Assignment Two: Haskell Lists and Higher-order functions

Due: 11:59pm, Tuesday 23 January 2024.

You may do this question either alone or with a partner (i.e., in a group of 1 or 2). Make sure you both understand and can explain your solution.

Submit your answers in a text file using Canvas. Make sure you name(s), student number(s) is at the top of each file. The file should run with Haskell Platform. Your code needs comments that include the specification of the meaning of all functions. The explanations should be comments in the Haskell file. Both members of a pair should submit the same file.

Question One

(a) Consider the program

```
tails [] = [[]]
tails (e:r) = (e:r):tails(r)
```

i) What is the type of `tails`

ii) What is the value of `tails "happy"`

iii) What is the type of `tails "happy"`

iv) Can `tails` be implemented by foldr or foldl with lambda and : (and no recursion)? If so, give a definition (call it `tails1`), if not, explain why not.

(b) Consider the definition:

```
doif f g [] = []
doif f g (h:t)
   | f h = g h : doif f g t
   | otherwise = h : doif f g t
```

i) What is the inferred type of `doif`?

ii) what is the value of

```
doif even (‘div‘ 2) [11,22,33,44,55,66]
```

iii) Using doif, assuming only the functions elem (which tests whether a value is an element of a list), and toUpper (which makes a character into upper-case, defined below), write a non-recursive function capvowel that takes a string capitalizes all vowels (a,e,i,o,u), with the following behaviour:

```
toUpper :: Char -> Char
toUpper x = toEnum( fromEnum x - fromEnum ’a’ + fromEnum ’A’)
*Main> capvowel "abcdefghij"
"AbcdEfghIj"
*Main> capvowel [’a’..’z’]
"AbcdEfghIjklmnOpqrstUvwxyz"
```

iv) Implement doif using list comprehensions. You may only also use lambda (\) and guards (or if-then-else), and constructors : and []. Call it doif1.

v) Implement doif using either foldl or foldr. You may only also use lambda (\) and guards (or if-then-else), and constructors : and []. Call in doif2.

Question Two

Implement the following *without using recursion*, but using only list comprehensions, foldr and/or foldl. You can also use `tails` (from Question 1), lambda, guards, if-then-else, functions defined in Bool, Num, Fractional, Integral, Eq, Ord, as well as list functions elem, head, tail, !!. Include the most general type that makes sense.

Groups of two should do four parts. Groups of one should do three parts. (Everyone is encouraged to try all parts, but not all parts are equally straightforward).

(a) Implement *harmonic* from the previous assignment.

(b) Implement *myremoveduplicates* which removes duplicates from a list. For example:

```
myremoveduplicates "abacad" => "bcad"
myremoveduplicates [7,3,2,1,3,2,2,1,1,3,7,8] => [2,1,3,7,8]
```

(c) Implement *myordered lst* that is True if list *lst* is ordered (by $\leq$).

```
myordered "abcdeefg" => True
myordered "ba" => False
```

(d) *myreplace x y lst* replaces all occurrences of x in list *lst* with y. For example,

```
myreplace 7 3 [7,0,7,1,7,2,7,3] => [3,0,3,1,3,2,3,3]
myreplace 'a' 'x' "" => ""
myreplace 'a' 'x' "abacad" => "xbxcxd"
```

(e) Implement *myapply lst sub* where sub is a list of (x, y) pairs, replaces each occurrence of x by y in *lst*.

```
myapply "abcdec" [('a','f'), ('c','3'), ('g','7')] =>  "fb3de3"
myapply "baab" [('a','b'), ('b','a')] => "abba"
```

Question Three

For each question, specify how long you spend on it, and what you learned. Was the question reasonable? (This question is part of the assignment, so please do it!)