Functional Programming Project Report

Burak Bekci - 150150127

17 - May 2019

1 Data

In the project a color can be represented as either RGB or HSV. A RGB triple has three integer values representing red, green and blue channels. A HSV value has three float values representing hue, saturation and value. I write a algebraic data type Color which can be either RGB or HSV in the given form:

```
data Color = RGB Int Int Int | HSV Float Float Float
```

I wrote a new Show method for the Color.

```
instance Show Color where
  show (RGB r g b) = "RGB ( " ++ show r ++ ", "++ show g ++ ", " ++ show b ++ " )"
  show (HSV h s v) = "HSV ( " ++ show h ++ ", "++ show s ++ ", " ++ show v ++ " )"
```

After these definitions the colors from w3schools is kept with a tuple of String and a Color. Some of these colors kept in the code is given below:

```
htmlColors :: [ (String, Color) ]
htmlColors  = [ ("AliceBlue", (RGB 240 248 255)), ("AntiqueWhite", (RGB 250 235 215)),
                ("Aqua", (RGB 0 255 255)), ("Aquamarine", (RGB 127 255 212)),
                ("Azure", (RGB 240 255 255)), ("Beige", (RGB 245 245 220)),
```

2 Functions

For the project, I created two files. All the functions asked in the documentation is written in ColorMain.hs file and the program that asks user input and operates written in main.hs file.The file main.hs imports ColorMain.hs and uses the functions of it.

2.1 Helper Functions

Apart from the asked functions in the documentation, I wrote some helper functions to make the code more readable and easier to interpret.

tripletOrd: Takes a ordering function (max, min) and 3 elements from the same type. Returns the result of two ordering function call.

```
tripletOrd :: Ord a => (a -> a -> a) -> a -> a -> a -> a
tripletOrd f x y z = f (f x y) z
```

deltaHelper: Used in hue calculation. Handles the $\frac{G'-B'}{\Delta}$, $\frac{B'-R'}{\Delta}$ and $\frac{R'-G'}{\Delta}$ equations.

```
deltaHelper :: Float -> Int -> Int -> Float
deltaHelper d x y = (fromIntegral (x - y) / 255.0) / d
```

calculateH: Calculates H value using the equations below:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

```
calculateH :: Float -> Int -> Int -> Int -> Float
calculateH 0.0 _ _ _ = 0
calculateH delta r g b
| cmax == r = 60.0 * (mod' (deltaHelper delta g b) 6.0)
| cmax == g = 60 * (deltaHelper delta b r + 2)
| cmax == b = 60 * (deltaHelper delta r g + 4)
where
    cmax = tripletOrd max r g b
```

calculateS: Calculates S value using the equations below:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

```
calculateS :: Float -> Float -> Float
calculateS 0    delta = 0
calculateS cmax delta = delta / cmax
```

normRGB: Handles $(R' + m) * 255$, $(G' + m) * 255$ or $(B' + m) * 255$ parts in the HSV to RGB convertion.

```
normRGB :: Float -> Float -> Int
normRGB x m = round ((x + m) * 255)
```

cxmToRGB: For given H value returns the write equation of RGB using m, c and x. Used in HSV to RGB convertion.

```
cxmToRGB :: Float -> Float -> Float -> Float -> Color
cxmToRGB c x m h
 | 0   <= h && h < 60  =  RGB (normRGB c m) (normRGB x m) (normRGB 0 m)
 | 60  <= h && h < 120 =  RGB (normRGB x m) (normRGB c m) (normRGB 0 m)
 | 120 <= h && h < 180 =  RGB (normRGB 0 m) (normRGB c m) (normRGB x m)
 | 180 <= h && h < 240 =  RGB (normRGB 0 m) (normRGB x m) (normRGB c m)
 | 240 <= h && h < 300 =  RGB (normRGB x m) (normRGB 0 m) (normRGB c m)
 | 300 <= h && h < 360 =  RGB (normRGB c m) (normRGB 0 m) (normRGB x m)
```

hueDesc, satDes, vDesc: They are used to convert HSV values into meaningful descriptions. Each of them maps given value to the corresponding strings.

2.2 Main Functions

The function asked to be implemented are using the above Helper Functions.

rgb2hsv: Takes an element of type RGB and returns an element of type HSV. Uses calculateH, calculateS and tripletOrd helper functions.

```
rgb2hsv :: Color -> Color
rgb2hsv (RGB r g b) = HSV (calculateH delta r g b) (calculateS cmax delta) cmax
where
    cmax = ( fromIntegral (tripletOrd max r g b) ) / 255.0
    cmin = ( fromIntegral (tripletOrd min r g b) ) / 255.0
    delta = cmax - cmin
```

hsv2rgb: Takes an element of type HSV and returns an element of type RGB. Uses cxmToRGB helper function.

```
hsv2rgb :: Color -> Color
hsv2rgb (HSV h s v) = cxmToRGB c x m h
  where
    c = v * s
    x = c * (1 - (abs  (mod' (h / 60) 2 - 1)))
    m = v - c
```

name2rgb: Takes a string and finds the RGB of the color in the html colors list. Uses linear search. In each function call it returns the RGB triple if names are match or it call itself with the rest of the list.

```
name2rgb :: String -> Color
name2rgb str = findName str htmlColors
```

```
  where
    findName :: String ->  [ (String, Color) ] -> Color
    findName str [] = error "No such color name"
    findName str (x:xs) = if str == fst x then snd x else findName str xs
```

hsvGradient: Takes two element of type HSV. Calculates the difference to be added in each iteration to the h, s and v values. Using the list comprehension finds h, s and v values for each iteration in seperate lists. Lastly uses zipWith3 method to combine those lists in HSV triples.

```
hsvGradient :: Color -> Color -> Int -> [Color]
hsvGradient (HSV h1 s1 v1) (HSV h2 s2 v2) step = zipWith3 HSV hs ss vs
  where
    hStep = ((h2 - h1) / (fromIntegral step))
    sStep = ((s2 - s1) / (fromIntegral step))
    vStep = ((v2 - v1) / (fromIntegral step))
    hs = [h1 + hStep* fromIntegral n | n <- [0..step] ]
    ss = [s1 + sStep* fromIntegral n | n <- [0..step] ]
    vs = [v1 + vStep* fromIntegral n | n <- [0..step] ]
```

hsv2desc: Takes an element of type HSV. Prints its description using satDes, vDesc and hueDesc helper functions. In order to make the function same with the one given in the homework documentation via GitHub, I converted the color representation from hsv to hsl.

```
hsv2desc :: Color -> IO()
hsv2desc (HSV h s v) =  putStrLn (hueDesc h ++ satDes s ++ lDesc l)
where
    RGB r g b = hsv2rgb $ HSV h s v
    l =  fromIntegral ((tripletOrd max r g b)
    + (tripletOrd min r g b)) / 510
```

3 The Program

The asked program is written in the main.hs file. It has a main function that asks user to give an starting and ending color names with a number of steps. The program prints the gradient steps. An example running is given below:

```
*Main> main
Enter a starting color name
Red
Enter a ending color name
Blue
Enter step number
5
HSV ( 0.0, 1.0, 1.0 ) RGB ( 255, 0, 0 ) red very saturated normal
HSV ( 48.0, 1.0, 1.0 ) RGB ( 255, 204, 0 ) yellow very saturated normal
HSV ( 96.0, 1.0, 1.0 ) RGB ( 102, 255, 0 ) green very saturated normal
HSV ( 144.0, 1.0, 1.0 ) RGB ( 0, 255, 102 ) green very saturated normal
HSV ( 192.0, 1.0, 1.0 ) RGB ( 0, 204, 255 ) cyan very saturated normal
HSV ( 240.0, 1.0, 1.0 ) RGB ( 0, 0, 255 ) blue very saturated normal
```

4 Closest Color

In the documentation of the homework, it is asked to find a name of the color which is the closest to given color. In order to calculate the distances between colors, I used HSV representations. I summed the absolute differences between h,s and v values. In order to make the differences in the same range the difference of h values divided by 360. The code for calculating the difference is given below:

```
calculateHSVDist :: Color -> Color -> Float
calculateHSVDist (HSV h s v) (HSV h2 s2 v2) = (abs (h - h2) ) / 360 +
                                               (abs (s - s2) ) +
                                               (abs (v - v2) )
```

Then I used foldl1 with a custom function to traverse whole htmlColors and calculate the differences with the given color. The custom function is named whichOneCloser. It takes one Color element and two htmlColors element and returns the htmlColor element which is closer to Color element. The implementation is given below:

```
whichOneCloser :: Color -> (String,Color) -> (String,Color) -> (String,Color)
whichOneCloser (RGB r1 g1 b1) (str, (RGB r2 g2 b2)) (str2, (RGB r3 g3 b3)) =
  if calculateHSVDist (rgb2hsv $ RGB r1 g1 b1) (rgb2hsv $ RGB r2 g2 b2) >
    calculateHSVDist (rgb2hsv $ RGB r1 g1 b1) (rgb2hsv $ RGB r3 g3 b3)
    then (str2, (RGB r3 g3 b3))
    else (str, (RGB r2 g2 b2))
whichOneCloser (HSV h s v)  (str, (HSV h2 s2 v2)) (str2, (HSV h3 s3 v3)) =
  if calculateHSVDist (HSV h s v) (HSV h2 s2 v2) >
    calculateHSVDist (HSV h s v) (HSV h3 s3 v3)
  then (str2, (HSV h3 s3 v3))
  else (str, (HSV h2 s2 v2))
```

The main function for finding the closest color is findClosest implemented like below:

```
findClosest :: Color -> String
findClosest (RGB r g b) =  fst $ foldl1 (whichOneCloser $ RGB r g b) htmlColors
findClosest (HSV h s v) =  fst $ foldl1  (whichOneCloser $ hsv2rgb $ HSV h s v) htmlColors
```

Since all htmlColor elements are in RGB representation findClosest converts given HSV elements to RGBs. An example output is given below:

```
*Main> findClosest $ RGB 12 128 128
"Teal"
*Main> findClosest $ RGB 12 128 12
"Green"
*Main> findClosest $ HSV 122 0.42 0.12
"DarkSlateGray"
```