UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08013 INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Friday 12 th August 2016

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.**
3. **This is an OPEN BOOK examination: notes and printed material are allowed, and USB sticks (read only), 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 :: String -> Int` that converts a list of binary digits to the corresponding numerical value. For example:

```
f "101" = (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 5
f "11" = 3
f "1101" = 13
f "110111" = 55
```

Use *basic functions*, *list comprehension*, and *library functions*, but not recursion. You may assume that the input is a string of binary digits. Credit may be given for indicating how you have tested your function.

[**Hint:** Start by reversing the order of the digits in the list.] [*16 marks*]

(b) Write a second function `g :: String -> Int` 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.

[**Hint:** Again, start by reversing the order of the digits in the list.] [*16 marks*]

2. (a) Write a function `p :: [Int] -> Bool` that checks that every number in a list that is divisible by 3 is odd. For example:

```
p [1,15,153,83,64,9]  =  True
p [1,12,153,83,9]     =  False
p []                  =  True
p [2,151]             =  True
```

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. The following data type represents propositional formulas built from a single variable (`X`), constants true (`T`) and false (`F`), negation (`Not`) and implication (`:->:`):

```
data Prop = X
          | F
          | T
          | Not Prop
          | Prop :->: Prop
```

The template file includes a function `showProp :: Prop -> String` which converts formulas into a readable format, and code that enables QuickCheck to generate arbitrary values of type `Prop`, to aid testing.

(a) Write a function `eval :: Prop -> Bool -> Bool`, which given a propositional formula and the value of the variable `X` returns the value of the formula. Recall the truth tables for negation and implication:

P	`Not` P
F	T
T	F

P	Q	P `:->:` Q
F	F	T
F	T	T
T	F	F
T	T	T

For example,

```
eval (Not T) True                    =  False
eval (Not X) False                   =  True
eval (Not X :->: Not (Not X)) True   =  True
eval (Not X :->: Not (Not X)) False  =  False
eval (Not (Not X :->: F)) True       =  False
eval (Not (Not X :->: F)) False      =  True
```

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

(b) Write a function `simplify :: Prop -> Prop` that converts a propositional formula to an equivalent simpler formula by repeated use of the following laws:

$$\begin{aligned}
\texttt{Not T} &= \texttt{F} \\
\texttt{Not F} &= \texttt{T} \\
\texttt{Not (Not } p\texttt{)} &= p \\
\texttt{T :->: } p &= p \\
\texttt{F :->: } p &= \texttt{T} \\
p \texttt{ :->: T} &= \texttt{T} \\
p \texttt{ :->: F} &= \texttt{Not } p
\end{aligned}$$

QUESTION CONTINUES ON NEXT PAGE

For example,

```
simplify (Not F)                        =  T
simplify (Not X :->: Not (X :->: T))    =  X
simplify (Not (Not X :->: Not T))       =  Not X
simplify (Not (F :->: Not (Not X)))     =  F
```

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