CS 403/503 Exam 2 **Spring 2018** **Name ___________________**

Write each Haskell function as described below, and also show the type of each function.
You may show either the inferred type, or a declared type that suffices for all the given examples.
Write any non-standard helper functions that you use, but you don't need to show their types.

1. (fun f g p id list) determines which values in the list satisfy predicate p, then applies unary function g to each such value, and then combines all the results using binary function f. However, if no values satisfy p, then return id. Examples:
   fun (+) (^4) (\x -> x>0 && odd x) 0 [2,3,-2,6,5,-3] returns $3^4 + 5^4 = 706$.
   fun (:) head (not . null) [ ] ["abcd","","efg","hij","klm","","nopq"] returns "aehkn".
   fun (++) tail (not . null) [ ] ["abcd","","efg","hij","klm","","nopq"] returns "bcdfgijlmopq".

   fun :: ____

2. (isPermutation xs ys) returns True if lists xs and ys are permutations of each other, and otherwise returns False. Examples:
   isPermutation [1,2,3,4] [2,4,1,3] returns True.
   isPermutation [1,2,3,4] [3,2,1,0] returns False.
   isPermutation "abcdef" "fcebda" returns True.

   isPermutation :: ____

3. (equiv n) returns a list of the equivalence classes of the non-negative integers modulo n, where each of the n sublists is an infinite list. Example: equiv 5 returns [[0,5,10,15,20,…],[1,6,11,16,21,…],[2,7,12,17,22,…],[3,8,13,18,23,…],[4,9,14,19,24,…]].

   equiv :: ____

4. Haskell's let construct is semantically equivalent to Scheme's letrec. That is, the following two expressions are equivalent:

   Scheme: (**letrec** ((x_1 e_1) (x_2 e_2) … (x_n e_n)) e)
   Haskell: let { x_1=e_1; x_2=e_2; … x_n=e_n; } in e

   For each Scheme expression below, write an equivalent Haskell expression using anonymous functions (lambdas):

a. Scheme: (**let** ((x_1 e_1) (x_2 e_2) … (x_n e_n)) e)

   Haskell:

b. Scheme: (**let*** ((x_1 e_1) (x_2 e_2) … (x_n e_n)) e)

   Haskell:

5. **CS 403:** (evalpoly p v) takes a polynomial with coefficients given in list p, and evaluates it at v. Example: The list [2,3,5,6,8,9] denotes polynomial $2 + 3x + 5x^2 + 6x^3 + 8x^4 + 9x^5$, therefore evalpoly [2,3,5,6,8,9] 10 returns $2 + 3*10 + 5*10^2 + 6*10^3 + 8*10^4 + 9*10^5 = 986532$.

   CS 503: Write the evalpoly function described above using <u>both</u> of these two different styles:
   (i) Use recursion, but do not call any predefined higher-order functions.
   (ii) Use predefined higher-order functions, but do not write any recursion.

   evalpoly :: ____

6. **CS 403:** (evaluate e) evaluates expression e using the data type Expr shown below. Example: evaluate (Mul (Add (Val 3) (Val 4)) (Sub (Val 8) (Neg (Val 2)))) returns (3+4)*(8-(-2)) = 70.

   data Expr = Val Integer | Neg Expr | Add Expr Expr | Sub Expr Expr | Mul Expr Expr

   CS 503: (evaluate e list) evaluates expression e using the data type Expr shown below. The list contains pairs of identifiers and their integer values. Example:
   evaluate (Mul (Add (Sym "w") (Sym "x")) (Sub (Sym "y") (Neg (Sym "z"))))
   [("w",3),("x",4),("y",8),("z",2)] returns (3+4)*(8-(-2)) = 70.

   data Expr = Val Integer | Sym String | Neg Expr | Add Expr Expr | Sub Expr Expr | Mul Expr Expr

   evaluate :: ____