# CS61A Notes 14 – Your Father's Worst Nightmare [Solutions v1.0]

**Of Pills And Other Euphoric Items**

**QUESTIONS: What is printed at each line?**

1. 
```
> (define x (+ 2 3))
> x ==> 5
> (define y ((lambda(a) a) (* 3 4)))
> y ==> 12
> (define z ((lambda(b) (+ b 10)) y))
> z ==> 22
```
Note that, without mutation or assignment, lazy evaluation works just like normal STk.

2. 
```
> (define count 0)
> (define (foo x y) (x y))
> (define z (foo (lambda(a) (set! count a) (* a a))
                 (begin (set! count (+ 1 count)) count)))
> count ==> 0
> z ==> infinite loop (doh!)
```

3. 
```
> (define count 0)
> (define (incr!) (set! count (+ count 1)))
> (define (foo x)
   (let ((y (begin (incr!) count)))
      (if (<= count 1)
          (foo y)
          x)
> (foo 10) ==> infinite loop
```

---

**Nondeterministic and Indecisive**

**QUESTIONS**

1. **Suppose we type the following into the amb evaluator:**
```
> (* 2 (if (amb #t #f #t)
           (amb 3 4)
           5))
```
**What are all possible answers we can get?**
6, 8, 10, 6, 8

2. **Write a function `an-atom-of` that dispenses the atomic elements of a deep list (not including empty lists). For example,**
```
> (an-atom-of '((a) ((b (c))))) ==> a
> try-again ==> b
(define (an-atom-of ls)
   (cond ((null? ls) (amb))
         ((atom? ls) ls)
         (else (amb (an-atom-of (car ls))
                    (an-atom-of (cdr ls))))))
```

3. **Use `an-atom-of` to write `deep-member?`.**

```
(define (deep-member? X ls)
   (let ((maybe-x (an-atom-of ls)))
      (require (equal? x maybe-x))
      #t))
```

4. **Fill in the blanks:**

```
> (define (choose-member L R)
   (cond ((null? R) (amb))
         ((= (car L) (car R)) (car L))
         (else (amb (choose-member L (cdr R))
                    (choose-member (cdr L) R)))))
> (choose-member '(1 2 3) '(4 2 3))
3

> try-again
2

> try-again
2
```