Exam 2 Corrections

Logan Montgomery

1.)

triangleNumber :: Int -> Int

triangleNumber 0 = 0

triangleNumber n = n + triangleNumber (n-1)

On the exam I had the recursive call look like this triangleNumber(n+n-1)). I thought I could add n and also call triangleNumer of n-1 in one call but clearliy this doesn’t work, I must add n to the triangle number of n-1.

2.)

triangleNumber' n = foldr (+) 0 [1..n]

I forgot the “0” needed to be in there to fulfil all arguments of the foldr function, also didn’t have the “n” before the “=”.

3.)

repeats3 :: Eq a => [a] -> Bool

repeats3 [] = False

repeats3 [\_] = False

repeats3 [\_,\_] = False

repeats3 (x:y:z:xs) = if x == y && y == z then True else repeats3 (z:xs)

I did not complete this problem on the exam. I Had only created the type declaration and the first base case

4.)

filter' :: (a -> Bool) -> [a] -> [a]

filter' p xs = foldr (\x xs -> if p x then x:xs else xs ) [] xs

On the exam I had tried to use map, later I realized that filter is best thought of using a fold, I used foldr and followed the pattern in the book outlined for how filter is defined with p and xs

5.)

sumpairs :: [(Int,Int)] -> [Int]

sumpairs = map (\ (x,y) -> x + y )

For some reason I put “lst” after the lambda function and I don’t know why...

6.)

append :: [a] -> [a] -> [a]

append xs ys = foldr (:) ys xs

On the exam I attempted to create a lambda function to take in the ‘xs’ and ‘ys’ and do a ‘:’ operator on the head on ys to the tail of xs. Insead I should have apllied the ‘:’ to each list like I have here in the foldr.