Solutions to Sample Final Exam

Problem 1

((a b c) a b c)

((a) a)

((a b c) a b c)

((a) a)
**Problem 2**

\[
\begin{aligned}
\text{(define (map-two-streams f s1 s2)} \\
\quad \text{(cons-stream (f (stream-car s1)) (stream-car s2))} \\
\quad \text{(map-two-streams f (stream-cdr s1) (stream-cdr s2)))}
\end{aligned}
\]

(\text{define (min-two-streams s1 s2)} \\
\quad \text{(map-two-streams min s1 s2)})

There is one multiplication done to compute each element of the stream, so the nth element will need n multiplications. However, if the answer was already computed, no multiplications would be necessary. Or if the (n-1)th element had been computed, just one more multiplication is necessary.

Without memoization: The nth element still needs n multiplications to be computed. However, if we ask for the (n+1)th element after computing n, we’ll need to redo all of the prior n multiplications since we’re not using memoization.

**Problem 3**

(\text{define (make-opl-lecturer name birthplace threshold)} \\
\quad (let ((person (make-person name birthplace threshold))) \\
\quad \text{(lambda (message) \\
\quad \quad (cond ((eq? message 'bring-cookies-to-exam) \\
\quad \quad \text{(lambda (self) \\
\quad \quad \quad (let ((location (ask self 'place))) \\
\quad \quad \quad \text{(ask self 'move-to cookie-store)} \\
\quad \quad \quad \text{(ask self 'take cookies)} \\
\quad \quad \quad \text{(ask self 'move-to location)))}) \\
\quad \quad \text{((eq? message 'say) \\
\quad \quad \text{(lambda (self stuff) \\
\quad \quad \quad \text{(ask person 'say stuff)} \\
\quad \quad \quad \text{(ask person 'say '(Scheme is fun!)})}) \\
\quad \quad \text{(else (get-method person message))))))})
**Problem 4**

Insert before application? in mc-eval:

```
((infix? exp)
  (mc-apply (mc-eval (infix-operation exp) env)
    (list-of-values (infix-operands exp) env)))
```

Rest of necessary code:

```
(define (infix? exp)
  (or (eq? (cadr exp) 'uml:+)
      (eq? (cadr exp) 'uml:-)
      (eq? (cadr exp) 'uml:*)
      (eq? (cadr exp) 'uml:/)))

(define (infix-operator exp) (cadr exp))

(define (infix-operands exp)
  (cons (car exp) (cddr exp)))
```

**Problem 5**

```
(define (make-frame variables values)
  (if (null? variables)
      nil
      (cons (cons (car variables) (car values))
           (make-frame (cdr variables) (cdr values)))))
```

Here’s the complete lookup-variable-value function. The underlined portions show what was changed from the original.

```
(define (lookup-variable-value var env)
  (define (env-loop env)
    (define (scan frame)
      (cond ((null? frame)
             (env-loop (enclosing-environment env)))
            ((eq? var (caar frame)) (cdar frame))
            (else (scan (cdr frame))))
    (if (eq? env the-empty-environment)
        (error "Unbound variable" var)
        (scan (first-frame env)))
  (env-loop env))
```

The following procedures would need to be changed:

- set-variable-value!
- define-variable!
- add-binding-to-frame!
- frame-variables
- frame-values