CPSC 449:
Sample question for the Haskell Lab. Exam
(for discussion in labs!)

October 14, 2022

1. Show (by hand) how Haskell evaluates:

```
myappend [1,2] [3,4,5,6]
```

   given the code:

```
myappend:: [a] -> [a] -> [a]
myappend [] bs = bs
myappend (a:as) bs = a:(myappend as bs)
```

2. Show (by hand) how Haskell evaluates:

```
sfList [SS 3,SS 4]
```

   Given the code:

```
data SF a = SS a | FF

sfList:: [SF a] ->  SF [a]
sfLIst [] = SS []
sfList (FF:_) = FF
sfList ((SS a):xs) =
        case (sfList xs) of
                SS as -> SS (a:as)
                FF -> FF
```

3. Show (by hand) how Haskell evaluates:

```
myOR [False,True,False]
```

Given the code:

```
foldr:: (a -> b -> b) -> b  -> [a] -> b
foldr f b [] = b
foldr f b (x:xs) = f x (foldr f b xs)

myOR:: [Bool] -> Bool
myOR = foldr myor False

myor:: Bool -> Bool -> Bool
myor  False False = False
myor _ _ = True
```

4. Show (by hand) how Haskell evaluates:

```
unzip [(1,'a'),(2,'b')]
```

given the code:

```
unzip:: [(a,b)] -> ([a],[b])
unzip [] = ([],[])
unzip ((a,b):xs) = case unzip xs of
          (as,bs) ->  (a:as,b:bs)
```

5. Answer the questions concerning Haskell syntax below:

   (a) Which of the types below are the same type:

      i. `a -> b -> c -> d`
      ii. `(a -> b) -> c -> d`
      iii. `a -> (b -> c) -> d`
      iv. `a -> b -> (c -> d)`
      v. `(a -> b) -> (c -> d)`
      vi. `((a -> b) -> c) -> d`
      vii. `a -> (b -> (c -> d))`

   (b) In the following terms what are the types of the functions `f` given that `x,y,z:: Integer`:

      i. `(f x) (y, z)`
      ii. `f x y z`
      iii. `f (x,y,z)`
      iv. `f (x,(y,z))`

   (c) Give the types of the following terms (if indeed they type) and indicate which are equal:

      i. `"abcd"`
      ii. `[('a','b'),('c','d')]`
      iii. `('a':['b']):('c':['d'])`
      iv. `'a':('b':'c':'d':[])`
      v. `["ab","cd"]`

6. Given the following programs do they type and what is their most general type?

   (a) Given that `myfoldr: b -> (a -> b -> b) -> [a] -> b` give the type of `f` in:

```
f = myfoldr id g where
     g a h x =  h (a:x)
```

   (b) Given that `mymap:: (a -> b) -> [a] -> [b]` give the type of `f` in:

```
f = mymap (mymap (mymap h))
```

   (c) Given `mymap` as above what is the type of `f` in

```
f =  mymap mymap
```

7. Given a list of items and a predicate write code to split the list into two lists: a list of item which satisfies the predicate and a list of items which does not:

```
split_list:: (a -> Bool) -> [a] -> ([a],[a])
```

8. In a merge sort one step is to merge two ordered lists of items into one ordered list. Write the code for this step

```
merge:: (Ord a) => [a] -> [a] -> [a]
```

9. Write the code to split a list into two lists such that the elements with odd index are in one list while the elements with even index are in the other list:

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

   Can you write this using a `foldl`?

10. Write a function to determine whether a string is a substring of another string:

```
substring :: String -> String -> Bool
```

11. Write a function

```
grow :: String -> String
```

   which changes a string $a_1a_2a_3...$ to $a_1a_2a_2a_3a_3a_3...$ so grow "now!" == "noowww!!!".

12. Write a function to produce the list of all the sublists of a list.

13. Why is the following "naive" function for reversing a list $\mathcal{O}(n^2)$:

```
reverse: [a] -> [a]
reverse [] = []
reverse (a:as) = (reverse as) ++ [a]
```

Give a "fast" $\mathcal{O}(n)$ version of reverse.

14. Why is the complexity of the following program $\mathcal{O}(\mathsf{Fib}(n))$ (assuming a positive input!):

```
fib 0 = 0
fib 1 = 1
fib n = (fib n-1) + (fib (n-2))
```

Give a $\mathcal{O}(n)$ version of `fib`.

15. A programmer writes the following code but fails to provide types:

```
data SF a = SS a | FF
    deriving (Show,Eq)

myhead [] = FF
myhead (a:as) = SS a

mytail [] = []
mytail (_:xs) = xs

myzip [] _ = []
myzip _ [] = []
myzip (a:as) (b:bs) = (a,b):(myzip as bs)

mystery xs =
   myhead [x|(x,y) <- myzip xs (mytail xs),x==y]
```

(a) Provide the types for `myhead`, `mytail`, `myzip`, and `mystery`.

(b) Explain what `mystery` does: what is the result of `mystery "abccdeffghii"`?

(c) (Bonus) Can you rewrite the code using a `foldr`?

16. A programmer writes the following code but fails to provide a type:

```
mycode f g [] = ([],[])
mycode f g (a:as) = case (mycode f g as) of
          (xs,ys) -> ((f a):xs, (g a):yx)
```

(a) What is the most general type of the code?

(b) What is the result of evaluating the following

```
mycode (\a -> a+6)
          (\b -> b `mod` 2 \= 0)
             [3,7,10,2,9,17]
```

(c) (Bonus) Rewrite `mycode` using a `foldr`.

17. Prove by structural induction (assuming finite lists) given:

```
append:: [a] -> [a] -> [a]
append [] ys = ys
append (x:xs) ys = x:(append xs ys)

flatten [] = []
flatten (xs:xss) = append xs (flatten xss)

filter p [] = []
filter p (a:as) | p a = a: (filter p as)
                      | otherwise = filter p as

map f [] = []
map f (x:xs) = (f x):(map f xs)

foldr x f [] = x
foldr x f (c:cs) = f c (foldr x f cs)
```

that

(a) `append x (append y z) = append (append x y) z`

(b) `append x y = foldr y (:) x`

(c) `filter p (append x y) = append (filter p x) (filter p y)`

(d) `filter p (filter q x) = filter q (filter p x)`

(e) `filter p (flatten xss) = flatten (map (filter p) xss)`

(f) `flatten (flatten xsss) = flatten (map flatten xsss)`

(g) `flatten = foldr [] append`

18. Who are the following people? When did they live? What did they do?

    (a) Basile Bouchon
    (b) Jean-Baptiste Falcon
    (c) Jacques de Vaucanson
    (d) Joseph Marie Jacquard
    (e) Napoleon Bonaparte
    (f) Charles Babbage
    (g) Ada Lovelace
    (h) Herman Hollerith
    (i) Paul Otlet
    (j) George Stibitz
    (k) Konrad Zuse
    (l) Alan Turing
    (m) Alonzo Church
    (n) Haskell Curry
    (o) Greta Thunberg