Repeat of Rules: Eval

- \textit{name} - Look up \textit{name} in the current environment, if found return value, otherwise lookup in enclosing (parent) environment.

- \texttt{(lambda (params) body)} - Create double bubble with code ptr to \textit{params} and \textit{body} and env ptr to current environment.

- \texttt{(define name value)} - Evaluate \textit{value} and then create/replace binding for \textit{name} with the result.

- \texttt{(set! name value)} - Evaluate \textit{value} and then replace the first binding for \textit{name} in the chain of environments, starting with the current env.

- \texttt{(proc args ...) } - Evaluate \textit{proc} and \textit{args} in the current environment, then apply.

- Otherwise – Follow the correct rule (numbers, if, cond, begin, quote, etc.)

Repeat of Rules: Apply

- Step 1 - Drop a new frame

- Step 2 - Link frame pointer of new frame to environment pointed to by env pointer of double bubble being applied.

- Step 3 - Bind \textit{params} of double bubble in the new frame.

- Step 4 - Eval the \textit{body} in the new frame.
Three Counter Attempts

1. (define make-count-proc-1
   (lambda (f)
     (lambda (x)
       (let ((count 0))
         (cond ((eq? x 'count) count)
               (else
                (set! count (+ count 1))
                (f x))))))

(define sqrt-c-1
  (make-count-proc-1 sqrt))

(sqrt-c-1 4)
(sqrt-c-1 'count)
2. (define make-count-proc-2
   (lambda (f)
      (let ((count 0))
        (lambda (x)
           (cond ((eq? x 'count) count)
                (else
                 (set! count (+ count 1))
                 (f x))))))

   (define sqrt-c-2
      (make-count-proc-2 sqrt))
   (define sqr-c-2
      (make-count-proc-2 square))

   (sqrt-c-2 4)
   (sqrt-c-2 'count)

   (sqr-c-2 4)
   (sqr-c-2 'count)
3. (define make-count-proc-3
   (let ((count 0))
     (lambda (f)
       (lambda (x)
         (cond ((eq? x 'count) count)
               (else
                (set! count (+ count 1))
                (f x))))))

(define sqrt-c-3
  (make-count-proc-3 sqrt))
(define sqr-c-3
  (make-count-proc-3 square))

(sqrt-c-3 4)
(sqrt-c-3 'count)

(sqr-c-3 4)
(sqr-c-3 'count)
4. The procedure `last-pair` returns the last pair of a list (guaranteed to have `()` in the cdr).

```
(define (list-inserters lst)
  (let ((last (last-pair lst)))
    (list (lambda (x)
              (set-cdr! lst (cons x (cdr lst)))
              lst)
          (lambda (y)
              (set-cdr! last (cons y '()))
              (set! last (cdr last))
              lst))))

(define the-list (list 1 3 4))

(let ((ins (list-inserters the-list)))
  ((list-ref ins 0) 2)
  ((list-ref ins 1) 5))
```

Finish the environment diagram.

![Environment Diagram]

```