3.1

```
(define (make-accumulator num)
  (lambda (x) (begin (set! num (+ num x)) num)))

(define A (make-accumulator 5))
```

3.2

```
(define (make-monitored f)
  (let ((counter 0))
    (lambda (x)
      (cond ((equal? x 'how-many-calls?) counter)
            ((equal? x 'reset-count)
             (begin (set! counter 0) 'counter-reset))
            (else (begin (set! counter (+ counter 1)) (f x)))))))

(define s (make-monitored sqrt))
```

3.3

```
(define (make-account balance password)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
    (set! balance (+ balance amount))
    balance)
  (define (dispatch pw m)
    (if (equal? pw password)
        (cond ((eq? m 'withdraw) withdraw)
              ((eq? m 'deposit) deposit)
              (else (error "Unknown request -- MAKE-ACCOUNT"
                           m)))
        (lambda (x) '(Incorrect-Password))))
  dispatch)
```

3.4

```
(define (make-account balance password)
  (define wrongcounter 0)
  (define (call-the-cops) '911)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
    (set! balance (+ balance amount))
    balance)
  (define (dispatch pw m)
    (cond ((equal? pw password)
           (begin (set! wrongcounter 0)
                  (cond ((eq? m 'withdraw) withdraw)
                        ((eq? m 'deposit) deposit)
                        (else (error "Unknown request --
MAKE-ACCOUNT"
                                     m)))))
          ((equal? wrongcounter 7) (lambda (x) (call-the-cops)))
          (else (lambda (x) (begin
                              (set! wrongcounter (+ wrongcounter 1))
                              '(Incorrect-Password))))))
  dispatch)
```

3.7

```
(define (make-joint acc pass newpass)
  (if (number? ((acc pass 'deposit) 0))
      (lambda (password m)
        (if (equal? password newpass)
            (acc pass m)
            (lambda (x) '(Incorrect Password))))
      (lambda (y)
        (lambda (p q)
          '(Wrong Password)))))
```

3.8

```
(define f
  (let ((count 0))
    (lambda (x)
(begin (set! count (+ count x)) count))))
```