National Undergraduate Programme in Mathematical Sciences

National Graduate Programme in Computer Science

Functional Programming in Haskell

End-semester Examination, Semester I, 2023–2024

Date : November 27, 2023
Time : 1100–1330

Marks : 100
Weightage : 40%

This paper has two parts. Each Part A question is worth 8 marks. Each PArt B question is worth 20 marks. For Part A, provide short answers. For Part B, provide crisp and short programs. 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

1. List all the values belonging to the following type

```
Maybe (Bool, Maybe Bool)
```

2. Consider the following data declaration:

```
data Dir = North | East | West | South deriving (Eq, Ord)
```

What is the minimum value of the type `(Dir, Maybe Bool)`?

3. Consider the following two `IO` actions.

```
act m = do
    b1 ← readLn :: IO [Int]
    b2 ← readLn :: IO [Int]
    return ((even m) == (length b1 ⩽ length b2))

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

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

4. Consider the following data declaration:

```
data BinTree = Nil | Node BinTree Int BinTree
```

Write a program `multiplyAll :: BinTree → Int` that returns the product of all elements in the input tree.

5. What is the time complexity of the following program, in terms of the length of the list `xs`?

```
f xs    = foldr (\x l → x:sum l) [] xs
```

Part B

1. Trace the computation of `main`, given the following definition of **`map`**. It is important to remember that Haskell is lazy.

```
map f []        = []
map f (x:xs)    = f x : map f xs

main    = map (+1) (map (^2) [1..5])
```

2. 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 outputs the decorated binary tree corresponding to a binary tree. Try to ensure that your program runs in time proportional to the size of the input tree.

3. Given the above definition of `BTree`, define a function

```
leaves :: BTree → [Int]
```

that lists the leaves (in order from leftmost leaf to rightmost leaf). Define a function

```
buildBTree :: [Int] → BTree
```

such that `leaves (buildBTree l)` $=$ `l`. The function should take $O(n)$ time and produce a tree of height $O(\log n)$