Fall 2022 Haskell Exam

Problem 1

Write a `main` function that accepts two command-line arguments and prints them with an arrow pointing from the first to the second. For example, if you run your program with `runhaskell main.hs caterpillar butterfly`, it produces this output:

```
caterpillar -> butterfly
```

Your solution:

```haskell
import System.Directory.Internal.Prelude (getArgs)
main = do
  first : second : _ <- getArgs
  putStrLn (first ++ " -> " ++ second)
```

Run

Problem 2

Write function `prettyTime` that accepts a total number of seconds as an `Int`. It returns a more readable `String` that chunks the total into hours, minutes, and seconds. For example, `prettyTime 18099` yields `"5 hours, 1 minute, 39 seconds"`.

Your solution:

```haskell
isPlural :: Int -> String -> String
isPlural x str
  | x == 1 = str
  | otherwise = str ++ "s"

prettyTime :: Int -> String
prettyTime x = do
  let h = x `div` 3600
  let m = (x - h * 3600) `div` 60
  let s = x - (h * 3600 + m * 60)
  show h ++ (isPlural h " hour") ++ ", " ++ show m ++ (isPlural m " minute
```

Run

Problem 3

Write function `runts` that accepts a list of pairs of lists. It returns a list of the smaller lists of each pair. In the case of a tie, the first component is chosen as the runt. For example, `runts [("no", "yes"), ("madrugada", "nariz")]` yields `["no", "nariz"]`.

Your solution:

```haskell
runts :: [([a], [a])] -> [[a]]
runts list = do
  let fsts = map (\(x,y) -> if length x <= length y then x else y) list
  fsts

```

Run

Problem 4

Write function `dilate` that accepts a list of `Int`. It returns a pair of lists in which the first component is the list of negatives and the second component is the list of positives. However, all numbers have been incremented or decremented away from zero. For example, `dilate [3, -6, 0, 1]` yields `([-7], [4, 2])`.

Your solution:

```haskell
dilate :: [Int] -> ([Int], [Int])
dilate nums = do
  let n = filter (\x -> x < 0) nums
  let n1 = map (-1+) n
  let p = filter (\x -> x > 0) nums
  let p1 = map (1+)p
  (n1,p1)
```

Run

Problem 5

Write a `main` function that reads in a file whose path is passed as a command-line argument. It prints two lines of output. The first is the number of zeroes found in the file. The second is the number of ones.

Your solution:

```haskell
import System.Directory.Internal.Prelude (getArgs)
main = do
  file : _ <- getArgs
  text <- readFile file
  let zeroes = filter (== '0') text
  print $ length zeroes
  let ones = filter (== '1') text
  print $ length ones
```

Run

Problem 6

Define a `Rectangle` data type with two variants. `Square` holds a side length. `Oblong` holds a width and height. All fields are `Int` values. Make `Rectangle` implement the `Show` typeclass. Define also a function `area` that accepts a `Rectangle` parameter and returns its area.

Your solution:

```haskell
data Rectangle = Square Int | Oblong Int Int
  deriving Show

area :: Rectangle  -> Int
area rect =
  case rect of
    Square s -> s * s
    Oblong w h -> w * h

```

Run