September 6, 2016 Assignment – 1 Due: September 15, 2016

COP – 4555

Given below are a few problems. Solve them using pure F# functional programming code (no mutable variables, no loops, etc.). Develop your F# script file (as explained in Class Notes) that contains solutions of all problems, using Microsoft Visual Studio 2015. Do not use built-in F# functions other than what we have discussed in class.

Document your program well so that the reader can read and understand it quickly. You are free to define auxiliary functions to assist you in solving some of the given problems. Every function defined should be very well documented.

Make sure that you clearly understand the types that are inferred by F# for all your functions. Hopefully, this will help you understand F# Types better.

You will be graded on the correctness of your answers as well as its quality and completeness. I expect you to write your code using Functional Programming principles and not translating code from an imperative programming sense. **Completeness of submission means that all test cases (enough to cover all nuances of the problem – you should be an expert on testing your programs) are included.** And oh, I should be able to execute your code and test cases just as you submit without doing any extra work. These instructions and policies apply to all assignments in this course.

1. A fraction like 2/3 can be represented in F# as a pair of type int \* int. Define infix operators .+ and .\* to do addition and multiplication of fractions:

> (1,2) .+ (1,3);; // (1/2) + (2/3) = (5/6)

val it : int \* int = (5, 6)

> (1,2) .+ (2,3) .\* (3,7);; // (1/2) + (2/3) \* (3/7) = (11/14)

val it : int \* int = (11, 14)

> (1,8) .+ (3,8);; // (1/8) + (3/8) = (1/2)

Val: it:int \* int = (1, 2)

Note that the F# syntax for defining such an infix operator looks like this:

let (.+) (a,b) (c,d) = ...

Also note that .+ and .\* get the same precedence as + and \*, respectively, which is why the second example above gives the result it does.

Finally, note that your functions should always return fractions in *lowest terms*. To implement this, you will need an auxiliary function to calculate the gcd (*greatest common divisor*) of the numerator and the denominator; this can be done very efficiently using Euclid's algorithm:

gcd (x, y) = x; if y=0

= gcd (y, x mod y); if y>0

1. Write an F# function revlists xs that takes a list of lists xs and reverses all the sub-lists:

> revlists [[0;1;1];[3;2];[];[5]];;

val it : int list list = [[1; 1; 0]; [2; 3]; []; [5]]

*Hint*: This is trivial using List.map.

1. Write an F# function interleave l1 l2 (defined below with examples) that interleaves two lists of arbitrary size:

> interleave [1..3] [4..6];;

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

> interleave [1..10] [3..5];;

val it : int list = [1; 3; 2; 4; 3; 5; 4; 5; 6; 7; 8; 9; 10]

> interleave [3..5] [1..10];;

val it : int list = [3; 1; 4; 2; 5; 3; 4; 5; 6; 7; 8; 9; 10]

> interleave ["cat"; "dog"; "elephant"; "lion"] ["tiger"; "rat"];;

val it : string list = ["cat"; "tiger"; "dog"; "rat"; "elephant"; "lion"]

1. Write an F# function cut xs that cuts a list into two equal parts:

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

val it : int list \* int list = ([1; 2; 3], [4; 5; 6])

Assume that the list has even length.

To implement cut, first define an auxiliary function gencut(n, xs) that cuts xs into two pieces, where n gives the size of the first piece:

> gencut(2, [1;3;4;2;7;0;9]);;

val it : int list \* int list = ([1; 3], [4; 2; 7; 0; 9])

Paradoxically, although gencut is more general than cut, it is easier to write! (This is an example of Polya's *Inventor's Paradox*: "The more ambitious plan may have more chances of success.")

*Another Hint*: To write gencut efficiently, it is quite convenient to use F#'s *local* let expression (as in the cos\_squared example in the Notes).

1. Write an F# function shuffle xs that takes an even-length list, cuts it into two equal-sized pieces, and then interleaves the pieces (On a deck of cards, this is called a *perfect out-shuffle*).

> shuffle [1;2;3;4;5;6;7;8];;

val it : int list = [1; 5; 2; 6; 3; 7; 4; 8]

1. Write an F# function countshuffles n that counts how many calls to shuffle on a deck of n distinct "cards" it takes to put the deck back into its original order:

> countshuffles 4;;

val it : int = 2

(To see that this result is correct, note that shuffle [1;2;3;4] = [1;3;2;4], and shuffle [1;3;2;4] = [1;2;3;4].) What is countshuffles 52?

*Hint*: Define an auxiliary function countaux(deck, target) that takes two lists and returns the number of shuffles it takes to make deck equal to target.

Submit:

1. The hard copy of your well-documented F# script file in class. Also, submit the hard copy of the output of this program for a variety of test cases illustrating all nuances of your solution. Use “printf” statement judiciously to make your output self-explanatory.
2. The soft copy of the same F# program on SCIS Moodle page. Include illustrative test cases for all problems as part of your test cases. For example, for problem-1, a few illustrative test cases might be (1,2) (.+) (3,4); (1,2)\*(16,25); (1,2) (.+) (2,3) (.\*) (3,4); and so on.

Good Luck!