# CSCE330-002 Fall 2018 Final Exam

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rules:

* READ these instructions and the problem instructions carefully.
* Using Standard Prelude functions (**take**, **drop**, **head**, **length, concat**…) is fine unless otherwise noted.
* You can *always* define a function to help, just make sure to satisfy the problems constraints somewhere (e.g. use recursion or a guarded equation). It’s fine if the helper function satisfies the constraints (e.g. it does the actual recursion).
* If you cannot answer a question the way it requested (e.g. “using recursion”, “using a higher order function”, then penalty for an alternative method will be approximately 50%.
* Higher order function, here, will be possible with **map**, **filter**, or **foldr**. You will often need to post-process the return of the function or preprocess the input. Standard Prelude functions are highly desirable.

1. (2) COBOL is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-typed, meaning that variable names and their associated types are bound at compile time.
2. (3) Lisp has a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, which kept track of data being used and freed it when it was no longer needed, *automatically*. Java also uses one.
3. (2) Haskell is a particularly strong example of a programming language that uses the \_\_\_\_\_\_\_\_\_\_\_\_\_\_ programming paradigm.
4. (3) Alonzo Church developed \_\_\_\_\_\_\_\_\_\_\_\_ Calculus which is important for understanding how Haskell functions work in many ways.
5. (5) Give the type of the following function (make the type as generic as possible, i.e. don’t over constrain the acceptable inputs).

**f :: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**f x y = x == y**

1. (5) Give the type of the following function (make the type as generic as possible, i.e. don’t over constrain the acceptable inputs):

**g:: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**g (x,y) s = (x\*s , y\*s)**

1. (5) What is the type of the following value, i.e. what would **:t** return?

**( [ "Kale" , "chili"] , "Jim" )**

1. (5) Define, with **pattern matching** a function  **fst**, which returns the first item in a tuple, e.g.

**fst ("first", "second")** returns "first"

**fst :: (a,b) -> a**

1. (5) Define, with a **conditional expression**, an **even** function that returns True if the input is even, False otherwise.

**even :: Integral a => a -> Bool**

1. (5) Give a **lambda expression** that returns the second item of a list. For clarity, if you defined a **second** function that did the same thing, **second [2,5,6,7,8]** would return **5**. You may define **second** with your answer, if you choose.

**second :: [a] -> a**

**second =**

1. (5) Give a **lambda expression** that returns the nth even natural number, starting at 0. The 0th natural number is 0, the 1st is 2, etc.

**nthEven :: Int -> Int**

**nthEven =**

1. (5) Define a **justOnes** function using a **list comprehension** which reads in bits, as a String, and just returns a String having all the ones in the input, but none of the ‘0’s, e.g. **justOnes** **"100101"** returns "**111**"

**justOnes :: String -> String**

1. (5) Define the **nthMultiple** function with a **list comprehension** that takes as input a number and an **Int** and returns the nth multiple. You may use Prelude function **sum**. E.g. **nthMultiple 4 1** returns **4** and **nthMultiple 4 3** returns **12 .**

**nthMultiple :: Num a => a -> Int -> a**

1. (10) Define, **using recursion**, a Boolean/logical and function, **bwAnd,** that “ands” two binary numbers, in String form where the only characters are ‘0’s and ‘1’s, e.g. **bwAnd "101" "111"** returns **"101"**, **bwAnd "101" "010"** returns **"000".** You may assume the strings are the same length.

**bwAnd :: String -> String -> String**

1. (10) Define, with **recursion**, a **dupSepBy1** function that returns True if the single input, a list, has two duplicates separated by exactly one item, false otherwise. Make sure it works with all possible list lengths. For example: **dupSepBy1 [1..10]** returns **False** while **dupSepBy1 [1,2,3,2,1]** returns **True** but **dupSepBy1 [1,2,3,3,2,1]** returns **False**.

**dupSepBy1 :: Eq a => [a] -> Bool**

1. (5) Define an appropriate type for students, where students have first names, last names, and a student number.

**type Student =**

1. (5) Define an appropriate type for numeric dates, e.g. 2018/12/14.

**type NumDate =**

1. (5) Define, with one or more **higher order functions**, a function that tells you whether the string “Waldo” is anywhere in a list of strings. Examples:

**\*Main> isWaldoThere ["fred", "ted", "ug"]** returns **False \*Main> isWaldoThere ["fred", "ted", "ug", "Waldo"]** returns **True \*Main> isWaldoThere ["fred", "Waldo", "ted", "ug"]** returns **True**

**isWaldoThere :: [String] -> Bool**

**isWaldoThere =**

1. (10) Define, with one or more **higher order functions**, a function **bitFlip** which flips which reads in bits, as a String, and flips all **1**s to **0**s and vice versa, e.g. **bitFlip "100101"** returns "**011010**"

**bitFlip :: String -> String**

**bitFlip bs =**