A. "ludicrous" - the input to this function is a single integer.

The output is a boolean (either true #t or false #f), indicating

whether or not the input number is "ludicrous".

A number is deemed to be "ludicrous" if every one of

its digits is even, or if every one of its digits is odd.

For example, (ludicrous 8126) would return #f, because one digit is odd

while the others are even. As another example, (ludicrous 315) would

return #t, because every digit is odd. Similarly, (ludicrous 860)

would return #t (true), because every digit is even.

You'll probably need to use the modulo function, and you'll probably also

need integer division, which can be found in Racket's help facility.

B. "cappedSum" - takes as input an integer C, and a list L of integers.

The input integer C is called the "cap". The output is the integer sum of

each integer in L, "capping" each number at a maximum value of C.

For example, (cappedSum 7 '(6 4 9 2 8)) should return 26,

because 6 + 4 + 7 + 2 + 7 = 26 (note that the 9 and 8 are both greater than 7,

and so they have been replaced with 7).

C. "reverseShuffle" - takes two lists, and returns a single list that is

the "reverseShuffle" merging of the two input lists. A "reverseShuffle"

is a merging of two lists L and M, that is built in the following order:

1 - the first element of L

2 - the last element of M

3 - the second element of L

4 - the next-to-last element of M

etc.

Thus the output list is built by alternating items from L and M,

in the forward direction from L, and the backward direction from M.

For example, (reverseShuffle '(3 8 2 6) '(7 1 5 9))

should return: (3 9 8 5 2 1 6 7)

You can assume that the two incoming lists are both the same length.

You may find it useful to break the problem down and create some

helper functions.

D. "biggestListSize" - takes a list M as input, and outputs an integer.

M is a list of lists, each of potentially different sizes. The output

integer is the size of the largest list in M. For example:

(biggestListSize '(() (3 4 7) (2 1) (5 5 2) (8))) should return 3.

E. "functionPairs" - takes as input a function F and a list L. The output

is a list consisting of each ith element in L for which F(L[i])=F(L[i+1]).

For example, (functionPairs isEven '(3 7 5 2 8)) should return (3 7 2).

As another example, (functionPairs square '(-2 2 4 5 -5)) returns (-2 5).

This second example assumes that (square x) is defined as (\* x x).

F. "nestedCappedSum" - the same as problem "B", above, except that the input

list can be nested. For example:

(nestedCappedSum 7 '(6 4 ((9 2) 8) 5)) should return 31,

because 6 + 4 + 7 + 2 + 7 + 5 = 31.

You will find it useful to use the "list?" function,

which works as follows: (list? L) returns true if L is a list.

G. A function "makeExploder", that takes as input a list L of length 3.

It then builds and returns an "exploder" function based on L.

The "exploder" function that is produced (by your function makeExploder)

would have the property that it takes as input an integer N, and returns

an "exploded" version of N. "Exploding" a number means first adding X to it,

then multiplying it by Y, then raising that result to the power of Z.

The values of X, Y, and Z correspond to the three values in the original

list L that was sent to makeExploder.

For example, if makeExploder was called as follows:

(define P (makeExploder '(2 5 3)))

then the the produced function P would behave as follows:

(P 4) would return 27000

(P 5) would return 42875

Your task is to write makeExploder, not P.

You may find it useful to write one or more utility functions.

Of course, makeExploder should work for any reasonable input list,

not just the one shown above.