National Undergraduate Programme in Mathematical Sciences

Introduction to Programming

Mid-semester Examination, Semester I, 2024–2025

Date : October 3, 2024
Time : 0930–1130

Marks : 30
Weightage : 30%

This paper has two parts. Each Part A question is worth 2 marks, and each Part B question is worth 5 marks. For Part A, provide short answers. For Part B, provide crisp and short programs. Syntax should more or less correspond to Haskell, but minor inaccuracies like wrong indentation will be ignored. You can assume standard library functions, but if you use something not taught in class, you must explain briefly what it does.

Part A

1. Suppose `revInto`, `l1` and `l2` are defined as follows:

```
revInto [] a        = a
revInto (x:xs) a    = revInto xs (x:a)

l1 = foldr revInto [] [[0,1], [2..5], [6..13]]
l2 = foldl revInto [] [[0,1], [2..5], [6..13]]
```

   What are the values of `l1` and `l2`?

2. How many times is the second line of the definition (of `revInto`) invoked in the computation of `l1`?

3. How many times is the second line of the definition (of `revInto`) invoked in the computation of `l2`?

4. What is the position of `(5,2)` in the following list (counting positions from `0`)?

```
[(j,i) | i←[0..9], j ← [(i+1)..9]]
```

5. Give a polymorphic type of the function `g` defined as follows:

```
g f = foldr (\x l → f x : l) []
```

Part B

1. Define a function `inverse :: (Integer → Integer) → Integer → Integer` whose behaviour is as follows. For any `f` which satisfies ~~`x <= f x`~~ for all `x :: Integer`, `inverse f n` will produce the largest `m` s.t. `f m ⩽ n < f (m+1)`. The number of times `f` is called from `inverse f n` should be proportional to `log n`.

2. Define a function `maximums :: [Int] → [Int]` which returns the maximum value of each non-empty prefix of the given list. Here are a few sample cases:

```
maximums []                = []
maximums [0..9]            = [0,1,2,3,4,5,6,7,8,9]
maximums [9,8..0]          = [9,9,9,9,9,9,9,9,9,9]
maximums [-8,-3,-10,22,5]  = [-8,-3,-3,22,22]
```

   The number of steps taken by your program should be proportional to the length of the input list.

3. Define a function `isZigZag :: [Int] → Bool` which returns **True** exactly if the input list `[x1, x2, x3, x4, ..., xn]` satisfies the condition `x1 ⩽ x2 ⩾ x3 ⩽ x4 ⩾ ...`. The number of steps should be proportional to the length of the input list.

4. Given the following definition of `fib :: Integer → Integer`, give the complete trace of the computation of `fib 6`. (Remember that any definition is expanded only if there is a need. To compute `e1 + e2`, one needs to reduce first `e1` and then `e2` to a number.)

```
fib n = if n ⩽ 1 then n else fib (n-2) + fib (n-1)
```