COMPX307-20B Programming Languages

Second Haskell coursework

1. Write a function

```
halve :: [a] -> ([a],[a])
```

that splits an even-length list into two equal-length halves and an odd-length list into two parts where one part is one longer than the other , so:

```
halve [1,2,3,4,5,6] = ([1,2,3],[4,5,6])

halve [1,2,3,4,5,6,7] = ([1,2,3],[4,5,6,7])
```

(10 marks)

2. Define a function

```
meld :: [Int] -> [Int] -> [Int]
```

which composes two *sorted* lists of integers together, to get a sorted list which contains all the elements from the two original lists, so:

```
meld [3,6,9] [1,8,10] = [1,3,6,8,9,10]
```

The definition should use explicit recursion and no pre-defined sorting functions.

(10 marks)

3. Use the function `meld` from question two to define a function

```
integerSort :: [Int] -> [Int]
```

which sorts a list of integers. The empty list and a singleton list are already sorted, and any other list should be sorted by composing together the two lists that result from sorting the two halves of the list separately.

(10 marks)

4. Define, using a list comprehension (and not explicit recursion), a function `replicate :: Int -> a -> [a]` that produces a list of a given length consisting of identical items. For example

```
replicate 3 'a' = ['a', 'a', 'a']
```

(5 marks)

5. Show how `[f x | x <- xs, p x]` can be expressed using only `map`, `filter` and composition, `(.)`. You should not use explicit recursion or list comprehension, in particular.

(15 marks)

6. Using `foldl`, define a function `d2i :: [Int] -> Int` that converts a list of decimal digits into an integer. For example:

```
d2i [1,2,3,4] = 1234
```

(10 marks)

7. Why do the following expressions not type-check?

(i) `[`a`,1,2]`
(ii) `[(*), 0, (+)]`
(iii) `[(1,True),(`2`,False)]`
(iv) `length (1,2,3,4)`

(5 marks)

8. Using the definition of trees given by

```
data Tree a = Node (Tree a) (Tree a) | Leaf a
```

define tree versions of the list functions `zip` and `zipWith`. There will be cases at the leaves or where trees are of different shapes where you'll have to make design decisions. Try to make your decisions as elegant as possible (of course)…or ask me if you get stuck!

(20 marks)

9. Here is a datatype `Expr`:

```
data Expr = C Float | Expr :+ Expr | Expr :- Expr | Expr :*
Expr
                | Expr :/ Expr
```

and here is a function to evaluate expression of type `Expr`:

```
evaluate :: Expr -> Float
evaluate (C x)      = x
evaluate (e1 :+ e2) = (evaluate e1) + (evaluate e2)
evaluate (e1 :- e2) = (evaluate e1) - (evaluate e2)
evaluate (e1 :* e2) = (evaluate e1) * (evaluate e2)
evaluate (e1 :/ e2) = (evaluate e1) / (evaluate e2)
```

(i) Adapt the data type by allowing variables and let expressions (rather like Haskell's "where"), though the let expression should not allow recursive definitions (e.g. things like "let a = a + 1" where the name "a" being introduced also appears on he right-hand side in "a + 1").

(5 marks)

(ii) Extend the definition of `evaluate` above to work on this extended type of expression. For example

```
evaluate (Let "n" (C 42) (V "n" :+ C 2))
```

reduces to the value `44`. I.e. the expression is saying: "let n be 42 in n+2".

(10 marks)

Your answers for all the questions above are due at 1000 on Friday 20th August 2021.
You must submit your answers as a plain text file with suffix .hs via Moodle. This **MUST** be a plain text file (**not** a PDF, **not** a MS-format file or any other sort of particular format) since we will want to load your solutions and try them out in ghci. Put non-code answers as free text between Haskell comment symbols {-…….-}, or write a literate Haskell script (with suffix .lhs).

Please also put your name and number at the top of the file.