UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08013 INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Tuesday 11 th August 2015

14:30 to 16:30

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.**
3. **This is an OPEN BOOK examination: notes and printed material are allowed, and USB sticks, but no electronic devices.**
4. **CALCULATORS MAY NOT BE USED IN THIS EXAMINATION**

Convener: D. K. Arvind
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Write a function `f :: [a] -> [a] -> [a]` that *interleaves* two lists. The result should start with the first item of the first list, then the first item of the second list, then the second item of the first list, and so on. If one list is longer than the other, the remainder of the longer list should be discarded. For example:

```
f "itrev" "nelae"        = "interleave"
f "arp" "butmore"        = "abrupt"
f [] [1,2,3]             = []
f [1,1,1] [33,11,22,44] = [1,33,1,11,1,22]
```

Use *basic functions*, *list comprehension*, and *library functions*, but *not recursion*. Credit may be given for indicating how you have tested your function.

[*16 marks*]

(b) Write a second function `g :: [a] -> [a] -> [a]` that behaves like `f`, this time using *basic functions* and *recursion*, but *not list comprehension* or *library functions*. Credit may be given for indicating how you have tested your function. [*16 marks*]

2. (a) Write a function `p :: [Int] -> Bool` that checks that the square of every strictly positive number in a list is odd. For example:

```
p [13]               =  True
p []                 =  True
p [-3,3,1,-3,2,-1]   =  False
p [3,7,-3,0,3,-7,5]  =  True
p [4,-2,5,-3]        =  False
```

Use *basic functions*, *list comprehension*, and *library functions*, but *not recursion.* Credit may be given for indicating how you have tested your function.

[*12 marks*]

(b) Write a second function `q :: [Int] -> Bool` that behaves like `p`, this time using *basic functions* and *recursion*, but *not list comprehension* or *library functions.* Credit may be given for indicating how you have tested your function. [*12 marks*]

(c) Write a third function `r :: [Int] -> Bool` that also behaves like `p`, 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
```

Do *not* use *recursion* or *list comprehension.* Credit may be given for indicating how you have tested your function. [*12 marks*]

3. Consider binary trees with `Int`-labelled nodes and leaves, defined as follows:

```
data Tree = Empty
          | Leaf Int
          | Node Tree Int Tree
```

and the following example of a tree:

```
t = Node (Node (Node (Leaf 1)
                     2
                     Empty)
               3
               (Leaf 4))
         5
         (Node Empty
               6
               (Node (Leaf 7)
                     8
                     (Leaf 9)))
```

Each label in a tree can be given an "address": this is the path from the root to the label, consisting of a list of directions:

```
data Direction = L | R

type Path = [Direction]
```

The empty path refers to the label at the root — in `t` above, the label `5`. A path beginning with `L` refers to a label in the left, or first, subtree, and a path beginning with `R` refers to the right, or second, subtree. Subsequent `L`/`R` directions in the list then refer to left/right subtrees of that subtree. So, for example, `[R,R,L]` is the address of the label `7` in `t`.

The template file includes code that enables QuickCheck to generate arbitary values of types `Tree` and `Direction`, to aid testing, and the tree `t` above and `t'` below. A function `present :: Path -> Tree -> Bool`, which returns `True` for paths in a tree that are the addresses of labels, is also provided for use in testing. For example, `present [] t = True` and `present [R,R,L] t = True` but `present [R,L] t = False`.

QUESTION CONTINUES ON NEXT PAGE

(a) Write a function `label :: Path -> Tree -> Int`, which given a path and a tree, returns the label at the address given by the path. For example,

```
label [] t = 5
label [L] t = 3
label [R] t = 6
label [R,R] t = 8
label [R,R,L] t = 7
```

Credit may be given for indicating how you have tested your function. [*8 marks*]

(b) Another way of representing a tree is as a function from paths to labels:

```
type FTree = Path -> Int
```

Write a function `toFTree :: Tree -> FTree` that converts a tree to this form. Credit may be given for indicating how you have tested your function. [*8 marks*]

(c) The mirror image of a tree is obtained by switching left and right subtrees throughout. For instance, the mirror image of `t` above is

```
t' = Node (Node (Node (Leaf 9)
                      8
                      (Leaf 7))
                6
                Empty)
          5
          (Node (Leaf 4)
                3
                (Node Empty
                      2
                      (Leaf 1)))
```

Write a function `mirrorTree :: Tree -> Tree` that produces the mirror image of a tree. Credit may be given for indicating how you have tested your function. [*8 marks*]

(d) Write a function `mirrorFTree :: FTree -> FTree` that produces the mirror image of a tree that is represented as a function. Credit may be given for indicating how you have tested your function. [*8 marks*]