CS 61A, Fall 1997
Midterm 3
Professor Harvey

Problem #1

What will the Scheme interpreter print in response to each of the following expressions? Also, draw a "box and pointer" diagram for the result of each expression. Hint: It'll be a lot easier if you draw the box and pointer diagram first!

```
(let ((x (list 1 2 3 4)))
  (set-cdr! x (cdr x))
  x)

(let ((x (list 1 2 3 4)))
  (set-car! (cddr x) (cdr x))
  x)

(let ((x (list 1 2 3 4)))
  (set-cdr! (cdr x) (cadr x))
  x)
```

Problem #2

An improper list is a data structure made of pairs, in which the cdr of some pair is anything other than a pair or the empty list.

Write the procedure deep-fix! that takes a pair as its argument, examines the data structure headed by that pair, and replaces any improper cdr with the empty list. For example:

> (deep-fix! '(1 2 (3 4 .5) (6 (7 . 8) 9) 10 . 11))
(1 2 (3 4) (6 (7) 9) 10)

(In the example I've broken the rule about not mutating quoted data, in order to make the example easier for you to read.)

NOTE: Do not allocate any new pairs in your solution. Modify the existing pairs.

Problem #3

In weaving, a horizontal thread is pulled through a bunch of vertical threads. The horizontal thread passes over some of the vertical ones, and under others. The choice of over or under determines the pattern of the weave.

We will represent a pattern as a list of the words OVER and UNDER, repeated as needed. Here's an example:
(OVER OVER UNDER OVER UNDER UNDER)
The pattern may be of any length (it depends on the desired width of the woven cloth), but it must contain at least one OVER and at least one UNDER.

Write a Scheme expression to compute the (infinite) stream of all possible patterns.

Problem #4

We're going to add to the adventure game a new kind of person, called a wizard. Wizards can move around in the same way other people do, but they also remember every place they've ever seen. Once a wizard has been someplace, he can return to that place directly, by magic.

The wizard class will accept a revisit message, whose argument is the name of a place where the wizard has already been. If the argument is valid, the wizard will go directly to the place with that name. If not, print an error message.

Implement the wizard class in OOP notation.

Here's an abbreviated definition of the person class, to jog your memory:

```
(define-class (person name place)
  (method (person? #t)
  (method (look-around) ...)
  (method (exits) (ask place 'exits))
  (method (go direction) ...)
  (method (go-directly-to new-place) ...)
  ...)
```

Problem #5

(a) Suppose we say

```
>(define baz 10)
>(define s (make-serializer))
>(parallel-execute (s (lambda() (set! baz (/ baz 2))))
                  (s (lambda() (set! baz (+ baz baz)))))
```

What are the possible vales of baz after this finishes?

(b) Now suppose that we change the example to leave out the serializer, as follows:

```
>(define baz 10)
>(parallel-execute (s (lambda() (set! baz (/ baz 2))))
                  (s (lambda() (set! baz (+ baz baz)))))
```

What are all the possible values of baz this time?
