YOUR NAME: ____________________________

Programming in Haskell Quiz 2

Friday October 4, 2024

20 points

Rules: You are allowed to (1) use class notes and the textbook, (2) run examples in the interpreter and (3) discuss with classmates. You are NOT allowed to search the internet (most quiz functions and structures are available in hackage.)

Once you're done, type the answers into a text file and submit in Submitty.

Questions 1 and 2 refer to the `NL` structure and `atomcount` and `flatten` functions from class:

```
data NL a = Atom a
          | List [NL a]

> x = List [Atom 1, Atom 2, Atom 3]
> y = List [List [Atom 1, Atom 2, Atom 3]]
> z = List [Atom 0, List [List [Atom 1]]]

> atomcount x -- Counts number of atoms (nested arbitrarily deeply) in x
3
> atomcount z
2
> flatten z -- Flattens z into a list of a-values ordered by infix traversal
[0,1]
```

Question 1. (4pts) Define a suitable `foldMap` function for `NL`.

```
instance Foldable NL where
     -- foldMap :: _____________________________________ -- fill in signature
     foldMap f nl =
```

Question 2. (4pts) Define `atomcount` and `flatten` in terms of `foldMap`. You may assume the `Sum` and `Product` monoids we defined over the integers.

```
atomcount :: _____________________________________ -- fill in signature
atomcount nl =

flatten :: _____________________________________ -- fill in signature
flatten nl =
```

Question 3. (4pts) Rewrite the program into a semantically equivalent version that uses `>>=` or `>>`.

```
ggf :: Sheep -> Maybe Sheep
ggf sh = do
        s <- return sh
        m <- mother s
        gf <- father m
        father gf
```

Question 4. (4pts) Rewrite the program into a semantically equivalent version that uses `>>=` or `>>`.

```
lenProgram :: IO Int
lenProgram = do
    let x = length [1,2,3]
    putStrLn ("Length is" ++ show x)
    return x
```

Hint: Translate the late binding into `let x = length [1,2,3] in ...` and proceed to fill in the let-block with the rewrite of the action sequence.

Question 5. (4pts) Rewrite the program into a semantically equivalent version that uses `>>=` or `>>`.

```
fancyProgram :: IO ()
fancyProgram = do
    putStrLn "What is your name?"
    inpStr <- getLine
    if inpStr == "Haskell"
        then do putStrLn "You rock!"
                return ()
                putStrLn "Really!!"
        else putStrLn ("Hello " ++ inpStr)
    putStrLn "That's all!"
```