusive). The function should return `0` if the list is empty or there are no numbers between `0` and `100`. For example,

```
f [40,50,60]         ==  60
f [40,30,50,70,60]   ==  70
f [-10,20,80,110]    ==  80
f [-10,110]          ==   0
f []                 ==   0
```

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

(b) Write a second function `g :: [Int] -> Int` 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 :: [Int] -> Int` 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 distinct numbers and counts up or down from the first to the second. The list returned should include the first number but exclude the second. For example,

```
p 3 7  ==  [3,4,5,6]
p 7 3  ==  [7,6,5,4]
```

[*12 marks*]

(b) Using `p`, write a function `r :: [Int] -> [Int]` that takes a non-empty list of integers, in which no two adjacent elements are equal, and counts up or down between successive numbers in the input, including the first number but excluding the last. For example,

```
r [3,7,4,8]       ==  [3,4,5,6,7,6,5,4,5,6,7]
r [1,3,5,3,5,7]   ==  [1,2,3,4,5,4,3,4,5,6]
```

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, the function append `(++)`, and *recursion*, but not list comprehension or other library functions. [*12 marks*]

3. (a) Write a function `t :: [Int] -> Int` that takes a list of integers and returns the number of runs of zeros in the list, where a run consists of one or more successive zeros. For example,

```
t []                 ==  0
t [0,0,0]            ==  1
t [0,0,1,1,0,0]      ==  2
t [1,0,0,0,0,1]      ==  1
t [1,1,0,0,2,2,0,0]  ==  2
```

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

(b) Write a second function `u :: [Int] -> 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*]