UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1A: FUNCTIONAL PROGRAMMING

Thursday 21 August 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, arithmetic and comparison operators are listed in Figure 1 and some 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
```

Figure 1: Arithmetic, comparison, and logic

```
sum, product :: (Num a) => [a] -> a         and, or :: [Bool] -> Bool
sum [1.0,2.0,3.0] = 6.0                     and [True,False,True] = False
product [1,2,3,4] = 24                      or [True,False,True] = True

(:) :: a -> [a] -> [a]                      (++) :: [a] -> [a] -> [a]
'g' : "oodbye" = "goodbye"                  "good" ++ "bye" = "goodbye"

(!!) :: [a] -> Int -> a                     length :: [a] -> Int
[9,7,5] !! 1  =  7                          length [9,7,5]  =  3

head :: [a] -> a                            tail :: [a] -> [a]
head "goodbye" = 'g'                        tail "goodbye" = "oodbye"

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

splitAt :: Int -> [a] -> ([a],[a])          reverse :: [a] -> [a]
splitAt 4 "goodbye" = ("good","bye")        reverse "goodbye" = "eybdoog"

elem :: (Eq a) => a -> [a] -> Bool          replicate :: Int -> a -> [a]
elem 'd' "goodbye" = True                   replicate 5 '*' = "*****"

concat :: [[a]] -> [a]
concat ["con","cat","en","ate"] = "concatenate"

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

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

Figure 2: Some library functions

1. (a) Write a function `f :: [Int] -> Bool` that takes a list of integers and returns true if every positive number in the list is even, and false otherwise. For example,

```
f [2,10,8]   returns  True,   and
f [2,9,8]    returns  False,  and
f [2,-9,8]   returns  True.
```

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

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

(c) Write a third function `h :: [Int] -> Bool` that also behaves like `f`, this time using 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 arithmetic, comparison, and logic, but do not use list comprehension, recursion, or any other library functions. [*12 marks*]

2. (a) Write a function `p :: Int -> Int -> Int` that returns the absolute value of the difference of its two arguments. For example,

```
p 1 2   returns  1,   and
p 2 4   returns  2,   and
p 4 7   returns  3,   and
p 7 3   returns  4,   and
p 3 8   returns  5.
```

Your definition may use arithmetic, comparison, and logic, but not library functions, list comprehension, or recursion. [*12 marks*]

(b) Using `p`, write a function `q :: [Int] -> Int` that takes a non-empty list of integers, and returns the sum of the absolute value of the difference of each pair of adjacent numbers in the list. For example,

```
q [1,2,4,7,3,8]  returns  15,  and
q [8,3,7,4,2,1]  returns  15,  and
q [1,2,3,4,5,6]  returns  5,   and
q [6,5,4,3,2,1]  returns  5,   and
q [3,3,3,3,3,3]  returns  0.
```

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

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

3. (a) Write a function `t :: Int -> Int -> [a] -> [a]` that takes two indexes `i` and `j` and a list, and returns a list of all elements in the list between index `i` (inclusive) and index `j` (exclusive), where indexes count from zero. You may assume that `i` is not smaller than zero, that `j` is not smaller than `i`, and that the length of the list is not smaller than `j`. For example,

```
t 0 6 "abcdef"  returns  "abcdef",  and
t 1 5 "abcdef"  returns  "bcde",    and
t 2 4 "abcdef"  returns  "cd",      and
t 3 3 "abcdef"  returns  "".
```

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

(b) Write a second function `u :: Int -> Int -> [a] -> [a]` that behaves like `t`, this time using *recursion*, arithmetic, comparison, logic, and list constructors `(:)` and `[]`, but not comprehensions or other library functions. Try to avoid using auxiliary functions in your definition of `u`. [*14 marks*]