UNIVERSITY OF EDINBURGH
COLLEGE OF SCIENCE AND ENGINEERING
SCHOOL OF INFORMATICS

Date: Monday 30th October 2006
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, copies of Figures 5.1 and 5.2 (pp. 91 and 92) are included from Simon Thompson, *Haskell, The Craft of Functional Programming*. You will not need all the functions listed.

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

MATRICULATION NUMBER	NAME

1. (a) Write a function `sign :: Int -> Char` that takes an integer and returns
   - `'+'` if the integer is between 1 and 9 (inclusive),
   - `'0'` if the integer is 0,
   - `'-'` if the integer is between -1 and -9 (inclusive),

   and indicates an error otherwise (using the `error` function).

   [*6 marks*]

   (b) Write a function `signs :: [Int] -> String` that takes a list of integers, and returns the sign of each integer between -9 and 9 (inclusive), ignoring any number out of that range. For example, `signs [5, 10, -5, 0]` returns `"+-0"`. Your definition should use *list comprehension*, but not recursion. You should use no library functions other than arithmetic and comparison.

   [*6 marks*]

   (c) Write a second definition of `signs`, this time using *recursion* and not a list comprehension. You should use no library functions other than arithmetic and comparison.

   [*6 marks*]

2. (a) Write a function `upcount :: [Int] -> Int` that returns a count of the number of transitions in a non-empty list from a smaller number to a larger. For example, `upcount [1,7,2,2,5,6]` returns `3` (counting the transitions from 1 to 7, 2 to 5, and 5 to 6). Your definition should use *list comprehension*, but not recursion. You may also use any library function from Figures 9.2 or 9.3, and arithmetic and comparison.

[*6 marks*]

(b) Write a second definition of `upcount`, this time using *recursion* and not a list comprehension. You should use no library functions other than arithmetic and comparison.

[*6 marks*]

(c) Using `upcount`, write a definition of `ascending :: [Int] -> Bool` that takes a non-empty list and returns true if it is in strictly ascending order.

[*5 marks*]