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

Functional Programming in Haskell

Mid-semester Examination, I Semester, 2022–2023

Date : September 27, 2022
Time : 0930 – 1230

Marks : 100
Weightage : 30%

This paper has three parts. Each Part A question is worth 4 marks, and each Part B question is worth 6 marks. Part C is worth 50 marks. For Part A, provide answers in the answer sheet, and write your option as well as the answer, like "(a) 225" or "(b) True".

Part A

1. What is the result of the following expression?

```
length $ filter (≥ -5) $ reverse [(-25)..10] ++ [(-25)..10]
```

(a) 0 (b) 16 (c) 30 (d) 32

2. What is the result of the following expression?

```
length $ takeWhile (≥ -5) $ reverse [(-25)..10] ++ [(-25)..10]
```

(a) 0 (b) 16 (c) 30 (d) 32

3. Which of the following is a possible type of the function `foldr (zipWith (+))`?

(a) `[a] → a` (b) `a → [a] → a`
(c) `[a] → [[a]] → [a]` (d) `[Int] → [[Int]] → [Int]`

4. Suppose `(++)` is defined as follows:

```
[]     ++ ys = ys
(x:xs) ++ ys = x: xs ++ ys
```

How many times is the second line of the definition invoked in the computation of the following expression?

```
foldr (++) "" ["abcde", "fghij", "klmno", "pqrst", "uvwxy", "z"]
```

(a) 25 (b) 26 (c) 75 (d) 101

5. How many times is the second line of the definition (of (++)) invoked in the computation of the following expression?

```
foldl (++) "" ["abcde", "fghij", "klmno", "pqrst", "uvwxy", "z"]
```

(a) 25 (b) 26 (c) 75 (d) 101

Part B

1. What is the position of (5,2) in the following list (counting positions from 0)?

```
[(j,i) | i←[0..9], j ← [(i+1)..9]]
```

2. Fill in values for g and v such that the following equation holds (for any `f :: a → [b]`).

```
concatMap f = foldr g v
```

What are the types of your g and v?

(*Hint*: Recall that

```
concatMap f [x1, x2, ..., xn] = f x1 ++ f x2 ++ ... f xn.
```

Also note that your expression for g can involve f.)

3. What is the value of the following expression?

```
concatMap (flip replicate 'a') [0..5]
```

4. Define the function `isSubseq :: [Char] → [Char] → Bool` such that `isSubseq xs ys` is `True` exactly when `xs` is a subsequence of `ys`, i.e. `xs` is obtained by omitting some characters in `ys` and reading the remaining characters from start to end.

5. Define a function `member :: Eq a = a → [a] → [a]` with the following behaviour.

If x is not in the list ys, the return value is `[]`. Otherwise it is the suffix of ys starting from the first occurrence of x.

Sample cases:

```
member 'a' ['b'..'m'] = []
member 2 (reverse [0..9] ++ [0..9]) = [2,1,0,0,1,2,3,4,5,6,7,8,9]
```

Part C

1. Given the following definition of `fib :: Int → Integer`, give the complete trace of the computation of `fib 5`. (Remember that any definition is expanded only if there is a need. To compute `e1 + e2`, one needs to reduce first `e1` and then `e2` to a number.) (10 marks)

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-2) + fib (n-1)
```

2. Trace the computation of `fib 3` for the following definition of `fib`, defined below. (10 marks)

```
fib n                = fibs `at` n where
    fibs             = 0:1:z
    z                = 1:go (1:z) z
    go (x:xs) (y:ys) = x+y: go xs ys
    at (x:xs) 0      = x
    at (x:xs) n      = at xs (n-1)
```

3. Consider the following function for computing the length of the longest common subsequence between as and bs.

```
lcs :: String → String → Int
lcs ""       _        = 0
lcs _        ""       = 0
lcs (a:as)   (b:bs)
    | a == b          = 1 + lcs as bs
    | otherwise       = max (lcs (a:as) bs) (lcs as (b:bs))
```

Modify the above function to obtain the following function.

```
lcsCount :: String → String → (Int, Integer)
```

`lcsCount as bs` returns `(len,n)` where `len` is the length of the longest common subsequence of `as` and `bs`, while `n` is the number of recursive calls made to `lcs` in the process of computing `lcs as bs` (including the initial call). (15 marks)

Sample cases:

```
lcsCount "abcd"          ""          = (0,1)
lcsCount ""              "abcdef"    = (0,1)
lcsCount "aaa"           "aaaa"      = (3,4)
lcsCount "ab"            "cb"        = (1,9)
lcsCount "abbb"          "cbbb"      = (3,27)
lcsCount "LeBron James"  "Michael Jordan"  = (4,7464489)
```

4. Consider a function that constructs a roman numeral representation from its positive integer argument, *n*. The Roman method of writing numbers uses two kinds of symbols – the basic symbols are I=1, X=10, C=100 and M=1000 the auxiliary symbols are V=5, L=50 and D=500. A rule prescribes that the symbol for the larger number always stands to the left of that for the smaller number. An exception is motivated by the desire to use as few symbols as possible. For example, the number nine can be represented as VIIII (5+4) or IX (10-1), but the second representation is preferred. Therefore, if the symbol of a smaller number stands at the left, the corresponding number has to be subtracted, not added. It is not permitted to place several basic symbols or an auxiliary symbol in front. For example, we should use VIII for 8 instead of IIX, and CML for 950 instead of LM.

Sample values

```
numToRom 2022    = "MMXXII"
numToRom 3348    = "MMMCCCXLVIII"
numToRom 2989    = "MMCMLXXXIX"
```

Given below is a program to compute the roman numeral representation of a number. Complete the parts indicated by ... so that the resulting program works correctly. *(15 marks)*

```
valReps :: [(Int, String)]
valReps = [
  (999, "IM"), (990, "XM"), (900, "CM"), (500, "D" )
, (499, "ID"), (490, "XD"), (400, "CD"), (100, "C" )
, ( 99, "IC"), ( 90, "XC"), ( 50, "L" ), ( 49, "IL")
, ( 40, "XL"), ( 10, "X" ), (  9, "IX"), (  5, "V" )
, (  4, "IV"), (  1, "I" )
]

numToRom :: Int → String
numToRom n  = replicate q 'M' ++ go r valReps where
    (q,r)   = n `quotRem` 1000
    go :: Int → [(Int,String)] → String
    go 0 _                  = ""
    go _ []                 = ""
    go m l@((val,rep):rest)
        | m < val           = go ... ...
        | otherwise         = ... ++ go ... ...
```


4