CS450: Structure of Higher Level Languages

Assignment 5, part 1

Due: Monday, November 16

Taken from assignments by Profs. Carl Offner and Ethan Bolker

Elementary computations with streams

These exercises concern streams and delayed evaluation, and provide some practice using them in a remarkable and non-trivial example – computing the decimal digits of π.

Guidelines

For this part of the assignment, you will hand in one file: `streams.scm`. The theoretical questions will be answered in a single file that will be submitted into gradescope as an additional file. Use the macro definition to implement streams:

```
(define-syntax cons-stream
  (syntax-rules ()
    ((cons-stream head tail)
     (cons head (delay tail)))))
```

This file should begin with some definitions from the textbook, which are included in the handout `pi.scm`. You should copy these definitions into the top of your file `streams.scm` (in addition to any other definitions you might find useful before starting the code for the various problems below). This is important: `streams.scm` should not load any other file – it should be a self-contained file that contains any code it needs.

Please note: nowhere in the code for this assignment do you need to use `set!` or any other operator that changes the value of a variable. Sometimes informally, I write in the paper of "changing the value" of some variable. What that always really means is a recursive call in which the new value is passed. But no variable actually changes its value.

Please be careful about this. The whole point of this assignment is to use streams and recursion. If you read what I have written here carefully and do what I suggest, I think you will be amazed at how simple and clear the code for these procedures is.

1. Define a function `(display-n stream n)` that prints the first n elements of stream, each on a separate line.
2. Exercise 3.50 (page 324) This exercise is pretty simple, provided you follow the suggestion in the book. In fact, you can forget about checking for the empty stream – we will only use this procedure for infinite streams.

   After you get this working (and test it), you should answer the following questions in the additional file you upload to Gradescope:

   (a) What is the purpose of the apply and the cons in the last two lines of the procedure? Why could you not just leave them out and replace the last two lines by the line

   ```
   (stream-map proc (map <??>
   argstreams))))
   ```

(b) What about replacing the last two lines by this:

```
(apply stream-map proc (map <??>
argstreams))))
```

Please note: I'm not asking for answers such as "It works", or "It doesn't work". I'm looking for explanations. And the explanations have to be correct. (Prof. Offner's comment): In the past, I have had students write things that not only were wrong, but that they could easily have seen were wrong if they had just tried them out.

A good answer to this question will take some careful explaining. And what I wrote in an earlier assignment still holds: any explanation you give about how Scheme works has to be based explicitly on the Scheme language definition R5RS. If it isn't, then the explanation is flat-out wrong.

Your answer should be in the form of an extended Scheme comment.

3. As explained in section 3.5.2, we can define an infinite stream of ones and use this to define the stream of positive integers:

```
(define ones (cons-stream 1 ones))

(define integers (cons-stream 1 (add-streams ones integers)))
```

Type in these definitions and verify that they work by using the display-n procedure. Comment the tests out before you submit. **The actual task:** Generate the stream `notdiv-235` of all integers that are not divisible by any of the numbers 2, 3, or 5. (Use `stream-filter`.)