Due date: **October 27th, 10:30am**

This lab is a sequence of exercises in Scheme. Each exercise requires you to write a program in a file called `exN.scm`, where `N` is the number of the exercise.

General advice and rules:

- Efficiency is not a major worry (within reason); we're exploring what you can do, not how to get the very best solution.
- Don't worry about detecting input errors; assume the input is correct. That implies that both the value and the type of the data supplied are correct.
- **Make sure all programs run on mzscheme on the ECF workstations.**
- Your are required to use appropriate functional programming style. **Do not use any loops. Do not use any functions with names ending in `'!'`.**
- Your functions should not print any output. All communication will be through parameters/arguments and return values.
- Make sure you exactly follow the specifications for input (parameters) and output (return value) for each question.
- Where a parameter is a list, your function must be able to handle the empty list, unless the question explicitly states that the list will always be non-empty.
- The specifications below are meant to be precise, unlike real-world requirements. However, there are bound to be places where clarity has not been achieved. If you find yourself confused, please ask.
- Some exercises look ahead to topics not covered at the time when the lab is handed out.
- These exercises are short, but you should still follow the usual style rules, which are just as relevant for all three programming languages used in this course as for any other languages you have used. For example, you should choose helpful names for variables and functions. Comments are valuable in all languages; even a very short function or code segment may be worth commenting, if you had to think hard to write it.

Academic offenses:

The standard rule is in effect: don't share your work with anyone else. Complete this lab by yourself.

Submission:

Submit your work using the command `submitcsc326f`. Set the first argument – the "assignment#" – to 2.

Exercise 1 [5 marks]

Write a function called **`rmlast`** that takes a single parameter, which is a non-empty list. The return value is the same list but with the last item removed.

Your submitted file must be named `ex1.scm`.

Example:

```
> (rmlast '(a b c))
(a b)
```

Exercise 2 [8 marks]

Write a function called **`addlists`** that takes two parameters `L1` and `L2`, both lists of numbers. The return value is a single list containing the pairwise sums of the numbers in `L1` and `L2`. If `L1` and `L2` are not the same length, the last element of the return value is the sum of the unmatched numbers in the longer of `L1` and `L2`.

Your submitted file must be named `ex2.scm`.

Examples:

```
> (addlists '(2 3 4) '(5 6 7))
(7 9 11)

> (addlists '(2 3 4) '(5 6 7 8 9 10))
(7 9 11 27)
```

Exercise 3 [8 marks]

Write a function called **`replace`** that takes three parameters: symbols `s1` and `s2` and a list `L`. The return value is a list that is the same as `L`, except with every occurrence of `s1` replaced by `s2`, no matter how deeply it is nested in `L`.

Your submitted file must be named `ex3.scm`.

Examples:

```
> (replace 'a 'b '(b d e a f g h))
(b d e b f g h)

> (replace 'c 'd '(d f (c) c (a (b c)) h))
(d f (d) d (a (b d)) h)
```

Exercise 4 [12 marks]

Write a function called **`secondsmallest`** that takes a single parameter, which is a list of numbers. The return value is the second-smallest number in the list, where "second-smallest" is defined the same way it was in Exercise 2 of Lab 1. If there is no second-smallest number (that is, if no number satisfies the definition of "second-smallest" given in Exercise 2 of Lab 1), the return value is the empty list. If there is exactly one second-smallest number, the return value is simply this number (***not*** a list). If there is more than one second-smallest number, the return value is a list containing all the second-smallest numbers.

Your submitted file must be named `ex4.scm`.

Examples:

```
> (secondsmallest '(4 1 0 10))
1

> (secondsmallest '(2 1 3))
2

> (secondsmallest '(3 3 3 2))
(3 3 3)

> (secondsmallest '(7 7 8 8))
()
```

Exercise 5 [10 marks]

Write a function called **`2-norm`** that takes one parameter, a list `v` representing a vector of numbers. The return value is the 2-norm (Euclidean norm) of the vector – that is, the square root of the sum of the squares of the numbers in `v`.

For full marks, you must **not** use recursion, and you must **not** define any helper functions. Note that the built-in function `sqrt` will be useful for this exercise.

Your submitted file must be named `ex5.scm`.

Examples:

```
> (2-norm '(-3 5 5 -5 4))
10

> (2-norm '(4 -3 6 8 2 -3))
11.74734012447073
```

Exercise 6 [12 marks]

Write a function called **`compose`** that takes one parameter, a list `L` of unary functions (a unary function is a function that takes one parameter). The return value is a function that is the composition of the functions in the list. When `L` is empty, the return value is the identity function.

Your submitted file must be named `ex6.scm`.

Examples:

```
> ((compose ()) 'x)
x

> ((compose '(sqrt car)) '(4 5 7))
2

> ((compose '(null? cdr cdr)) '(a b))
#t

> ((compose '(car car cdr)) '(w (z) y))
z
```