Scheme Lab 00000010

*Parts of this assignment are adapted (with permission) from Professor Brent Yorgey’s CS 194 assignment.*

Please submit all responses to your Dropbox in a file called lab02.rkt.

**The next several methods will be mimicking functionality that already exists within Scheme, but for these exercises those functions will be considered off-limits, you may use no other functions that process lists besides list, cons, and members of the car and cdr family.**

Make

**(get-element list n)**

which takes a list and returns the nth element. (Just like arrays, we will start counting from zero). This is mimicking the *list-ref* function.

Make

**(append-element list y)**

which takes a list x and an element, y, and returns list with y added to the end. This is mimicking the *append* function.

> (append-element '(1 3 5) 7)

'(1 3 5 7)

Make

**(append-list first second)**

which takes a list first and a list second, and returns a combined list. This is mimicking the *append* function.

> (append-list '(1 3 5) '(4 6 8))

'(1 3 5 4 6 8)

Make

**(backwards list)**

which takes a list, and returns a reversed list. This is mimicking the *reverse* function.

> (backwards '(1 3 5 8))

'(8 5 3 1)

Validating Credit Card Numbers

Adapted from the first practicum assigned in the University of Utrecht functional programming course taught by Doaitse Swierstra, 2008-2009.



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.

For the functions you are creating below, you may choose to create additional helper functions. Every function should be well-defined and contain appropriate contracts.

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

toDigits : integer -> list

toDigitsRev : integer -> list

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) 🡪 '()

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

doubleEveryOther : list -> list

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

Example: (doubleEveryOther '(8 7 6 5)) 🡪 '(16 7 12 5)

Example: (doubleEveryOther '(1 2 3)) 🡪 '(1 4 3)

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

sumDigits : list -> integer

to calculate the sum of all digits.

Example: (sumDigits '(16 7 12 5)) 🡪 1 + 6 + 7 + 1 + 2 + 5 = 22

**Exercise 4** Define the function

validate : integer -> boolean

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) 🡪 #t

Example: (validate 4012888888881882) 🡪 #f