UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

Date: Monday 24th October 2011
Duration: 35 minutes

INFORMATICS 1 — FUNCTIONAL PROGRAMMING CLASS TEST

INSTRUCTIONS TO CANDIDATES

- **Note that ALL QUESTIONS ARE COMPULSORY.**
- **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS.** Take note of this in allocating time to questions.
- WRITE YOUR ANSWERS ON THE EXAM PAPER ITSELF. Write as legibly as possible.
- 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.
- Unless otherwise stated, you may use any function from the standard prelude, including the libraries Char, List, and Maybe. You need not write import declarations.
- As an aid to memory, some functions from the standard prelude that you may wish to use are listed on the next page. You need not use all the functions.

PLEASE INSERT YOUR NAME AND MATRICULATION NUMBER IN THE SPACE BELOW:

MATRICULATION NUMBER	NAME

```
div, mod :: Integral a => a -> a -> a
even, odd :: Integral a => a -> Bool
(+), (*), (-), (/) :: Num a => a -> a -> a
(<), (<=), (>), (>=) :: Ord => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
max, min :: Ord a => a -> a -> a
isAlpha, isAlphaNum, isLower, isUpper, isDigit :: Char -> Bool
toLower, toUpper :: Char -> Char
ord :: Char -> Int
chr :: Int -> Char
```

Figure 1: Basic functions

```
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

maximum, minimum :: (Ord a) => [a] -> a    reverse :: [a] -> [a]
maximum [3,1,4,2]  =  4                    reverse "goodbye" = "eybdoog"
minimum [3,1,4,2]  =  1

concat :: [[a]] -> [a]                     (++) :: [a] -> [a] -> [a]
concat ["go","od","bye"]  =  "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"

init :: [a] -> [a]                         last :: [a] -> a
init "goodbye" = "goodby"                  last "goodbye" = ’e’

takeWhile :: (a->Bool) -> [a] -> [a]       take :: Int -> [a] -> [a]
takeWhile isLower "goodBye" = "good"       take 4 "goodbye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]       drop :: Int -> [a] -> [a]
dropWhile isLower "goodBye" = "Bye"        drop 4 "goodbye" = "bye"

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

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

Figure 2: Library functions

1. (a) Write a function `f :: Char -> Int` that converts a hexadecimal digit to its value. A hexadecimal digit is one of the characters '0' through '9' or 'a' through 'f'. The values of the digits '0' through '9' are the integers 0 through 9, and the value of the letters 'a' through 'f' are the numbers 10 through 15. Upper and lower case letters have the same value. For example,

```
f '0'  ==  0          f 'a'  ==  10          f 'A'  ==  10
f '2'  ==  2          f 'c'  ==  12          f 'C'  ==  12
f '9'  ==  9          f 'f'  ==  15          f 'F'  ==  15
```

For any character that is not a hexadecimal digit, `f` should return an error.

[*25 marks*]

(b) Using `f`, define a function `g :: String -> Int` that given a string returns the maximum value of any hexadecimal digit in the string, ignoring any character that is not a hexadecimal digit If the string contains no digit or letter it should return -1. For example,

```
g  "3142"  ==  4     g "a2cz!"  ==  12      g ""  ==  -1
```

[*15 marks*]

(c) Again using `f`, define a function, `h :: String -> Int` that behaves identically to `g`, this time using *basic functions* and *recursion*, but not list comprehension or library functions.

[*15 marks*]

2. (a) Write a function `c :: [Int] -> Int` that returns the product of the difference of successive elements in the list. If the list has one element the function should return one, and if the list is empty it should indicate an error.

```
c [3,1,4,2,5]   =  (3-1) * (1-4) * (4-2) * (2-5)    =  36
c [2,4,6,8]     =  (2-4) * (4-6) * (6-8)            =  -8
c [1,2,2,1]     =  (1-2) * (2-2) * (2-1)            =  0
c [-1,2,-3,4]   =  ((-1)-2) * (2-(-3)) * ((-3)-4)   =  105
c [42]                                              =  1
c []                                                =  error
```

Your definition may use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

[*20 marks*]

(b) Define a second function, `d :: [Int] -> Int` that behaves identically to `c`, this time using *basic functions* and *recursion*, but not list comprehension or other library functions.

[*20 marks*]

(c) Write a QuickCheck property `prop_cd` to confirm that `c` and `d` behave identically. Give the type signature of `prop_cd` and its definition.

[*5 marks*]