Exercise 7: Streams and Continuations

Starter code

Download these files from Quercus files.

- `Exercises/ex7/ex7.rkt`
- `Exercises/ex7/stream.rkt`

Submission instructions

Submit the file `ex7.rkt` to MarkUs: https://mcsmark.utm.utoronto.ca/csc324f20/

But before you submit, check that:

- Your code does not have any syntax error. If you only completed part of the exercise, replace any possibly syntactically incorrect code with the starter code.
- The `(provide ...)` code in your Racket file is left intact. Otherwise we won't be able to run our tests!

Task 1: More Stream Functions [3 pt]

For this task, we will be using the stream implementation from this week's notes and lecture. Write the following two stream functions:

- `s-map` that has the same signature as the list function `map`, but for streams.
- `s-filter` that has the same signature as the list function `filter`, but for streams.
- `s-stutter` that takes a stream and a number `n`, and repeats every element of the stream `n` times. For example, calling `stutter` with a stream containing `1 2 3` and `n = 2` will result in the stream containint `1 1 2 2 3 3`.

These stream functions should follow the lazy evaluation order for streams, meaning that the functions should work for infinite streams.

Recall that you may write as many helper functions as you need. In particular, you may want to write some helper functions for the `s-stutter` function.

Task 2: Continuations [7 pt]

Write a function `continuations` that computes and displays a representations of the continuations of every subexpression of an input datum.

We'll use a very simple grammar for our datum here: only numeric literals and nested binary `+` function calls are allowed. You may assume all inputs are both syntactically- and semantically-valid.

We'll represent a continuation by a Racket datum that uses the special symbol `'_` to represent where to put the value of the subexpression. For example, in the expression `(+ (+ 3 4) 9)`, we represent the continuation of the `4` as the Racket datum `'(+ (+ 3 _) 9)`.

Warning: even though there's only one task on this exercise, the nature of continuations adds enough complexity that a naive recursive approach runs into some trouble. We've provided a fairly detailed design in the starter code, and some discussion about the technical challenges. **However, you're welcome to use your own approach, as long as it adheres to the global restrictions for exercises in this course.**

Please note that you *may* use `eval` on this exercise. Our starter code has used it in one place , and you may choose to use it in other places as well, especially if you use a different approach.

As an aside, once you complete this exercise, you may find it interesting to attempt this problem in an imperative, mutating style/language—it wasn't immediately clear to us how this more familiar setting would actually make this problem easier!