6.001 Recitation 18: Quiz 2 Review

RI: Gerald Dalley, dalleyg@mit.edu, 18 Apr 2007

http://people.csail.mit.edu/dalleyg/6.001/SP2007/

Announcements / Schedule

- Quiz 2 is being held in Walker (50-340) from 7:30-9:30pm
- Old quizzes are generally a good study guide (http://web.mit.edu/amdragon/www/6.001-quizzes/)
- 2 pages of notes are allowed (8.5×11).
- Everything up to 6 April is fair game
- Key topics: symbols, tagged data, mutation, trees, environment model, procedures with local state, message passing procedures.

Procedures with State

Suppose we want a procedure `prev` with the property that it returns the previous value it was applied to. For example:

```
(prev 1)    ==> undef
(prev 2)    ==> 1
(prev 3)    ==> 2
```

Which of the following attempts to implement `prev` works correctly? (you probably want to draw an environment diagram for each)

```
(define prev1
  (let ((x 'undef))
    (lambda (y)
      (let ((z x))
        (set! x y)
        z))))

(define prev2
  (let ((x (list 'undef)))
    (lambda (y)
      (let ((z x))
        (set-car! x y)
        (car z)))))

(define prev3
  (let ((x (cons 'undef 'undef)))
    (lambda (y)
      (set-car! x (cdr x))
      (set-cdr! x y)
      (car x))))
```

Suppose we have a correctly implemented version of `prev`. What is the result of evaluating the expressions below? Assume a fresh evaluation of our `prev` definition for each.

```
(let ((x (prev prev)))
  (((prev +) prev) 1 2))
```

Answer:

```
(let ((foo (prev prev)))
  (((foo +) foo) 1 2))
```

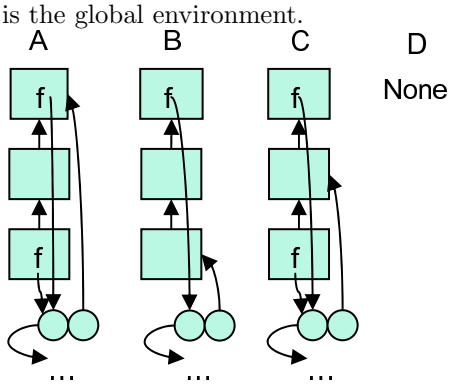
Answer:

```
(let* ((g (prev prev))
       (f (prev prev)))
  (((f +) f) 1 2))
```

Answer:

Environmental Matching

Match each of the following code blocks to one of the environment models. Assume that the topmost frame is the global environment.




```
(define f (lambda () ...))
(let (( ))
  (let ((f f)) ))
```

Environment diagram:

```
(define f
  (let (( ))
    (lambda (f) ...)))
(f f)
```

Environment diagram:

```
(define f
  (let (( ))
    (let ((f (lambda () ...))) f)))
```

Environment diagram:

```
(define f
  (let (( ))
    (lambda ()
      (define f f)
      ...)))
(f)
```

Environment diagram:

```
(define f
  (let (( ))
    (let (( ))
      (lambda () ...))))
```

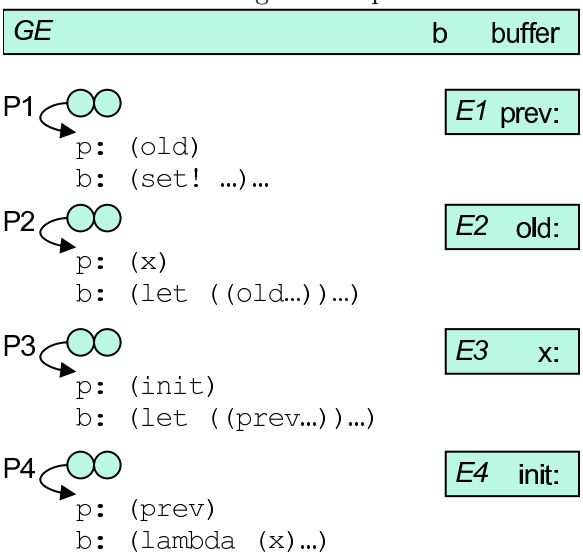
Environment diagram:

Wiring

Consider the following expressions:

```
(define (buffer init)
  (let ((prev init))
    (lambda (x)
      (let ((old prev))
        (set! prev x)
        old))))
(define b (buffer 10))
(b 20)
```

Here are the parts of the environment diagram that results from evaluating these expressions:



For each of the following frames, indicate the lowest frame of the enclosing environment, choosing one of GE, E1, E2, E3, E4, none, or not shown.

Frame	Enclosing Environment
GE	
E1	
E2	
E3	
E4	

For each of the following procedure objects, indicate the lowest frame of the enclosing environment, choosing one of GE, E1, E2, E3, E4, none, or not shown.

Procedure	Enclosing Environment
P1	
P2	
P3	
P4	

For each of the following variable names, indicate the value to which it is bound in the specified environment at the end of the evaluation of the expressions. Indicate the value by choosing one of GE, E1, E2, E3, E4, E5 or P1, P2, P3 or a symbol, a number, a list of numbers, or a boolean value.

Variable	Environment	Value
b	GE	
buffer	GE	
prev	E1	
old	E2	
x	E3	
init	E4	

Solutions

Announcements / Schedule

- Quiz 2 is being held in Walker (50-340) from 7:30-9:30pm
- Old quizzes are generally a good study guide (`http://web.mit.edu/amdragon/www/6.001-quizzes/`)
- 2 pages of notes are allowed (8.5×11).
- Everything up to 6 April is fair game
- Key topics: symbols, tagged data, mutation, trees, environment model, procedures with local state, message passing procedures.

Procedures with State

Suppose we want a procedure `prev` with the property that it returns the previous value it was applied to. For example:

```
(prev 1)    ==> undef
(prev 2)    ==> 1
(prev 3)    ==> 2
```

Which of the following attempts to implement `prev` works correctly? (you probably want to draw an environment diagram for each)

`prev1` and `prev3` work as desired. `prev2` doesn't work because the `set-car!` affects both `x` and `z`.

```
(define prev1
  (let ((x 'undef))
    (lambda (y)
      (let ((z x))
        (set! x y)
        z))))

(define prev2
  (let ((x (list 'undef)))
    (lambda (y)
      (let ((z x))
        (set-car! x y)
        (car z)))))

(define prev3
  (let ((x (cons 'undef 'undef)))
    (lambda (y)
      (set-car! x (cdr x))
      (set-cdr! x y)
      (car x))))
```

Suppose we have a correctly implemented version of `prev`. What is the result of evaluating the expressions below? Assume a fresh evaluation of our `prev` definition for each.

```
(let ((x (prev prev)))
  (((prev +) prev) 1 2))
```

Answer: 3

```
(let ((foo (prev prev)))
  (((foo +) foo) 1 2))
```

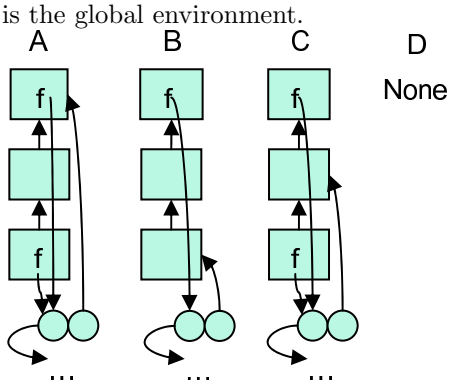
Answer: error

```
(let* ((g (prev prev))
       (f (prev prev)))
  (((f +) f) 1 2))
```

Answer: 3

Environmental Matching

Match each of the following code blocks to one of the environment models. Assume that the topmost frame is the global environment.




```
(define f (lambda () ...))
(let (( ))
  (let ((f f)) ))
```

Environment diagram: A

```
(define f
  (let (( ))
    (lambda (f) ...)))
(f f)
```

Environment diagram: C

```
(define f
  (let (( ))
    (let ((f (lambda () ...))) f)))
```

Environment diagram: C

```
(define f
  (let (( ))
    (lambda ()
      (define f f)
      ...)))
(f)
```

Environment diagram: C

```
(define f
  (let (( ))
    (let (( ))
      (lambda () ...))))
```

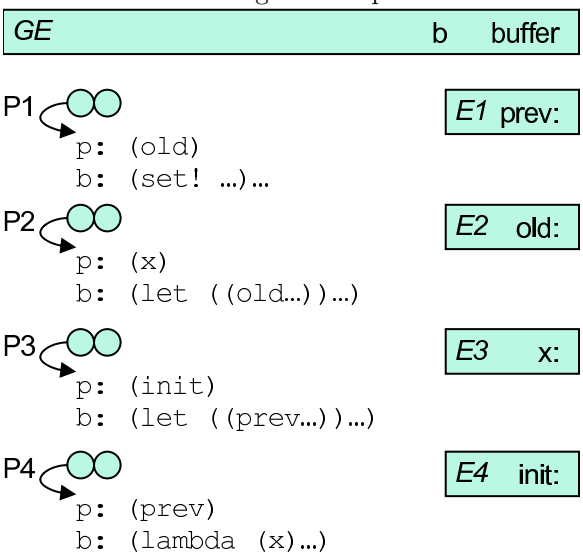
Environment diagram: B

Wiring

Consider the following expressions:

```
(define (buffer init)
  (let ((prev init))
    (lambda (x)
      (let ((old prev))
        (set! prev x)
        old))))
(define b (buffer 10))
(b 20)
```

Here are the parts of the environment diagram that results from evaluating these expressions:



For each of the following frames, indicate the lowest frame of the enclosing environment, choosing one of GE, E1, E2, E3, E4, none, or not shown.

Frame	Enclosing Environment
GE	none
E1	E4
E2	E3
E3	E1
E4	GE

For each of the following procedure objects, indicate the lowest frame of the enclosing environment, choosing one of GE, E1, E2, E3, E4, none, or not shown.

Procedure	Enclosing Environment
P1	E3
P2	E1
P3	GE
P4	E4

For each of the following variable names, indicate the value to which it is bound in the specified environment at the end of the evaluation of the expressions. Indicate the value by choosing one of GE, E1, E2, E3, E4, E5 or P1, P2, P3 or a symbol, a number, a list of numbers, or a boolean value.

Variable	Environment	Value
b	GE	P2
buffer	GE	P3
prev	E1	20
old	E2	10
x	E3	20
init	E4	10