Higher Order Function Exam Solutions

1 2021 Q1

1.1 Part (a)

```
hof :: (a -> b -> c) -> [a] -> [b] -> [c]
hof op (x:xs) (y:ys) = (x `op` y) : hof op xs ys
hof op _ _ = []
```

1.2 Part (b)

```
f1 = hof (+)
f2 = hof (++)
f3 = hof (*)
f4 = hof (\x _ -> (x+42))
f5 = hof (\x y -> (y-x*x))
```

1.3 Part (c)

- zipWith
- Checking the type signatures can be blatant
- Manually working through the functions can be blatant

```
hof :: (a -> b -> c) -> [a] -> [b] -> c
hof (+) [1,2,3] [4,5,6]
hof (+) (1 + 4) : hof [2,3] [5,6]
hof (+) (1 + 4) : (2 + 5) : hof [3,6]
hof (+) (1 + 4) : (2 + 5) : (3 + 6) : hof []
hof (+) (1 + 4) : (2 + 5) : (3 + 6) : []

zipWith :: (a -> b -> a) -> [a] -> [b] -> c
zipWith (+) [1,2,3] [4,5,6]
zipWith (+) (1 + 4) : hof [2,3] [5,6]
zipWith (+) (1 + 4) : (2 + 5) : hof [3,6]
zipWith (+) (1 + 4) : (2 + 5) : (3 + 6) : hof []
zipWith (+) (1 + 4) : (2 + 5) : (3 + 6) : []
```

2 2021 Q1

2.1 Part (a)

```
hof :: (a -> b -> a) -> a -> [b] -> a
hof op x [] = x
hof op x (y:ys) = hof op (x `op` y) ys
```

2.2 Part (b)

```
f1 = hof (*)
f2 = hof (||)
f3 = hof (\a x -> 2*a+x)
f4 = hof (\xs ys -> ys ++ xs)
f5 = hof (-)
```

2.3 Part (c)

- foldl
- Checking the type signatures can be blatant
- Manually working through the functions can be blatant

```
hof :: (a -> b -> a) -> a -> [b] -> a
hof (+) 1 [1,2,3]
hof (+) (1 + 1) [2,3]
hof (+) ((1 + 1) + 2) [3]
hof (+) (((1 + 1) + 2)+ 3) []
hof (+) (((1 + 1) + 2)+ 3) = 7

foldl :: (a -> b -> a) -> a -> [b] -> a
foldl (+) 1 [1,2,3]
foldl (+) (1 + 1) [2,3]
foldl (+) ((1 + 1) + 2) [3]
foldl (+) (((1 + 1) + 2)+ 3) []
foldl (+) (((1 + 1) + 2)+ 3) = 7
```

3 2019 Q3

3.1 Part (a) (i)

```
hof :: (a -> b -> b) -> b -> [a] -> b
hof op e [] = e
hof op e (x:xs) = x `op` hof op e xs
```

3.2 Part (b) (ii)

```
f1 = hof 42 (*)
f2 = hof 0 (\x y -> 99 * y)
f3 = hof 0 (+)
f4 = hof [] (++)
f5 = hof 0 (\x xs -> (x-42) + xs)
```

4 2017 Q2

- Same Answer as in 2019 Q3

5 2016 Q2

5.1 Part (a)

```
hof :: (a -> b -> b) -> b -> [a] -> b
hof op e [] = e
hof op e (x:xs) = x `op` hof op e xs
```

5.2 Part (b)

```
f1 = hof (*) 1
f2 = hof (\_ xs -> 1 + xs) 0
f3 = hof (+) 0
f4 = hof (++) []
f5 = hof (\x xs -> (x*x) + xs) 0
```

6 2015 Q2

6.1 Part (a)

```
hof :: (a -> b -> c) -> [a] -> [b] -> [c]
hof op [] _ = []
hof op _ [] = []
hof op (x:xs) (y:ys) = (x `op` y) : hof op xs ys
```

6.2 Part (b)

```
hof :: (a -> b -> c) -> [a] -> [b] -> [c]
```

6.3 Part (c)

```
f1 = hof (*)
f2 = hof (+)
f3 = hof (\x y -> (x y))
f4 = hof (\x y -> (y,x))
f5 = hof (\x y -> (const x y))
```

6.4 Part (d)

- zipWith

7 2014 Q2

7.1 Part (a)

```
hof :: (a -> b -> a) -> a -> [b] -> a
hof op x [] = x
hof op x (y:ys) = hof op (x `op` y) ys -- Don't forget to give op pleasesee
```

7.2 Part (b)

```
f1 = hof (*)
f2 = hof (\x y -> x + 1)
f3 = hof (+)
f4 = hof (\x y -> x ++ y)
f5 = hof (\x y -> x + y * y)
```

7.3 Part (c)

- foldl

8 2016 Q2

- Same Answer as in 2019 Q2 for parts (a) and (b)

8.1 Part (c)

- foldr
- Checking the type signatures can be blatant
- Manually working through the functions can be blatant

```
hof :: (a -> b -> b) -> b -> [a] -> b
hof (+) 1 [1,2,3,4]
hof (+) 1 * [2,3,4]
hof (+) (1 * 2) [3,4]
hof (+) (1 * (2 * 3)) [4]
hof (+) (1 * (2 * (3 * 4))) []
hof (+) (1 * (2 * (3 * 4))) = 24

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr (+) 1 [1,2,3,4]
foldr (+) 1 * [2,3,4]
foldr (+) (1 * 2) [3,4]
foldr (+) (1 * (2 * 3)) [4]
foldr (+) (1 * (2 * (3 * 4))) []
foldr (+) (1 * (2 * (3 * 4))) = 24
```