Name :
Number :
Signature :

CSE 3215 Homework 1
Due Date : 16.10.2019

Have you ever wondered how websites validate your credit card number when you shop online? They don't check a massive database of numbers, and they don't use magic. In fact, most credit providers rely on a checksum formula for distinguishing valid numbers from random collections of digits (or typing mistakes).

In this section, you will implement the validation algorithm for credit cards. It follows these steps:

- Double the value of every second digit beginning from the right.That is, the last digit is unchanged; the second-to-last digit is doubled; the third-to-last digit is unchanged; and so on. For example,[1,3,8,6] becomes [2,3,16,6].
- Add the digits of the doubled values and the undoubled digits from the original number. For example, [2,3,16,6] becomes 2+3+1+6+6 = 18.
- Calculate the remainder when the sum is divided by 10. For the above example, the remainder would be 8.If the result equals 0, then the number is valid.

1) We need to first find the digits of a number. Define the functions

```
toDigits    :: Integer -> [Integer]
toDigitsRev :: Integer -> [Integer]
```

`toDigits` should convert positive `Integers` to a list of digits. (For `0` or negative inputs, `toDigits` should return the empty list.) `toDigitsRev` should do the same, but with the digits reversed.

Example: `toDigits 1234 == [1,2,3,4]`
Example: `toDigitsRev 1234 == [4,3,2,1]`
Example: `toDigits 0 == []`
Example: `toDigits (-17) == []`

2) Once we have the digits in the proper order, we need to double every other one. Define a function

```
doubleEveryOther :: [Integer] -> [Integer]
```

Remember that `doubleEveryOther` should double every other number beginning from the right, that is, the second-to-last, fourth-to-last, . . . numbers are doubled.

Example: `doubleEveryOther [8,7,6,5] == [16,7,12,5]`
Example: `doubleEveryOther [1,2,3] == [1,4,3]`

3) The output of `doubleEveryOther` has a mix of one-digit and two-digit numbers. Define the function

```
sumDigits :: [Integer] -> Integer
```

to calculate the sum of all digits.

Example: sumDigits [16,7,12,5] = 1 + 6 + 7 + 1 + 2 + 5 = 22

4) Define the function

```
validate :: Integer -> Bool
```

that indicates whether an `Integer` could be a valid credit card number. This will use all functions defined in the previous exercises.

Example: validate 4012888888881881 = True
Example: validate 4012888888881882 = False