Introduction to Programming: Mid-semester Make-up Examination

Date: November 20, 2026 **Time:** 1100 – 1300 hrs **Marks:** 100 **Weightage:** 25%

This paper has two parts.
There are 8 questions in Part A, each worth 5 marks. For this part, provide the solution and a short explanation justifying your answer.
There are 3 questions in Part B, each worth 20 marks. 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

(8 × 5 = 40 marks)

1. What is the type of the following function?

```
g f xs = zipWith f xs (tail xs)
```

2. Evaluate the following expression:

```
drop 3 (takeWhile (< 10) [2, 4 ..])
```

3. Evaluate the following expression:

```
zip [1 ..] (drop 4 "haskell")
```

4. Give a type for the expression `(take 2, drop 2)`.

5. What is the value of `zipWith (*) [1, 2, 3] (drop 1 [1, 2, 3, 4])`?

6. Evaluate the following expression:

```
concat (map (replicate 2) "abc")
```

7. Is it true that `filter p xs ++ filter (not . p) xs == xs` for every predicate `p` and every finite list `xs`? If yes, explain why; if not, give a counterexample.

8. Express `zipWith f xs ys` using only `map`, `zip` and `f` (and, if needed, a lambda expression).

Part B

(3 × 20 = 60 marks)

1. Consider the following function `chunks`.

```
chunks :: Int → [a] → [[a]]
chunks _ [] = []
chunks n xs = take n xs : chunks n (drop n xs)
```

   (a) Trace the computation of `chunks 3 [1, 2, 3, 4, 5, 6, 7, 8]`.

   (b) Describe (in a short sentence) the relationship between `concat (chunks n l)` and `l`, for any `n > 0` and any finite list `l`.

   (c) What happens when we evaluate `chunks 0 [1, 2, 3]`? Explain briefly.

2. Write a function `splitOn :: Char → String → [String]` that splits a string into the pieces separated by a given character. For example:

```
splitOn ',' "ab,c,,d" == ["ab", "c", "", "d"]
splitOn ',' "abc"     == ["abc"]
splitOn ',' ",a,"     == ["", "a", ""]
```

   You may use any standard library functions, but you must write the core logic yourself.

3. Write a function `localMaxima :: [Int] → [Int]` that returns, in order, all elements of the input list that are strictly greater than both their immediate neighbours. (The first and last elements, having only one neighbour, are never included.) Below are a few examples.

```
localMaxima [2,9,5,6,1] = [9,6]
localMaxima [2,3,4,1,5] = [4]
localMaxima [1,2,3,4]   = []
```