National Undergraduate Programme in Mathematical Sciences
National Graduate Programme in Computer Science
Functional Programming in Haskell

Final Examination, Semester I, September 2021–December 2021

Date : March 25, 2021
Time : 0930 – 1230

Marks : 100
Weightage : 40%

- *This paper has two parts. Each Part A question is worth 5 marks, and each Part B question is worth 20 marks.*
- *Submit your answer as a TXT file titled* `<username>-final.txt` *on Moodle.*
- *If for some reason you cannot submit a TXT file, scan your handwritten answers and submit a PDF.*
- *Write your name and CMI id on the first two lines of the file.*
- *You can use any of the commonly used library functions in your solutions.*

Part A

1. How many values belong to the type `Exp` defined below?

```
data Dig = Zero | One | Two | Three | Four | Five
         | Six | Seven | Eight | Nine deriving (Eq, Ord)
data Op = Add | Sub | Mul | Div deriving (Eq, Ord)
type Exp = (Op, Dig, Dig)
```

2. For how many inputs `e :: Exp` does `goodExp e` return `True`?

```
goodExp :: Exp → Bool
goodExp (op, d1, d2) = op ⩽ Sub || (d1 ⩾ Four && d2 ⩽ Six)
```

3. Is `20` an element of the following list?

```
zipWith (*) [2,4..] [6,4..]
```

4. Consider the following two `IO` actions.

```
act m = do
    b ← readLn :: IO Bool
    return (m || b)

main = do
    inp ← readLn :: IO Bool
    list ← replicateM 5 (act inp)
    if (and list) then return inp else return (not inp)
```

What are the types of `act` and `main`?

5. How many lines of user input are read when you run `main`?
6. Given below are two expressions `f` and `g`. Which takes more steps and why?

```
f = foldl (++) [] ["abc", "def", "ghi", "jkl"]
g = foldr (++) [] ["abc", "def", "ghi", "jkl"]
```

7. What is the result of `foldr (\x (y:ys) → y:x+y:ys) [0] [0..9]`?
8. Supply a function `f` such that `foldl f [] == reverse`.

Part B

1. (a) Write a function `myReplicate :: Int → a → [a]` such that `myReplicate i x` creates a list with `i` occurrences of `x` if `i > 0`, and creates `[]` otherwise.
   (b) Write a program `countFalse :: [Bool] → Int` that counts the number of occurrence of `False` in a list `ls :: [Bool]`.
   (c) Using `countFalse` and `myReplicate`, define `sortBools :: [Bool] → [Bool]` which sorts a list `ls :: [Bool]` in time proportional to `length ls`.
2. We defined binary trees in class, where each node has two children (either or both of which can be `Nil`). But we can have more general trees, where each node has any finite number of children. In this case the children of a node are represented as a list (the empty list indicating that there are no children). The Haskell definitions are given below:

```
data BTree = Nil | BNode Int BTree BTree
data Tree = Node Int [Tree]
```

There is a way to encode an arbitrary tree as a binary tree. Here the left child (of the binary tree) denotes the leftmost child of the original tree, while the right child denotes the next sibling of the node. Below, `tree1` is a general tree, and `tree2` is its binary tree encoding.

```
tree1 :: Tree
tree1 = Node 1 [Node 2 [], Node 3 [Node 4 [], Node 5 [], Node 6[]],
    Node 7 [Node 8 []]]

tree2 :: BTree
tree2 = BNode 1 (BNode 2 Nil (BNode 3 (BNode 4 Nil (BNode 5 Nil (BNode
    6 Nil Nil))) (BNode 7 (BNode 8 Nil Nil) Nil))) Nil
```

Define a function `encode :: Tree → BTree` that converts a general tree to its binary tree representation.

3. Define a function `decode :: BTree → Tree` that converts a binary tree to the general tree it encodes. (We assume that the binary tree is not `Nil` and that its root has no right subtree.)