UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Thursday 14 th August 2014

09:30 to 11:30

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.**

Convener: J. Bradfield
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Write a function `f :: String -> String` that repeats each successive character in the input string once more than the previous character, starting with a single occurrence of the first character. For example:

```
f "abcde"   =  "abbcccddddeeeee"
f "ZYw"     =  "ZYYwww"
f ""        =  ""
f "Inf1FP"  =  "Innfff1111FFFFFPPPPPP"
```

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 `g :: String -> String` that behaves like `f`, this time using *basic functions*, *library functions* and *recursion*, but not list comprehension. Credit may be given for indicating how you have tested your function. [*12 marks*]

2. (a) Write a function `p :: [String] -> Int` that, given a list of strings, adds together the lengths of the abbreviations in the list. For the purposes of this question, abbreviations are strings that contain a full stop (`'.'`). For example:

```
p ["Dr.","Who","crossed","the","ave."] = 7
p ["the","sgt.","opened","the","encl.","on","Fri.","pm"] = 13
p [] = 0
p ["no","abbreviations","4U"] = 0
```

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 `q :: [String] -> Int` that behaves like `p`, this time using *basic functions*, *library functions*, and *recursion*, but not list comprehension. Credit may be given for indicating how you have tested your function. [*16 marks*]

(c) Write a third function `r :: [String] -> Int` 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 leaves, defined as follows:

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

The template file includes code that enables QuickCheck to generate arbitary values of type `Tree`, to aid testing.

The *deepest* leaves are the ones that are furthest from the root of the tree. So in the following examples, the deepest leaves are the ones underlined. (These examples are provided in the template file for use in testing.)

```
t1 = Empty

t2 = Node (Leaf 1)
          Empty

t3 = Node (Node (Node (Leaf 3)
                      Empty)
                (Leaf 1))
          (Node Empty
                (Node (Leaf 3)
                      (Leaf 5)))

t4 = Node (Node (Node Empty
                      Empty)
                (Leaf 1))
          (Node Empty
                (Node Empty
                      Empty))
```

Note that all of the deepest leaves in a tree are at the same depth, and that `t1` has no deepest leaves.

QUESTION CONTINUES ON NEXT PAGE

(a) Write a function `leafdepth :: Tree -> Int` that returns the depth of the deepest leaves in a tree. For example, with reference to the examples above:

```
leafdepth t1 = 0
leafdepth t2 = 2
leafdepth t3 = 4
leafdepth t4 = 3
```

The result should be `0` for all trees in which — like `t1` — there are no leaves. Credit may be given for indicating how you have tested your function.

[**Hint:** You will need to treat trees of the form `tree s t` differently if both `s` and `t` contain no leaves.] [*8 marks*]

(b) Write a function `deepest1 :: Tree -> [Int]` that returns a list containing the labels of the deepest leaves in a tree, in the order that they appear. Your function should use your function `leafdepth` to compute the depth of the deepest leaves, and then return the list of labels of all leaves at that depth. With references to the examples above:

```
deepest1 t1 = []
deepest1 t2 = [1]
deepest1 t3 = [3,3,5]
deepest1 t4 = [1]
```

Credit may be given for indicating how you have tested your function. [*12 marks*]

(c) Write a function `deepest2 :: Tree -> [Int]` that returns a list containing the labels of the deepest leaves in a tree, in the order that they appear, using a different method: traverse the tree, only applying `deepest2` recursively to those subtrees whose leaves are deepest, as determined by `leafdepth`. The results should be the same as for `deepest1`. Do not make use of `deepest1` in your solution. Credit may be given for indicating how you have tested your function. [*12 marks*]