National Undergraduate Programme in Mathematical Sciences

National Graduate Programme in Computer Science

Functional Programming in Haskell

End-semester Examination, I Semester, 2022–2023

Date : November 22, 2022
Time : 0930 – 1230

Marks : 100
Weightage : 40%

This paper has two parts. Each Part A question is worth 4 marks, and each Part B question is worth 20 marks.

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 Bool
    b2 ← readLn :: IO Bool
    return (even m == b1 && 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. How many lines of input are read when `main` (given above) is run?

5. 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.

1

Part B

1. The Haskell notation `[f,s..l]` is used to generate the following list:

```
[f, f + d, f + 2*d, f + 3*d, ... , f + n*d]
```

(with `d = s-f` and n s.t. `f+n*d ≤ l < f + (n+1)*d`). Note that the list is empty if `f` is between `f+d` and `l`. Write a program

```
enumFromThenTo :: Int → Int → Int → [Int]
```

such that `enumFromThenTo f s l = [f,s..l]`.

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] → Int
```

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. 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.

4. 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 $O(\log n)$