CMPSC 461 **Programming Language Concepts** *Spring 2014*
Penn State University **Assignment #3** *Due: 6 February*

The purpose of this assignment is to gain more experience working with higher-order functions and lists in Scheme. You may not use any of Scheme's imperative features (assignment/loops). You must submit your solution as a source code attachment to the Angel Drop Box.

1. (10 pts) Recall that the function `map` takes a function and list of arguments and returns the list obtained by applying the function to each of the arguments. The function `funmap`, instead takes a list of functions and one argument and returns the list obtained by applying each function in the list to that one argument. E.g., `(funmap (list sqr, sqrt) 4)` evaluates to the list `(16 2)`. Define `funmap` directly using `map`. Your definition of `funmap` should not itself be recursive (but will use the recursive function `map`).

2. (10 pts) Define a function `funcompose` that takes a list of (unary) functions and returns their composition. E.g., `(funcompose (list sqr floor sqrt))` returns a function equivalent to `(lambda (x) (sqr (floor (sqrt x))))`.

3. (10 pts) Using only `lambda` expressions, function application, any of the logical operators (`and`, `or`, `not`) and the constants `#t` and `#f`, define the following two functions that take a Church numeral as an argument:

   (a) The function `zero-Church?` that returns `#t` when given the Church numeral 0 and returns `#f` when given any other Church numeral.

   (b) The function `even-Church?` that returns `#t` when given an even Church numeral and returns `#f` when given any other Church numeral.

4. (10 pts)

   (a) Define `Factorials` to be the infinite stream of factorial numbers, i.e., 1,1,2,6,24,120,...

   (b) Define the function `multiples-of` that takes a positive integer `k` and returns the stream of multiples of `k`. E.g., `(multiples-of 3)` returns the stream consisting of 3,6,9,12,...

5. (10 pts) Define `map-stream` to be the function that takes a function `f` and a stream `s` and returns the stream obtained by applying `f` to each element of the stream `s`.