CSC 402/502 Homework 3 Instructor: Jeff Ward
Covers: F#: Lists (Sections 4.1-4.5) 30 pts

For this assignment create a file called Program.fs. All of your code for this assignment should be placed in this .fs file.

If you are using JetBrains Rider as your IDE then you can create the file and runnable project as you did for the earlier assignments: Hit File->New... . Select "Console Application" under .NET Core. Give the solution a name (e.g. "HW3" without the quotes). The dialog box will fill in the same name (e.g. "HW3") as the name of the project. Click "Create". Rider will create a file named Program.fs under the HW3 project.

Each of the following problems ask you to write a recursive function that works on lists. The type of your function should match the type that is shown.

Try to make your solutions concise. The bodies of the instructor's sample solutions are 3-5 lines of relatively simple code for each problem. The problems are easiest to solve using pattern matching on the lists as demonstrated in Chapter 4. Please read Sections 4.1-4.5 before attempting the assignment. (Section 4.6 is cool as well, but not essential to getting through these problems.) Here are a couple of pattern matching examples from Sections 4.2-4.3 to refresh your memory:

```
let rec suml = function
    | []    -> 0                // Pattern matching on one list
    | x::xs -> x + suml xs;;

let rec mix xlst ylst =
    match (xlst,ylst) with      // Pattern matching on two lists
    | (x::xs,y::ys) -> x::y::(mix xs ys)
    | ([],[])       -> []
    | _             -> failwith "mix: parameter error";;
```

In the second example above (`mix`) we see that the `match` keyword enables us to convert pattern matching on two lists to pattern matching on a tuple (specifically, a pair), which was covered in Chapter 3.

Problem 1 (6 points) Write a recursive function `count` of type `'a -> 'a list -> int` so that `count x xs` is the number of occurences of x in xs. Use the simple equality comparison operator (=) to compare elements.

```
val count : e:'a -> _arg1:'a list -> int when 'a : equality
```

```
> count 18 [20;17;18;18;3;4;18;14];;
val it : int = 3
```

```
> count "hi" ["hello";"hi";"greetings";"welcome";"hi";"hola";"ni hao"];;
val it : int = 2
```

```
> count 9 [];;
val it : int = 0
```

Problem 2 (6 points) Write a recursive function `keepSatisfiers` that takes a function `f` and a list and returns the sublist consisting of all of the elements that satisfy `f`.

```
val keepSatisfiers : f:('a -> bool) -> _arg1:'a list -> 'a list
```

```
> keepSatisfiers (fun x -> x % 2 = 0) [2;3;4;5;7;10;11];;
val it : int list = [2; 4; 10]
```

```
> keepSatisfiers (fun x -> (abs x) > 100) [-103..102];;
val it : int list = [-103; -102; -101; 101; 102]
```

Problem 3 (6 points) Write a recursive function `pairOfListsToListOfPairs`. It takes a pair of lists and returns a list of pairs. The i-th element in the returned list consists of the i-th element in the first parameter paired with the i-th element in the second parameter. The length of the returned list is the same as the length of the shorter of the two parameters.

Tip -- Since the parameter is a pair, this function can be written as pattern matching on a pair:

```
let rec pairOfListsToListOfPairs = function
    | (x::xs,y::ys) -> ...  (fill in the dots)
    | ([],ys) -> ...
    | (xs,[]) -> ...
```

```
val pairOfListsToListOfPairs : 'a list * 'b list -> ('a * 'b) list
```

```
> pairOfListsToListOfPairs([2;3;5;7;11],['a';'b';'c';'d';'e']);;
val it : (int * char) list =
  [(2, 'a'); (3, 'b'); (5, 'c'); (7, 'd'); (11, 'e')]
```

```
> pairOfListsToListOfPairs(['J'..'P'],[-2..2]);;
val it : (char * int) list =
  [('J', -2); ('K', -1); ('L', 0); ('M', 1); ('N', 2)]
```

Problem 4 (6 points) Write a recursive function `maxOfList` that returns the largest element of a list. Your function should `failwith "max of empty list undefined"` if called on an empty list. You may use F#'s `max` function, which returns the maximum of two values.

```
val maxOfList : _arg1:'a list -> 'a when 'a : comparison
```

```
> maxOfList [1;4;1000;-2];;
val it : int = 1000
```

```
> maxOfList [42];;
val it : int = 42
```

```
> maxOfList ([]:int list);;
System.Exception: max of empty list undefined
   at FSI_0010.maxOfList[a](FSharpList`1 _arg1)
   at <StartupCode$FSI_0014>.$FSI_0014.main@()
Stopped due to error
```

Problem 5 (6 points) A list is considered *weakly increasing* if each element after the first element is greater than or equal to its predecessor. Write a `merge` function that returns the result of merging two lists. If the two lists are both weakly increasing then the result should be weakly increasing as well. (This is essentially the `merge` method from the Merge Sort section in the CSC 364 textbook. But we can write it much more concisely in F# than in Java.)

Hint: Consider the (higher-order) `mix` function defined in Section 4.3 and listed on the first page of this document. That function also has type `'a list -> 'a list -> 'a list`. Note that, unlike the `mix` function, the `merge` function will not fail if the lists have different lengths.

```
val merge : xlst:'a list -> ylst:'a list -> 'a list when 'a : comparison
```

```
> merge [2;3;5;7;11;13;17] [2;2;3;4;5;5;18;18;20];;
val it : int list = [2; 2; 2; 3; 3; 4; 5; 5; 5; 7; 11; 13; 17; 18; 18; 20]
```

```
> merge ["Abe";"Bill";"Cody"] ["Bertha";"Billie";"Cora";"Doris"];;
val it : string list =
  ["Abe"; "Bertha"; "Bill"; "Billie"; "Cody"; "Cora"; "Doris"]
```

What to turn in:

Submit on Canvas your Program.fs file. You are not required to include any comments in the file, but feel free to include them if you like.