# National Undergraduate Programme in Mathematical Sciences
# National Graduate Programme in Computer Science

## Functional Programming in Haskell

### Final Examination, Semester I, December 2020–March 2021

Date : March 25, 2021
Time : 0930 – 1230

Marks : 50
Weightage : 50%

---

- *This paper has two parts. Each Part A question is worth 2.5 marks, and each Part B question is worth 6 marks.*
- *Submit your answer as a TXT file titled* **`<username>-final.txt`** *on Moodle.*
- *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. List all values of the type **`Maybe (Maybe Bool)`**?
2. How many values belong to the type **`Test`** defined below?

```
data Day = Sun | Mon | Tue | Wed | Thu | Fri | Sat
data Test = History Day | Science Day Day
```

3. What is the maximum value in the type **`(Func, Bool)`**, where **`Func`** is defined as follows:

```
data Func = Foo | Bar | Loop | Gloop
    deriving (Eq, Ord)
```

4. Find a function **`f :: [Int] -> [Int]`** such that the following holds:

```
f [22, 23, 15, 4, 16, 2, 3, 28] = [2, 3, 4, 15]
```

5. Consider the following two **`IO`** actions.

```
act m = do
    b <- readLn :: IO Bool
    return (even m == b)

main = do
    inp <- getLine
    x <- act (length inp)
    if x then return inp else return (inp++inp)
```

What is the type of `main`?

6. How many lines of user input are read when you run `main`?

7. 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"]
```

8. What is the result of `foldr (\x (y:ys) -> y:x+y:ys) [0] [0..9]`?

## Part B

1. How many times is the recursive call to `f 0` made in the computation of `f 4`, given the following definition of `f`?

```
f 0 = v
f n = g n n where
    g i j = if i == 0
            then f (j-1)
            else f (j-1) + g (i-1) j
```

In general, what is the value of `f n`? Justify your answer briefly.

2. (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] -> [Bool]` that counts the number of occurrence of `False` in a list `ls :: [Bool]`.

(c) Use **`countFalse`** and **`myReplicate`** to write a program **`sortBools :: [Bool] -> [Bool]`** that sorts a list **`ls :: [Bool]`** in time proportional to **`length ls`**.

3. A *dyadic numeral* is a string of numbers of the form $x_n \ldots x_1$ ($n \geqslant 0$), where each $x_i \in \{1, 2\}$. (Note that the case when $n = 0$ represents the empty string.) Its value is given by

$$\sum_{i=1}^{i=n} x_i \cdot 2^{i-1}.$$

(Note that the value of the empty string is 0, since the sum of an empty sequence of numbers is 0, by convention.) The advantage of dyadic numerals over binary numbers (as you can verify at leisure) is that each natural number has a unique dyadic representation. We represent dyadic numerals as lists. The representations for 0 to 5 are **`[]`**, **`[1]`**, **`[2]`**, **`[1,1]`**, **`[1,2]`**, and **`[2,1]`** respectively.

Define a function **`value :: [Int] -> Int`** such that **`value ls`** gives the number represented by the dyadic numeral given by **`ls`**. You should not use **`(!!)`** or **`(2^)`** in your definition.

4. Define the function **`dyadic :: Int -> [Int]`** such that **`dyadic n`** gives the dyadic representation of the natural number **`n`**.

5. Consider the following definition of a binary tree which stores values only at the leaves.

```
data BTree = Leaf Int | Fork BTree BTree
```

Suppose we want to compute the minimum and maximum value of each subtree and store it at the **`Fork`**. That leads us to the following definition of a *decorated binary tree*.

```
data DBTree = DLeaf Int | DFork DBTree (Int, Int) DBTree
```

The idea is that in **`DFork dtl (y,z) dtr`**, **`y`** is the minimum value among all the leaves of **`dtl`** and **`dtr`**, whereas **`z`** is the maximum value among all the leaves of **`dtl`** and **`dtr`**.

Define a function **`decorate :: BTree -> DBTree`** that computes the decorated binary tree corresponding to a binary tree.