CS 360

Lab two
Programming Languages: Haskell
Learn You a Haskell for Great Good

---

1. All Haskell labs must use the main given. **In Canvas right click on the main.hs file to download.**
2. IMPORTANT! – When I click on your .hs file it will compile open in a ghci environment and compile. At the prompt I will enter main and all your code should run and produce the correct answers. Labs failing to compile or run from the main will not be graded.
3. While you are reading, have a session in GHCI open. For each and every concept you read about, try it out. This means typing in many examples from the book and trying out your own versions. Just to confirm, yes type it yourself, do not copy and paste! You'll store it better in your brain if you're the one typing it. And when you are trying out your own versions, try to think of other things to try. Be inquisitive. For example, the book says that GHCI will yell at you for trying to evaluate `5 * -3`. But `-3 * 5` works just fine. Why? Not all your examples should work, that's part of learning the syntax.

Recursion

Use recursion as the primary method of solving these problems. Beyond that it is your choice whether you want to use let expressions, where clauses, guards, case expressions, if-then-else expressions, etc.

1. Implement the *greatest common divisor algorithm* as

```
gcdMine :: Integral a => a -> a -> a
```

by following the definition here. Check that your answer matches the built-in algorithm with this function (you **MUST** include gcdCheck below in your code):

```
gcdCheck x y = (myAnswer, correctAnswer, comment)
    where
    myAnswer     = gcdMine x y
    correctAnswer = gcd x y
    comment      = if myAnswer == correctAnswer then "Matches" else "Does not Match"
```

e.g.

```
gcdCheck 111 259 == (37,37,"Matches")
gcdCheck 2945 123042 == (1,1,"Matches"))
gcdCheck (2*5*7) (7*23) == (7,7,"Matches")
```

2. Write a function to compute Fibonacci numbers. Follow the description given at the beginning of the chapter.

```
fibonacci :: Int -> Int
```

When finished check that you get this:

```
[fibonacci n | n <- [0..20]] ==
[0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765]
```

3. Write a function that counts how many items are found in a list that match the one given:

```
count :: (Eq a, Num b) => a -> [a] -> b
```

You should be able use it like this:

```
count 7 [1,7,6,2,7,7,9] == 3
count 'w' "western oregon wolves" == 2
```

4. Rewrite your answer for Assignment 1 question 13 (sanitize) using recursion. You can't use a list comprehension and you can't use the concatfunction.

Higher-order functions I

Use one or more of the following functions to solve the following problems. You can use other functions too of course, but these questions are to give you practice with these specifically so make sure one of them is in there somewhere. Not all must be used. (A good place to look these up is Hoogle)

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
takeWhile :: (a -> Bool) -> [a] -> [a]
dropWhile :: (a -> Bool) -> [a] -> [a]
flip :: (a -> b -> c) -> b -> a -> c
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
any :: (a -> Bool) -> [a] -> Bool
all :: (a -> Bool) -> [a] -> Bool
```

Write an expression for each that uses higher order functions and **not** list comprehensions. For some you may need an additional function. Either write it separately or use a lambda expression.

5. Multiply every element of a list by 10
6. Increment every element in a list. Write it again so that it can increment things that are not numbers, i.e. turn "Hello" into "Ifmmp"
7. Tell you True or False whether or not a list contains a value that is divisible by 42
8. Use zipWith to take a list like this [5,3,8,2,3,6,3] to this [100000,1000,100000000,100,1000,1000000,1000] i.e. it raises 10 to the power of the element in the list and places it in that list. HINT: You shouldn't have to ask how long the list is (use an infinite list!).
9. Remove all spaces from the end of a string (often called stripping a string).
10. Tell you True or False if a list contains nothing but even numbers.

11. Put the word "not" on the front of all strings in a list. e.g. ["Funny","cold","slow"] ==> ["not Funny","not cold","not slow"]
12. Reverse all strings contained in a list. e.g. ["This","is","a","sentence"] ==> ["sihT","si","a","ecnetnes"]

Rewrite as lambda functions

13. plus x y = x + y
14. (*4)
15. A function that gives you the second element in a list.
16. A function that takes the square root of a number and then rounds it.
17. words "This is a sentence written in the usual way."
    == ["This","is","a","sentence","written","in","the","usual","way."]
18. Use a Lambda expression and map to take a list of tuples and produce a list of values. The list contains the lengths of two sides of right triangles, a and b. You want to produce a list that contains the lengths of all three sides, where the third side, c, is found with the Pythagorean theorem.

```
[(3,4),(5,16),(9.4,2)]
```

```
== [(3.0,4.0,5.0),(5.0,16.0,16.76305461424021),(9.4,2.0,9.610411021387172)]
```

If it helps, write it first by defining a function separately and once it is working, rewrite it as a lambda expression.

Higher-order Functions II & Modules

Please write your answers to questions 1 through 4 within a module called Assn2b in a file called Assn2b.hs, where you export the functions requested. Make sure you can import your module and use your functions from the GHCI prompt.
Like this:

```
module Assn2b
(
--comma separated list of exported function names here
) where
```

```
Imports and function definitions follow..
```

1. Trace out the execution of each expression and show how they are different. Put your answer in your code commented out.

```
foldl (*) 6 [5,3,8]
foldr (*) 6 [5,3,8]
```

2. Write your own version of length

```
length :: [a] -> Int
```

using a fold. Export this function from your module.

3. Write two functions, one that uses a left fold and one that uses a right fold, to imitate the behavior of this map

```
map intToDigit [5,2,8,3,4] == "52834"
```

Call your functions

```
convertIntToStringLeft :: [Int] -> [Char]
convertIntToStringRight :: [Int] -> [Char]
```

Export these functions. To do the conversion use

```
intToDigit :: Int -> Char
```

from Data.Char

4. Rewrite **two of** the following expressions with as few parentheses as possible using the function application operator, $, and/or function composition.

```
length (filter (<20) [1..100])
take 10 (cycle (filter (>5) (map (*2) [1..10])))
sum (map length (zipWith (flip (++)) ["love you", "love me"] ["i ", "you "]))
sum (map (\x -> sin (sqrt (abs x))) [1..100])  -- write this one without the lambda function
```

Import Assn2b into your Assn2.hs file. Call all of the functions from Assn2b in Assn2.hs

Turn In: Your .hs file (this must be your code file, NOT a .txt, .doc, or .pdf!)