CS 303 Principles of Programming Languages

Assignment 1 [Deadline is 13th of October, 5pm]

With this assignment you will get some hands- on experience with OCaml. All the problems require relatively little code ranging from 2-15 lines.

Download pa1.zip and unzip it.

misc.ml contains several skeleton functions with missing bodies that you will fill in.

test.ml contains some sample tests for you to start with, you can extend this file to test your functions in more detail but this is optional.

The file misc.ml contains expression of the form:

```
 failwith "...".
```

Your task is to replace the text in those files with the appropriate OCaml code for each of these expressions.

Note: All the solutions can 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, while loops or library functions will receive no credit. 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.

Most of the points, will be awarded automatically, by evaluating your functions against a given test suite. test.ml contains a very small suite of tests which gives you a flavor of of these tests.
At any stage, by typing

```
% ocaml test.ml > log
```

at the shell, you will get a report on how your code stacks up against the simple tests.
Note: these are only *sample* tests, and we will use a larger test suite when grading your submission.

To submit your homework, send your misc.ml file to the email address email.cs303@gmail.com

Details of the questions

Problem #1: Digital Roots and Additive Persistence

(A)

Write an OCaml function

```
val sumList : int list -> int
```

tsuch that `sumList xs` returns the sum of the integer elements of xs. 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
```

(B) Write an OCaml function

```
val digitsOfInt : int -> int list
```

such that `digitsOfInt n` returns [] if n is not positive, and 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:

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

(C) 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:

```
val additivePersistence : int -> int
val 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:

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

Problem #2: Palindromes

(A)

Without using any built-in OCaml functions, write an OCaml function:

```
val listReverse : 'a list -> 'a list
```

such that `listReverse xs` returns the list of elements of xs in the reversed order (in which the appear in xs.) Once you have implemented the function, you should get the following behavior at the OCaml prompt:

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

(B)

A palindrome is a word that reads the same from left-to-right and right-to-left. Write an OCaml function

```
val palindrome : string -> bool
```

such that palindrome w returns true if the string is a palindrome and false otherwise. You may use the given helper function explode. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

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

References:

CSE 130 course of UCSD