CS 61A Summer 2009 Homework 5

Topic: Vectors, Streams, Evaluator

Lectures: Monday 7/13, Tuesday 7/14, Wednesday 7/15, Thursday 7/16

Reading: Abelson and Sussman: 3.1, 3.2, 3.3

1. (**Graded by Correctness**) Fill in the blanks in the following code so that the code behaves as followed: Part A:

```
> (define foo (list (cons 1 nil) 2))
> (define bar (list (cdr foo) (cdr foo)))

> (set-cdr! _____________ bar)
> foo
 ((1 (2) (2)) 2)
```

Part B:

```
> (define foo (list 1 2))
> (define bar (cons foo (cons 3 nil)))

> (append! bar _________________)
> bar
 ((1 2) 3 2)
```

DO NOT create new pairs in your code. You will receive 0 if you do. Be sure to draw box and pointer diagrams to help you.

2. Write a procedure `swap-content` that takes two pairs as input. It will exchange the content of the two pairs while preserving original pointers. For example:

```
> (define foo (cons 1 2))
> (define bar (cons 3 4))
> (swap-content foo bar)
 okay
> foo
 (3 . 4)
> bar
 (1 . 2)
```

3. Write a procedure `(vector-swap! vec1 vec2)` that takes two vectors. It should swap every element of vec1 and vec2. For example:

```
> (define foo #(1 2 3 4))
> (define bar #(5 6 7 8))
> (vector-swap! foo bar)
 okay
> foo
 #(5 6 7 8)
> bar
 #(1 2 3 4)
```

You can assume that the two input vectors will have the same length.

4. Write a procedure `(repeat? ls n)` that takes a **list of integers between 0 and** `n` as input and checks whether any integers appear more than once in the list. **Also, repeat? has to run in linear time**. The time constraint means that you can't just take the first element, scan through the whole list to see if it repeats, and then take the second element, scan through the whole list again, etc. HINT: use a vector to store what numbers you have encountered so far.

5. Recall the `(count-ways x y)` program from beginning of the class that takes a (x,y) coordinate and finds how many ways there is of getting to the origin by only moving left or down.

```
(define (count-ways x y)
  (if (or (= x 0) (= y 0)) 1
      (+ (count-ways (- x 1) y)
                (count-ways x (- y 1)))))
```

You have seen how memoization improved the runtime of the `fibs` program in lecture. Use vectors to similarly create a memoized version of the `count-ways` program.

6. Abelson and Sussman, 3.51, 3.53, 3.54, 3.55, 3.56 (typo on 3.56, it should be `(scale-stream S 5)` and not `(scale-stream 5 S)`

7. Streams are used heavily in video and audio transmission (hence, *streaming* a video). By delaying evaluation until when we want the data, Streams give us the illusion of having an infinitely large amount of data in our computer just as how when you stream a radio station, you get the illusion that the entire broadcast is stored in your computer.

So suppose the raw audio transmission from a radio is stored as a stream of numbers (representing amplitude of the signal at different times), write a procedure `(smooth-stream stream n)` that takes a stream and return a new stream where the first element of the returned stream is the average of the 1st to n-th element of the input stream, the 2nd element of the returned stream is the average of the 2nd to $n+1$-th element of the input stream, etc.

For example:

```
> (define ints (cons-stream 1 (stream-map 1+ ints)))
> (define smoothed-ints (smooth-stream ints 4))
> (ss smoothed-ints)
 (2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5 ...)
```

Because (1+2+3+4)/4 = 2.5, (2+3+4+5)/4 = 3.5, (3+4+5+6)/4 = 4.5, etc. In real world, noise can enter our signal and smoothing the signal dampens the effect of noises.

8. What are the first 15 elements of the following stream:

```
(define mystery
  (cons-stream 0
     (cons-stream 1
        (interleave
            (stream-map (lambda (x) (word 0 x)) mystery)
            (stream-map (lambda (x) (word 1 x)) mystery)))))
```

Try to figure it out without using the Scheme interpreter. Can you describe what stream `mystery` is

concisely?

9. Copy the file ~cs61a/lib/mceval-step1.scm to your home directory.

Modify the `mc-eval` procedure such that it checks whether the user typed in a list of length at least 2. If so, then mc-eval should return the second element of the list. Otherwise, mc-eval should just return what the user typed.

10. Copy the file ~cs61a/lib/mceval-step2.scm to your home directory.

Answer the following question: On line 17 with `((primitive-proc? exp) ...)`, we used `eval` On line 19 with `(map mc-eval (cdr exp))`, we used `mc-eval`

Why not use `eval` for line 19?

11. Copy the file ~cs61a/lib/mceval-step3.scm to your home directory.

Often times in Scheme, we would like to remove a variable name from the world completely. Implement a special form `(remove! var)` in our new Scheme such that it removes the variable and the value associated from the global environment list.