Introduction to Programming: Mid-semester Examination

Date: September 22, 2026 **Time:** 0930 – 1230 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 4 questions in Part B, each worth 15 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?

```
f x y z = x (y z)
```

2. Evaluate the following expression.

```
take 3 (filter even [1..10])
```

3. What is the value of `foldl (-) 0 [1, 2, 3, 4]`?

4. What is the value of `foldr (-) 0 [1, 2, 3, 4]`?

5. Find `f` and `v` such that `concat xss = foldr f v xss` for all lists of lists `xss`.

6. Find `f` and `v` such that `reverse xs = foldl f v xs` for all lists `xs`.

7. Give a type for the expression `zipWith (+)`.

8. Evaluate the following expression.

```
dropWhile (< 5) [1, 3, 5, 7, 2, 4]
```

Part B

(4 × 15 = 60 marks)

1. Consider the following function `f`.

```
f :: [Int] → [Int]
f = go [] where
  go _ [] = []
  go seen (x:xs)
    | x `elem` seen = go seen xs
    | otherwise     = x : go (x:seen) xs
```

   (a) Trace the computation of `f [1, 2, 3, 2, 4, 1, 5, 3, 2, 1]`.
   Tracing means that you write a sequence of rewrites, where you start with the original expression, and each step rewrites an appropriate subexpression by following the program.

   (b) Describe (in a short sentence) the relationship between any (finite) list `l` and `f l`.

2. Write a function `groupRuns` that takes a list and groups consecutive identical elements together. For example:

```
groupRuns [1, 1, 2, 3, 3, 3, 1] = [[1, 1], [2], [3, 3, 3], [1]]
```

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

3. Write a function `digsToInt :: [Int] → Int` whose input is a list of single-digit integers ($\geqslant 0$ and $\leqslant 9$) and whose output is the integer value obtained by reading the list from left to right. Below are a few examples.

```
digsToInt []           = 0
digsToInt [1,1,1,2]    = 1112
digsToInt [0,0,1,0,0,2] = 1002
```

4. Write a function `intToDigs :: Int → [Int]` whose input is a non-negative integer and whose output is a non-empty list of all digits of the integer, read from left to right. Below are a few examples.

```
intToDigs 0    = [0]
intToDigs 1112 = [1,1,1,2]
intToDigs 1002 = [1,0,0,2]
```