UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

Date: Monday 29th October 2007
Duration: 35 minutes

INFORMATICS 1A
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
(+), (*), (-), (/) :: Num a => a -> a -> a
(<), (<=), (>), (>=) :: Ord => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
```

Figure 1: Arithmetic, comparison, and logic

```
isAlpha :: Char -> Bool
isAlpha 'a' = True    isAlpha '1' = False
isAlpha 'A' = True    isAlpha '.' = False

sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24

and, or :: [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True

(:) :: a -> [a] -> [a]
'g' : "oodbye" = "goodbye"

(++) :: [a] -> [a] -> [a]
"good" ++ "bye" = "goodbye"

(!!) :: [a] -> Int -> a
[9,7,5] !! 1  =  7

length :: [a] -> Int
length [9,7,5]  =  3

head :: [a] -> a
head "goodbye" = 'g'

tail :: [a] -> [a]
tail "goodbye" = "oodbye"

take :: Int -> [a] -> [a]
take 4 "goodbye" = "good"

drop :: Int -> [a] -> [a]
drop 4 "goodbye" = "bye"

splitAt :: Int -> [a] -> ([a],[a])
splitAt 4 "goodbye" = ("good","bye")

reverse :: [a] -> [a]
reverse "goodbye" = "eybdoog"

elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye" = True

replicate :: Int -> a -> [a]
replicate 5 '*' = "*****"

concat :: [[a]] -> [a]
concat ["con","cat","en","ate"] = "concatenate"

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

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

Figure 2: Some library functions

1. (a) The function `ord :: Char -> Int` converts a character to its corresponding ASCII code. For example,

```
ord 'A'  =  65      ord 'B'  =  66    ...    ord 'Z'  =  90
ord 'a'  =  97      ord 'b'  =  98    ...    ord 'z'  = 122
```

Using `ord`, write a function `f :: Char -> Int` that converts a letter, lower case or upper case, to its ordinal position in the alphabet. For example,

```
f 'A'  =  0         f 'B'  =   1    ...      f 'Z'  =  25
f 'a'  =  0         f 'b'  =   1    ...      f 'z'  =  25
```

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

[*6 marks*]

(b) Using `f`, define a function `g :: String -> Int` that given a string returns the sum of the ordinal numbers of every letter in the string, ignoring all characters that are not letters. For example, `g "aBc4e"` returns `7`, and `g "?!"` returns `0`. Your definition may use *list comprehension*, arithmetic, comparison, logic, and *library functions*, but not recursion.

[*6 marks*]

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

[*6 marks*]

2. (a) Write a function `c :: [Int] -> [Int] -> [Int]` that takes two lists of integers, and returns the difference between corresponding elements. For example, `c [5,7,3] [1,2,4]` returns `[4,5,-1]`. Indicate an error if the two input lists are not the same length. Your definition may use *list comprehension*, arithmetic, comparison, logic, and *library functions*, but not recursion.

[*6 marks*]

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

[*6 marks*]

(c) Using `c`, write a function `e :: [Int] -> [Int] -> Bool` that returns true if the two lists are equal. (You could just use `==` for this, but don't. Find a method that uses `c` instead.)

[*5 marks*]