September 15, 2016 Assignment-2 Due: September 27, 2016

COP - 4555

**[Reminder: The first mid-term test is on September 27, 2016]**

This assignment is all about writing Recursive Functions. Follow the methodology that we used in class to develop the solutions. Using the steps of the checklist will be very helpful! Use “pattern matching” to solve problems 1, 2, and 3.

1. Write an uncurried F# function cartesian (xs, ys) that takes as input two lists xs and ys and returns a list of pairs that represents the Cartesian product of xs and ys. (The pairs in the Cartesian product may appear in any order.) For example,

> cartesian (["a"; "b"; "c"], [1; 2]);;

val it : (string \* int) list =

[("a", 1); ("b", 1); ("c", 1); ("a", 2); ("b", 2); ("c", 2)]

1. An F# list can be thought of as representing a set, where the order of the elements in the list is irrelevant. Write an F# function powerset such that powerset of set S returns the set of all subsets of S. For example,

> powerset [1;2;3];;

val it : int list list

= [[]; [3]; [2]; [2; 3]; [1]; [1; 3]; [1; 2]; [1; 2; 3]]

Note that you can order the elements of the powerset however you wish.

1. The *transpose* of a matrix M is the matrix obtained by reflecting M about its diagonal. For example, the transpose of

/ 1 2 3 \

\ 4 5 6 /

is

/ 1 4 \

| 2 5 |

\ 3 6 /

An *m*-by-*n* matrix can be represented in F# as a list of *m* *rows*, each of which is a list of length *n*. For example, the first matrix above is represented as the list

[[1;2;3];[4;5;6]]

Write an efficient F# function to compute the transpose of an *m*-by-*n* matrix:

> transpose [[1;2;3];[4;5;6]];;

val it : int list list = [[1; 4]; [2; 5]; [3; 6]]

Assume that all the rows in the matrix have the same length.

Hints: transpose [] should give an error message --- Use “failwith”

transpose [[]] is [] // Input argument is a list with one element, the empty list.

1. In this problem and the next, I ask you to *analyze code*, as discussed in the last section of the Checklist. Suppose we wish to define an F# function to sort a list of integers into non-decreasing order. For example, we would want the following behavior:

> sort [3;1;4;1;5;9;2;6;5];;

val it : int list = [1; 1; 2; 3; 4; 5; 5; 6; 9]

We might try the following definition:

let rec sort = function

| [] -> []

| [x] -> [x]

| x1::x2::xs -> if x1 <= x2 then x1 :: sort (x2::xs)

else x2 :: sort (x1::xs)

Analyze the correctness of this definition with respect to the [*Checklist for Programming with Recursion*](https://moodle.cis.fiu.edu/v2.1/mod/page/view.php?id=8949), being sure to address *all three Steps*.

1. Here is an attempt to write mergesort in F#:

let rec merge = function // Merges two sorted lists

| ([], ys) -> ys

| (xs, []) -> xs

| (x::xs, y::ys) -> if x < y then x :: merge (xs, y::ys)

else y :: merge (x::xs, ys)

let rec split = function

| [] -> ([], [])

| [a] -> ([a], [])

| a::b::cs -> let (M,N) = split cs(a::M, b::N)

let rec mergesort = function

| [] -> []

| L -> let (M, N) = split L

merge (mergesort M, mergesort N)

* 1. Analyze mergesort with respect to the [*Checklist for Programming with Recursion*](https://moodle.cis.fiu.edu/v2.1/mod/page/view.php?id=8949), again addressing all three Steps.
  2. Enter this program into F# and see what type F# infers for mergesort. Why is this type a clue that something is wrong with mergesort?
  3. Based on your analysis, correct the bug in mergesort.

1. Recall that an F# function that takes two arguments can be coded in either uncurried form (in which case it takes a pair as its input) or curried form (in which case it takes the first argument and returns a function that takes the second argument). In fact it is easy to convert from one form to the other in F#. To this end, define an F# function curry f that converts an uncurried function to a curried function, and an F# function uncurry f that does the opposite conversion. For example,

> (+);;

val it : (int -> int -> int) = <fun:it@13-7>

> let plus = uncurry (+);;

val plus : (int \* int -> int)

> plus (2,3);;

val it : int = 5

> let cplus = curry plus;;

val cplus : (int -> int -> int)

> let plus3 = cplus 3;;

val plus3 : (int -> int)

> plus3 10;;

val it : int = 13

What are the types of curry and uncurry?

Submit:

1. The hard copy of your well-documented F# script file (containing solutions to all problems) at the beginning of the class on the due date.
2. The soft copy of the same F# program on SCIS Moodle page. Include illustrative test cases (not just the ones I have specified above) for all problems as part of your test cases. Your output should be self-explanatory – use “printfn” with what you are printing and the result. I should be able to execute your code WITHOUT ANY MODIFICATIONS.

Good Luck!