Homework Assignment 2

Note 1: Solutions are expected to only use functions from the standard library that was taught. Before using a function from the standard library inquire if you are allowed to use it. If a solution uses a disallowed function, the autograder score is voided.
Note 2: to obtain full credit, your solution *must* use pattern matching; your solutions cannot use: `first`, `rest`, `car`, `cdr`, `pair-left`, or `pair-right`.

1. Implement a pair using the `struct` keyword. The name of the data-structure should be `pair` and the two fields of the data-structure should be `left` (that holds the left-hand side element) and `right` (that holds the right-hand side element). **Henceforth, we use the acronym lhs to signify left-hand side, and rhs to signify right-hand side.**

   (a) Implement a constructor function `(pair-set-left p l)` which takes a pair `p` and a "new" lhs element `l` and returns a pair that holds `l` in the lhs and the rhs of `p` at the rhs.

   (b) Implement a constructor function `(pair-set-right p r)` which takes a pair `p` and a "new" rhs element `r` and returns a new pair that holds the lhs element of `p` at the lhs and `r` at rhs.

   (c) Implement a function `(pair-swap p)` which returns a new pair that holds the rhs of `p` at the lhs, and the lhs of `p` at the rhs.

   (d) Implement a function `(pair-add p1 p2)` which adds two pairs *pointwise*, that is, at the lhs hold addition of the lhs of `p1` and the lhs of `p2`; at the rhs hold the addition of the rhs of `p1` and the rhs of `p2`. **You can only use `match*` one time. You cannot use `match`.**

2. Implement a data-structure called **name** only using `lambda` and `cond`. Your solution cannot rely on existing data-structures, such as `list` or instances of `struct`. The data-structure must hold a first name (given as a string) and a last name (given as a string) of a person. Read §2.1.3 of the SICP book, in particular the implementation of functions `cons`, `car`, and `cdr`.

   (a) Implement the constructor `(name first last)` as function `cons` is explained in §2.1.3.

   (b) Implement an accessor `(first-name n)` which takes a *name* and returns its first name.

   (c) Implement an accessor `(last-name n)` which takes a *name* and returns its last name.

   (d) Implement an accessor `(full-name n)` which returns a single string with the full name. String concatenation can be achieved using `(string-append s1 s2)`. See `string-append` in `https://docs.racket-lang.org/reference/strings.html`.

   (e) Implement an accessor `(initials n)` that returns a string that holds the first letter of the first name and the first letter of the second name. You must use `(substring str start end)` to retrieve the first letter of a string. See `substring` in `https://docs.racket-lang.org/reference/strings.html`.

3. Implement a function `(max-from n l)` that takes number `n` and a list `l` and returns the maximum number between `n` and every element of `l`. To calculate the maximum value between two values you can use `(max x y)`.

4. Implement a function `(min-from n l)` that takes number `n` and a list `l` and returns the minimum number between `n` and every element of `l`. To calculate the minimum value between two values you can use `(min x y)`.

5. Implement an auxiliary function that generalizes over functions `(min-from n l)` and `(max-from n l)`. Reimplement `min-from` and `max-from` so that both functions use the auxiliary function you created.

6. Implement a function `(count l)` that takes a list `l` and returns the number of the elements in `l`.

7. Implement a function `(sum l)` that takes a list `l` of numbers and returns the summation of every element of `l`.

8. Implement a function `(occurrences x l)` that counts how many times `x` occurs in list `l`.

9. Implement the *norm*, function `(norm l)`, of a list of numbers, which should implement the following expression:

$$norm([i_1, i_2, \ldots, i_n]) = \sqrt{i_1^2 + i_2^2 + \cdots + i_n^2}$$

Hints:

- Break down the problem into several smaller problems that are easier to solve on their own.
- Use function `(sqrt x)` to calculate $\sqrt{x}$.
- Try squaring every element of a list with `map`.