UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Monday 15 August 2011

14:30 to 16:30

Convener: J Bradfield
External Examiner: A Preece

INSTRUCTIONS TO CANDIDATES

1. **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 exam.**

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Write a function `f :: [String] -> String` to concatenate together each string that begins with a capital letter in a list of non-empty strings. For example,

```
f ["Once","Upon","a","Time"]  ==  "OnceUponTime"
f ["no","capitals","!"]       ==  ""
f ["ALL","CAPS"]              ==  "ALLCAPS"
f ["ab","Cd","Ef","gh","ij"]  ==  "CdEf"
```

Your definition may 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* and *recursion*, but not list comprehension or other library functions. Credit may be given for indicating how you have tested your function. [*12 marks*]

(c) Write a third function `h :: [String] -> String` 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 also use *basic functions*, but not list comprehension, other library functions, or recursion. Credit may be given for indicating how you have tested your function. [*12 marks*]

2. (a) Write a polymorphic function `p :: [a] -> [a]` that returns every third element in a list, starting with the first. For example:

```
p "abcdefghij"    ==   "adgj"
p [1,2,3,4,5]     ==   [1,4]
p [0,0,0,0,0]     ==   [0,0]
p []              ==   []
```

Your function may 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 :: [a] -> [a]` 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. [*16 marks*]

3. (a) The following data type represents terms with a free variable x. A term is a constant integer, the variable x, or the sum or product of two terms.

```
data Term  =  Con Int
           |  X
           |  Term :+: Term
           |  Term :*: Term
```

Write a function `eva :: Term -> Int -> Int`, which given a term and the value of the variable x returns the value of the term. For example,

```
eva (Con 3) 3                   ==  3
eva (Con 3) 5                   ==  3
eva X 3                         ==  3
eva X 5                         ==  5
eva (X :*: X) 3                 ==  9
eva ((X :*: X) :+: Con 1) 3     ==  10
eva (X :*: (X :+: Con 1)) 3     ==  12
eva ((Con 2 :*: (X :*: X)) :+: ((Con 3 :*: X) :+: Con 4)) 5
                                ==  69
```

Credit may be given for indicating how you have tested your function. [*16 marks*]

(b) Write a function `sho :: Term -> String` that converts a term to a string. Print a constant as itself, print the variable x as `"x"`, print sums and products using `"+"` and `"*"`, and print all parentheses. For example,

```
sho (Con 3)                   ==  "3"
sho (Con 3)                   ==  "3"
sho X                         ==  "x"
sho (X :*: X)                 ==  "(x*x)"
sho ((X :*: X) :+: Con 1)     ==  "((x*x)+1)"
sho (X :*: (X :+: Con 1))     ==  "(x*(x+1))"
sho ((Con 2 :*: (X :*: X)) :+: ((Con 3 :*: X) :+: Con 4))
                              ==  "((2*(x*x))+((3*x)+4))"
```

Credit may be given for indicating how you have tested your function. [*16 marks*]