# CS 61A, Fall 2001
# Midterm #3
# Professor Brian Harvey

**Question 1 (6 points):**

Suppose that `my-tree` is the following tree:

```
   1
  /|\
 / | \
2  5  3
|    / \
4   6   7
```

You are given the following procedures:

```
(define (foo tree)
  (if (null? (children tree))
      (datum tree)
      (foo (car (children tree)))))

(define (baz tree)
  (map datum (children tree)))

(define (garply tree)
  (make-tree 1 (map garply (children tree))))
```

What is the value returned by each of the following? If it's a tree, draw a picture of it, as above; if not, show how Scheme will print the value.

(a) `(foo my-tree)`

(b) `(baz my-tree)`

(c) `(garply my-tree)`

**Question 2 (6 points):**

```
(define-class (scoop flavor)
  ; maybe (parent (cone)) - see part (A) below
  )

(define-class (vanilla)
  (parent (scoop 'vanilla)))
(define-class (chocolate)
  (parent (scoop 'chocolate)))

(define-class (cone)
  ; maybe (parent (scoop)) - see part (A) below
  (instance-vars (scoops '()))
  (method (add-scoop new)
    (set! scoops (cons new scoops)))
  (method (flavors)
    (map ___see (B) below___ scoops)))
```

(A) Which of the `parent` clauses shown above should be used?

_____The `scoop` class should have `(parent (cone))`.

_____The `cone` class should have `(parent (scoop))`.

_____Both.

_____Neither.

(B) What is the missing expression in the `flavors` method?

(C) Which one of the following is the correct way to add a scoop of vanilla ice cream to a cone named `my-cone`?

_____`(ask my-cone 'add-scoop 'vanilla)`

_____`(ask my-cone 'add-scoop vanilla)`

_____`(ask my-cone 'add-scoop (instantiate 'vanilla))`

_____`(ask my-cone 'add-scoop (instantiate vanilla))`

**Question 3 (4 points):** Suppose we want to write a procedure `prev` that takes as its argument a procedure `proc` of one argument. `Prev` returns a new procedure that returns the value returned by *the previous call to* `proc`. The new procedure should return **`#f`** the first time it is called. For example:

```
> (define slow-square (prev square))
> (slow-square 3)
#f
> (slow-square 4)
9
> (slow-square 5)
16
```

Which of the following definitions implements `prev` correctly? **Pick only one.**

```
______ (define (prev proc)
         (let ((old-result #f))
           (lambda (x)
              (let ((return-value old-result))
                (set! old-result (proc x))
                return-value))))

______ (define prev
         (let ((old-result #f))
           (lambda (proc)
             (lambda (x)
               (let ((return-value old-result))
                 (set! old-result (proc x))
                 return-value)))))

______ (define (prev proc)
         (lambda (x)
           (let ((old-result #f))
             (let ((return-value old-result))
               (set! old-result (proc x))
               return-value))))

______ (define (prev)
         (let ((old-result #f))
           (lambda (proc)
             (lambda (x)
               (let ((return-value old-result))
                 (set! old-result (proc x))
                 return-value)))))
```

**Question 4 (8 points):**

Consider the following computation:

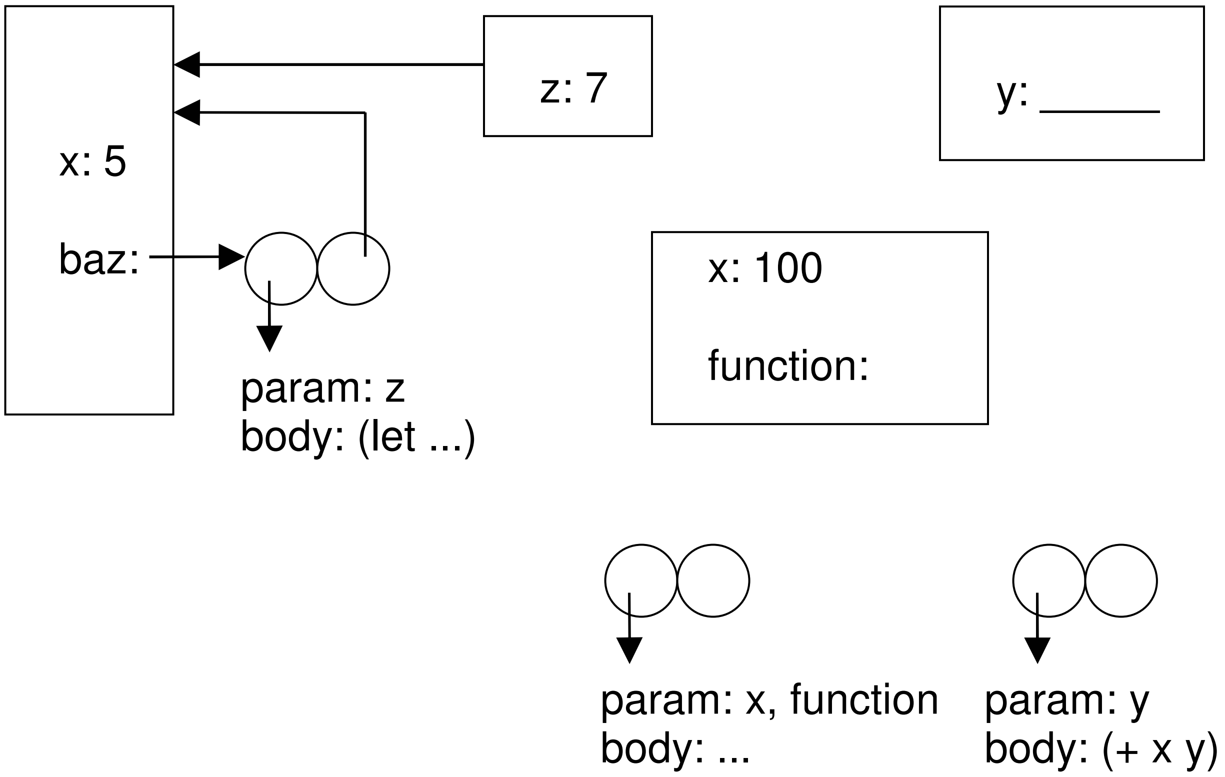
```
> (define x 5)

> (define (baz z)
    (let ((x 100)
          (function (lambda (y) (+ x y))))
      (function (* z z))))

> (baz 7)
```

(a) Here is an environment diagram representing the result of this computation, but with a few missing pieces. Complete the diagram by showing the following:

- The values of `y` and `function`.
- Arrows from the right bubbles of two procedures.
- Arrows from two frames showing what environments they extend.



(b) What is the value returned by `(baz 7)`? ________________

**Question 5 (7 points):**

Given the following definition:

```
(define ab-stream
  (cons-stream 'a
    (cons-stream 'b
      (interleave
        (stream-map (lambda (elt) (word elt 'a))
                    (stream-cdr ab-stream))
        (stream-map (lambda (elt) (word elt 'b))
                    (stream-cdr ab-stream))))))
```

(a) What are the first 10 elements of `ab-stream`?

(________________ ________________ ________________ ________________

________________ ________________ ________________ ________________

________________ ________________)

(b) What position is the `bbbbbbbb` (8 `b`'s)? (E.g., if it's the first element, write "1", if it's the second element, write "2", etc.) You may leave the answer as a math expression (e.g., 42! or $5^{97}$) if you prefer.

**Question 6 (8 points):**

Write the procedure `same-parity!` that takes as its argument a list of integers, and modifies the list by mutation so that it contains only those elements of the same parity (even or odd) as the first element:

```
> (same-parity! (list 3 8 1 5 2 9 4 6 7 1))
(3 1 5 9 7 1)
```

**Do not allocate any new pairs!** Remove elements of the wrong parity from the argument list by mutation. Don't forget to return the resulting list as the return value from your procedure.

You may use this helper procedure:

```
(define (same x y)
  (= (remainder x 2) (remainder y 2)))
```