CPSC 3400 Languages and Computation
Winter 2017

Homework 3
75 points
Due: Friday, February 24 Midnight

This assignment consists of three F# exercises. The code for all exercises are to be included in this single file `hw3.fs`.

For each of the exercises, you are only required to write the provided functions. There is no additional global functionality. Your functions should produce no output. To test your functions, either:

- Test the functions using F# Interactive in Visual Studio OR
- Add your own tests to the script (but be sure to remove them before submission)

1. (15 points) Write a function `sumEven` that takes a list of integers and returns the sum of the even integers only. If there are no even integers, return 0.

Examples in F# Interactive:

```
> sumEven [1; 2; 1; 2; 1; 2];;
val it : int = 6
> sumEven [1..10];;
val it : int = 30
> sumEven [1; 3; 5; 7; 9;];;
val it : int = 0
```

2. (30 points) Assigning people to tables at a wedding can be a difficult task. There are always people who simply can't sit at the same table. Write a function `isValidTable` that determines if the group of people assigned to a table is valid (no conflicts) or not. The function has two parameters:

- A list of constraints. Each constraint is represented by a tuple that contains two names and refers to two people who cannot sit at the same table.
- A list of names that represents people assigned at a table.

The function returns true if the table is free of conflicts and false if there are two (or more) people that cannot sit together at the table. If either parameter is an empty list, return true.

Examples in F# Interactive:

```
> let x = [("Eric", "Mark"); ("Anna", "Maya"); ("Beth", "Hope")];;

val x : (string * string) list =
  [("Eric", "Mark"); ("Anna", "Maya"); ("Beth", "Hope")]

> isValidTable x ["Eric"; "Anna"; "Beth"];;
val it : bool = true
> isValidTable x ["Greg"; "Eric"; "John"; "Anna"; "Beth"];;
val it : bool = true
> isValidTable x ["Hope"; "Eric"; "Anna"; "Beth"];;
val it : bool = false
> isValidTable x ["Mark"; "Beth"; "Eric"; "Anna"];;
val it : bool = false
```

3. (30 points) Write a function `getClosestPair` that takes a list of *x*, *y* coordinates and returns the coordinate that is the closest distance to the origin. An *x*, *y* coordinate is a tuple where both *x* and *y* are floating point values. If this list is empty, return `(infinity, infinity)`. Note `infinity` is a constant value in F# that represents positive infinity.

Examples in F# Interactive:

```
> closestPair [(1.0, 1.0); (2.0, 2.0); (0.5, 0.5); (4.0, 4.0)];;
val it : float * float = (0.5, 0.5)
> closestPair [(1.0, 30.0); (2.0, 20.0); (3.0, 3.0);];;
val it : float * float = (3.0, 3.0)
> closestPair [(-1.0, 0.0); (1.0, 0.0); (0.0, 0.0);];;
val it : float * float = (0.0, 0.0)
> closestPair [];;
val it : float * float = (infinity, infinity)
```

Hint: A tail-recursive approach may be helpful, but not necessary, in solving this exercise.

Style and Implementation Rules

- You may define and use additional functions if your feel it is appropriate.
- Keep each line at 80 characters.
- There are no specific indenting and commenting rules but keep the code easy-to-read.
- Without prior approval from me, you may only use the subset of F# described in class.
    - In particular, you may NOT use F# system functions or methods (such as `min`).

Submitting your Assignment

Programs will be submitted electronically via Canvas (not Linux). Submit a single file called `hw3.fs` that contains the code for all three exercises. Be sure to remove any testing code and make sure your functions have the same name as listed in this document.

You may submit multiple submissions but only the last submission will be graded. ***Late submissions are not accepted and result in a zero.***