CS 61A   Summer 2010   Week 6B Lab
Wednesday 7/28 Afternoon

1. What is the type of the value of `(delay (+ 1 27))`? What is the type of the value of `(force (delay (+ 1 27)))`?

2. Evaluation of the expression

```
(stream-cdr (stream-cdr (cons-stream 1 '(2 3))))
```

produces an error. Why?

3. Consider the following two procedures.

```
(define (enumerate-interval low high)
  (if (> low high)
    '()
    (cons low (enumerate-interval (+ low 1) high)) ) )

(define (stream-enumerate-interval low high)
  (if (> low high)
    the-empty-stream
    (cons-stream low (stream-enumerate-interval (+ low 1) high)) ) )
```

What's the difference between the following two expressions?

```
(delay (enumerate-interval 1 3))
(stream-enumerate-interval 1 3)
```

4. An unsolved problem in number theory concerns the following algorithm for creating a sequence of positive integers $s_1, s_2, ...$

```
Choose $s_1$ to be some positive integer.
For $n > 1$,
    if $s_n$ is odd, then $s_{n+1}$ is $3s_n + 1$;
    if $s_n$ is even, then $s_{n+1}$ is $s_n/2$.
```

No matter what starting value is chosen, the sequence always seems to end with the values 1, 4, 2, 1, 4, 2, 1, ... However, it is not known if this is always the case.

4a. Write a procedure `num-seq` that, given a positive integer `n` as argument, returns the stream of values produced for `n` by the algorithm just given. For example, `(num-seq 7)` should return the stream representing the sequence 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...

4b. Write a procedure `seq-length` that, given a stream produced by `num-seq`, returns the number of values that occur in the sequence up to and including the first 1. For example, `(seq-length (num-seq 7))` should return 17. You should assume that there is a 1 somewhere in the sequence.

5. Suppose we have the following parallel execution:

```
(define x 11)
(parallel-execute (lambda () (if (even? x) (set! x (+ x 1)) (set! x (- x 1))))
                  (lambda () (set! x (* 2 x))))
```

What are all possible values of `x` at the end? You can load ~`cs61a/lib/concurrent.scm` and then try running this code to see some of the possibilities, but you most likely won't be able to see all of them since the values depend on the order in which the atomic steps interleave and some of the possible results are very rare.

What are the correct results?

6. Load ~`cs61a/lib/concurrent.scm` Create a deadlock situation with exactly one serializer and one mutex. Test your code with `parallel-execute`