Programming Fundamentals 3
BAINFOR-17
Exam WS2025–26

Dr Afonso Arriaga

January 23th 2026

Rules

- Duration: 90 minutes (1h30).
- Please ensure that you write your **full name** and **student ID** on all provided pieces of paper.
- Please write your answers only on the provided pieces of paper.
- If you run out of writing space, ask for additional paper.
- Ensure clear legibility of your answers; prefer shorter and precise answers, avoid ambiguities.
- If you write your answer to a different place than the reserved space below the question, kindly label all your answers clearly with the corresponding question number, such as *"Q2.1 (b)"*.

Question 1 (Warm up quiz)

Mark the correct answer.
(In case you make a mistake and need to change the marking, **avoid double marking or mark corrections!** Instead, explain the correct answer using words in the empty space next to the list, explicitly referring to the letter label of the desired answer.)

Q1.1 (0.5 points) Which of the following is an invalid list in Haskell:

(a) `[(1, "one", [1])]`
(b) `[[1], [1,1]]`
(c) `[1,"one"]`
(d) `["one",[]]`

Q1.2 (0.5 points) The expression `["67","six-seven"]` has type:

(a) `[Num]`
(b) `[Int]`
(c) `[String, String]`
(d) `[Int, String]`
(e) `[String]`

Q1.3 (0.5 points) The function `partition` has type `(a -> Bool) -> [a] -> ([a], [a])`. What is the type of `uncurry partition`:

(a) `(a -> Bool) -> [a] -> ([a], [a])`
(b) `[a] -> ([a], [a])`
(c) `(a -> b -> c) -> (a, b) -> c`
(d) `(a -> Bool, [a]) -> ([a], [a])`
(e) `(a -> Bool) -> ([a], [a])`

Q1.4 (0.5 points) The expression `Node "RootNode" (Leaf "LeftNode" 1) (Leaf "RightNode" 2)` is a valid value of the datatype:

(a) `data Tree Int = Leaf String Int | Node Int`
(b) `data Tree = Leaf Tree | Node Int Int`
(c) `data Tree a = Leaf a | Branch Tree Tree`
(d) `data Tree a = Leaf String a | Node a (Branch a) (Branch a)`
(e) `data Tree a b = Leaf a b | Node a (Tree a b) (Tree a b)`

Question 2 (Prelude)

Q2.1 (1 point) Define the following functions from `Prelude` using recursion:

(a) `zipWith3 :: (a -> b -> c -> d) -> [a] -> [b] -> [c] -> [d]`

(b) `maximum :: Ord a => [a] -> a`
(Note: The type of `maximum` in Prelude is actually `(Foldable t, Ord a) => t a -> a`, but here we do not require an implementation for such fully generic type.)

Q2.2 (1 point) Define the following functions from `Prelude` using `foldr`:

(a) `length :: [a] -> Int`
(Note: The type of `length` in Prelude is actually `(Foldable t) => t a -> Int`, but here we do not require an implementation for such fully generic type.)

(b) `map :: (a -> b) -> [a] -> [b]`

Question 3 (Algorithms)

Q3.1 (3 point) Goldbach's conjecture says that every positive even number greater than 2 is the sum of two prime numbers. Example: 28 = 5 + 23. It is one of the most famous facts in number theory that has not been proved to be correct in the general case. The conjecture has been shown to hold for all natural numbers less than 4×10^{18}, but remains unproven despite considerable effort.

Recall the following (lazy) infinite list of prime numbers in Haskell:

```
primes = filterPrime [2..] where
  filterPrime (p:xs) =
    p : filterPrime [x | x <- xs, x `mod` p /= 0]
```

Write a function that given an even integer $n > 2$ returns a pair of prime numbers (p, q) such that $p + q = n$.

```
goldbach :: Integer -> (Integer, Integer)
```

Q3.2 (3 point) Bata-Miaou is a cat-themed card battle for two players with decks numbered 1-6. Each round, both reveal their top card:

- The higher card wins both cards, adding them to the bottom of their deck in this order: own card first, then opponent's card.
- A tie discards both cards.

The player who empties their opponent's deck wins.

Write a function that determines the winner, if there is one.

```
winner :: [Int] -> [Int] -> Maybe Int
```

However, some games cycle indefinitely:

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

To handle endless games, after 100 rounds the player with more cards wins.
A few examples:

```
winner [3,1] [2]     == Just 1
winner [2,3] [1,4]   == Just 2
winner [] []         == Nothing -- tie
```

Question 4 (Trees)

Q4.1 (1 point) Shortly explain what is an AVL tree, and the advantages and disadvantages of this data structure over regular Binary Search Trees (BST).

Q4.2 (2 points) Given the datatype `data Tree a = Nil | Node a (Tree a) (Tree a)`, write a Haskell function `isBST :: Ord a => Tree a -> Bool` to determine if a given `Tree a` is a BST.

Question 5 (Data structures)

Q5.1 (2 points) Consider the datatype `MList` below:

```
data MList = MList { value :: Int, next :: Maybe MList }

compareML :: Maybe MList -> Maybe MList -> Bool
```

Write a function `compareML` which compares two arguments of type `Maybe MList`, and outputs `True` if and only if they are equal in all values.

Question 6 (Monads)

Q6.1 (2 points) Show how the type `Response` defined below can be made an instance of typeclasses `Functor`, `Applicative` and `Monad`.

```
data Response a = ResponseMissing | ResponseData a
```

Q6.2 (2 points) Assuming that `Response` is an instance of `Monad`, write a function `summarize` that evaluates a list of responses (of type `Response String`) and outputs a single response where all responses are concatented into a single one (i.e., again of type `Response String`). If any response is missing, the output response should also be marked as missing.

```
summarize :: [Response String] -> Response String

ghci> summarize [ResponseData "aaa", ResponseData "xxx"]
ResponseData "aaaxxx"

ghci> summarize [ResponseData "bbb", ResponseMissing, ResponseData "yyy"]
ResponseMissing

ghci> summarize []
ResponseData ""        -- nothing failed, but the response is empty
```

Question 7 (Debugging session)

Find the mistake(s) is the following function definitions, shortly explain what is the problem and propose a fix.

Q7.1 (1 point) Below is a small Haskell program designed to read a string from `STDIN`, encode it using the Caesar Cipher with a shift of 13, and print the encoded string to the `STDOUT`. Find and fix the two bugs.

```
import Data.Char ( ord, chr, isAlpha, isUpper )

shiftChar :: Int -> Char -> Char
shiftChar shift c
    | isAlpha c = chr $ base + (ord c - base + shift) `mod` 26
    | otherwise = c
  where
    base = if isUpper c then ord 'a' else ord 'A'

caesarCipher :: Int -> String -> String
caesarCipher shift = shiftChar shift

main :: IO ()
main = interact (caesarCipher 13)
```

Here is an example of how quickly test this code.

```
% echo "This was so easy" | runhaskell caesarCipher.hs
Guvf jnf fb rnfl
```

(Use this page as additional space for your answers, if required.)