Homework #03

Complete By: Saturday, October 14th @ 11:59pm
Policy: Individual work only, late work *not* accepted
Assignment: various F# functions
Submission: submit "hw03.fs" electronically via Gradescope

Homework Exercises

The assignment is to write five functions in F#. A replit.com project named "**Homework 03**" is available for writing your functions. If you prefer to work in a different programming environment, copy/download the provided file templates from replit.com.

The code containing your five functions must be in a file named: hw03.fs

The code is to be called from the F# code in a file named: "main.fs". The file main.fs will act as the testing code that will contain all of the code needed to verify correct execution of the functions in the file "hw03.fs". The initial code in main.fs only has one test case included. You are expected to create additional test case code in main.fs to verify the functionality of your functions in hw03.fs. Note that the version of main.fs that is part of the autograder code is fairly long and contains quite a number of test cases. The test cases for the autograder will not be released to the class. So, students are encouraged to be thorough when creating your own test cases.

The five functions that are to be written for this assignment are described below. You are also to make sure you follow the directions specified in the comments for each function regarding:

- parameters
- return values
- required implementation details
- helper functions

Some of the functions may expect you to create helper functions to make sure they run properly.

Exercise #01: subset S L

```
//
// subset S L
//
// Returns true if S is a subset of L, false if not.
//
// Examples: subset [3; 1; 2] [1; 4; 2; 0; 3] => yes
```

```
//         subset [2; 1; 3] [1; 4; 0; 2; 8] => no
//         subset [] []                     => yes
//         subset [] [1; 2; 3]              => yes
//         subset [1..1000] [1..1000]       => yes
//
// NOTE: you can solve using tail-recursion, or using a
// higher-order approach. Whatever you prefer. Beware
// that List.length is an O(N) operation, since it
// traverses the entire list and counts the elements.
//
// HINT: there are a variety of solutions. One idea
// is write a helper function "contains e L" that
// returns true if e is an element of L, and false
// if not. Then build on that to define subset. This
// leads to an O(N^2) solution, which is fine.
   //
  let rec subset S L =
      ()
```

Exercise #02: **delete_tr e L** in a tail-recursive manner

```
//
// delete_tr e L
//
// Deletes all occurrences of e from the list L,
// returning the new list. This version is written
// recursively, using a helper function that is
// tail-recursive.
//
// Examples: delete_tr 3  [3; 1; 2]   => [1; 2]
//         delete_tr 99 [99; 0; 99] => [0]
//         delete_tr -2 []          => []
//         delete_tr "cat" ["dog"]  => ["dog"]
//         delete_tr 99999 [1..99999] => [1..99998]
//
// NOTE: write a tail-recursive version using a helper
// function; do not change the API of the original
// delete function. You'll also need to build the new
// list efficiently; you can use List.rev if need be.
//
  let delete_tr e L =
      ()
```

Exercise #03: **delete_ho e L** using a higher-order approach

```
//
// delete_ho e L
//
// Deletes all occurrences of e from the list L,
// returning the new list. This version uses a
```

```
// higher-order approach.
//
// Examples: delete_ho 3  [3; 1; 2]   => [1; 2]
//         delete_ho 99 [99; 0; 99] => [0]
//         delete_ho -2 []         => []
//         delete_ho "cat" ["dog"]  => ["dog"]
//         delete_ho 99999 [1..99999] => [1..99998]
//
  let delete_ho e L =
      ()
```

Exercise #04: examAverages LT

```
//
// examAverages LT
//
// Given a list of tuples of the form ("netid", [score;score;score;...]),
// computes each netid's average score as a real number and returns a
// list of tuples of the form ("netid", average).
//
// Example:
//   examAverages [("sdeitz2",[100;90;91]); ("psankar",[100;100;100;100;98])]
//     => [("sdeitz2",93.66666667); ("psankar",99.6)]
//
// NOTE: F# offers a function List.average L that computes the average of
// a list of real numbers.  However, the list of scores in the tuple are
// integers, so in order to use List.average, you would first need to convert
// the integers to reals --- List.map float L would work nicely here.
//
// You can solve using recursion, or higher-order, or both; tail recursion
// is not necessary.
   //
  let rec examAverages LT =
      []
```

Exercise #05: pairwise L1 L2

```
//
// pairwise L1 L2
//
// Given 2 lists L1 and L2, both the same length, merges the corresponding
// elements into pairs, returning a list of pairs.
//
// Example:
//   pairwise [1;3;5;7] [10;20;30;40]
//     => [(1,10); (3,20); (5,30); (7,40)]
//
// You must solve using recursion; tail recursion is not necessary.
// Note: there is a F# library function named zip that does this operation.
// You may not use that function in your solution.
```

```
//
  let rec pairwise L1 L2 =
     []
```

Requirements

No variables (i.e. no use of the keyword **mutable**). **No loops**. Instead, **use recursion or higher-order programming** as noted in the header comments above. Failure to use the proper approach will result in deduction of points after the assignment deadline.

Proper tail recursion is quite specific. Double check that any solutions that you think uses tail recursion really does use that approach. In previous semester, many students submitted assignments that were in fact NOT proper tail recursion. Tail recursion is required for Exercise #02. If you are not using a higher ordered approach for Exercise #01, tail recursion is required for Exercise #01 as well. (For Exercise #01, if you write a "contains" helper function, this function does not need to be tail recursive – but it is good practice.)

Summary of implementation requirements:

Exercise #01: subset – must use either tail recursion OR a higher-order approach
Exercise #02: delete_tr – must use tail recursion
Exercise #03: delete_ho – must use a higher-order approach
Exercise #04: examAverages – must use either recursion (any recursion is fine) or a higher-order approach
Exercise #05: pairwise – must use recursion (any recursion is fine)

Grading and Electronic Submission

A day or two before the assignment is due, login to Gradescope.com and look for the assignment "Homework 03". Submit your F# program file "hw03.fs" --- you have unlimited submissions. Keep in mind we grade your LAST SUBMISSION unless you select an earlier submission for grading. You need to complete all the functions to get credit for this assignment. Add your name to the header comment and comment any functions you write.

Academic Conduct Policy

Late work is not accepted for this assignment. All work is to be done individually — group work is not allowed. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml

In particular, note that you are guilty of academic dishonesty if you **extend or receive any kind of unauthorized assistance**. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums (e.g. you cannot download answers from Chegg). Other examples of academic dishonesty include emailing your program to another student, sharing your screen so that another student

may copy your work, copying-pasting code from the internet, working together in a group, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml .