CSCI 330
Fall 2017
Lab 1: Beginning OCaml
Due Monday, September 17th at 11:59pm
(75 pts, 60 functionality and 15 commenting)

The overall objective of this assignment is for you to gain some hands-on experience with OCaml. All the problems require relatively little code ranging from 2 to 15 lines. If any function requires more than that, you can be sure that you need to rethink your solution. Download the handout from autolab and add your function definitions and comments to the misc.ml file (it contains skeleton OCaml functions) Your task is to replace the 'failwith "to be written"' text in those files with the appropriate OCaml code for each of those expressions. Do NOT change any of the function headers and be sure to write each function to the exact specs of the lab, including name. Your code will be auto tested, and any deviations will cause your code to fail testing.

To compile your code, while in the utop or ocaml interpreter (run from the same directory as misc.ml), type:
#use "misc.ml";

Note: All the solutions must be done using the purely functional fragment of OCaml, using constructs covered in class, and most require the use of recursion. Solutions using imperative features such as references, arrays, or while loops will receive no credit. Do not use any additional libraries, just use standard OCaml as installed in the Linux and Windows labs.

It is a good idea to start this assignment early; ML programming, while quite simple (when you know how), often seems somewhat foreign at first, particularly when it comes to recursion and list manipulation.

1. (10 pts) sumList
   Write an OCaml function sumList : int list -> int that takes an integer list l and returns the sum of the elements of l. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

   # sumList [1;2;3;4];;
   - : int = 10
   # sumList [1;-2;3;5];;
   - : int = 7
   # sumList [1;3;5;7;9;11];;
   - : int = 36

2. (10 pts) digitsOfInt
   Write an OCaml function digitsOfInt : int -> int list that takes an integer n as an argument and if the integer is positive (i.e. I don't care what you return for the argument 0 or any
negative number), returns the list of digits of \( n \) in the order in which they appear in \( n \). Once you have implemented the function, you should get the following behavior at the OCaml prompt:

```ocaml
# digitsOfInt 3124;;
- : int list = [3;1;2;4]
# digitsOfInt 352663;;
- : int list = [3;5;2;6;6;3]
```

3. (20 pts, 10 each) additivePersistence and digitalRoot
Consider the process of taking a number, adding its digits, then adding the digits of the number derived from it, etc., until the remaining number has only one digit. The number of additions required to obtain a single digit from a number \( n \) is called the additive persistence of \( n \), and the digit obtained is called the digital root of \( n \). For example, the sequence obtained from the starting number 9876 is (9876, 30, 3), so 9876 has an additive persistence of 2 and a digital root of 3.

Write two OCaml functions `additivePersistence : int -> int` and `digitalRoot : int -> int` that take positive integer arguments \( n \) and return respectively the additive persistence and the digital root of \( n \). Once you have implemented the functions, you should get the following behavior at the OCaml prompt:

```ocaml
# digitalRoot 9876;;
- : int = 3
# additivePersistence 9876;;
- : int = 2
```

4. (15 pts) **Without using any built-in OCaml functions**, write an OCaml function `listReverse : 'a list -> 'a list` that takes a list \( l \) as an argument and returns a list of the elements of \( l \) in the reversed order. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

```ocaml
# listReverse [1;2;3;4];;
- : int list = [4;3;2;1]
# listReverse ["a";"b";"c";"d"];;
- : string list = ["d";"c";"b";"a"]
```

5. (5 pts) A palindrome is a word that reads the same from left-to-right and right-to-left. Write an OCaml function `palindrome : string -> bool` that takes a string \( w \) and returns true if the string is a palindrome and false otherwise. You may want to use the OCaml function `explode` (defined in misc.ml) as well as your `listReverse` function. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

```ocaml
# palindrome "malayalam";;
- : bool = true
# palindrome "myxomatosis";;
- : bool = false
```

Submission: Login to autolab and select the Lab 1 assignment. Upload and submit your revised version of misc.ml.

Attribution: This assignment was adapted from Sorin Lerner’s CSE 130 course at UCSD.