Spring 2023 Haskell Exam

Problem 1

Write a `main` function that accepts a single command-line argument representing the current color of a stoplight, which is one of `"green"`, `"yellow"`, or `"red"`. It prints the color that the light will switch to next. For example:

```
> runhaskell main.hs green
yellow
> runhaskell main.hs red
green
> runhaskell main.hs Red
?
> runhaskell main.hs flung
?
```

If the argument is any other string or uses a different case, print `"?"`.

Your solution:

```haskell
import System.Directory.Internal.Prelude (getArgs)
main = do
  color: _ <- getArgs
  putStrLn $ nextCol color

nextCol :: String -> String
nextCol x
  | x == "red" = "green"
  | x == "green" = "yellow"
  | x == "yellow" = "red"
  | otherwise = "?"
```

Run

Problem 2

Write function `mapFromTo` that accepts a list of any type and a transformation function. It returns a list of pairs. The first element of each pair is the corresponding element from the original list. The second element is the value to which it transforms. For example:

```
> mapFromTo length ["a", "an", "the"]
[("a", 1), ("an", 2), ("the", 3)]
```

Your solution:

```haskell
mapFromTo :: (a -> b) => [a] -> [(a,b)]
mapFromTo transform list = map (\x -> (x, transform x)) list
```

Run

Problem 3

Write function `days` that accepts a year, a month, and a list of dates. Each date is a 3-tuple of a year, month, and day of the month. The function returns the days of the month for those dates that fall in the given year and month. For example:

```
> days 2023 3 [(2023, 3, 27), (2023, 4, 1), (2023, 3, 15
[27, 15]
> days 2023 4 [(2023, 3, 27), (2023, 4, 1), (2023, 3, 15
[1]
> days 2023 5 [(2023, 3, 27), (2023, 4, 1), (2023, 3, 15
[]
```

Your solution:

```haskell
days :: Int -> Int -> [(Int, Int, Int)] -> [Int]
days year month dates = do
  let f = filter (\(y,m,d) -> y == year && m == month) dates
  let days = map (\(_,_,d) -> d) f
  days
```

Run

Problem 4

Write a `main` function that reads in a file whose path is passed as a command-line argument. It prints the number of lines in the file. For example:

```
> cat vowels.txt
A
E
I
O
U
> runhaskell main.hs vowels.txt
5
```

Your solution:

```haskell
import System.Directory.Internal.Prelude (getArgs)

main = do
  file : _ <- getArgs
  text <- readFile file
  let rows = lines text
  print $ length rows
```

Run

Problem 5

Define a `Distance` data type with two variants. `Meters` holds a distance in meters. `Feet` holds a distance in feet. All fields are of type `Double`.

Define also a `Metric` typeclass with a function `toMetric` that turns a value of an implementing type into its metric variant.

Make `Distance` implement the `Metric` typeclass. Consider 1 foot to be equivalent to 0.3048 meters. If the value is already metric, it is returned unchanged.

Your solution:

```haskell
data Distance = Meters Double | Feet Double

class Metric a where
  toMetric :: a -> a

instance Metric Distance where
  toMetric dist =
    case dist of
      Meters d -> Meters d
      Feet d -> Meters (d * 0.3048)
```

Run

Problem 6

Write function `braid` that accepts two lists of the same type. It returns a pair of lists that are like the parameters, but every other element has been swapped between the two lists. For example:

```
> braid "abcd" "1234"
("a2c4", "1b3d")
> braid "dime" "pulpit"
("dump", "pile")
```

Only elements that have a partner in the other list are included in the braid, as you can see in the second example.

Your solution:

```haskell
swap :: [a] -> [a] -> Int -> [a]
swap list1 list2 i = do
  if null list1 || null list2 then
    []
  else do
    if i `mod` 2 == 0 then
      (head list1) : (swap (tail list1) (tail list2) (i + 1))
    else do
      (head list2) : (swap (tail list1) (tail list2) (i + 1))

braid :: [a] -> [a] -> ([a],[a])
braid list1 list2 = do
  let newL1 = swap list1 list2 0
  let newL2 = swap list2 list1 0
  (newL1, newL2)
```

Run