Hannah Zhang

3.1, 3.2, 3.3, 3.4, 3.7, 3.8

3.1

Write make-accumulator

```
; originally, I wrote it like this:
(define (make-accumulator2 num)
  (lambda (amount)
    (begin (set! amount (+ num amount)) amount)))
; the problem is that num is not updated each time

; answer
(define (make-accumulator num)
  (lambda (amount)
    (begin (set! num (+ num amount)) num)))
; this updates num each time
; we update num each time since it is the input to the function
"A". we do not update amount since it is not an input
```

3.2

Write make-monitored

```
; the lambda is the input given to the function s
; counter is started at 0 and updated continuously using set!
(define (make-monitored f)
  (let ((counter 0))
    (lambda (input)
      (cond ((equal? input 'reset-count) (set! counter 0))
            ((equal? input 'how-many-calls?) counter)
            (else (begin (set! counter (+ counter 1)) (f
input)))))))
```

3.3

Write a modified make-account with a password

```
; first wrote it like this
; works when password is correct but doesn't work when password
is not correct because it returns a string when it is supposed
to return a procedure

  (define (dispatch password m)
```

```
    (if (equal? password 'secret-password)
        (cond ((eq? m 'withdraw) withdraw)
              ((eq? m 'deposit) deposit)
              (else (error "Unknown request -- MAKE-ACCOUNT"
                           m)))
        "Incorrect password"))

; this is the correct way since lambda makes it a procedure
(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 (getpass) password)
  (define (dispatch given-password m)
    (if (equal? password given-password)
        (cond ((eq? m 'withdraw) withdraw)
              ((eq? m 'deposit) deposit)
              (else (error "Unknown request -- MAKE-ACCOUNT"
                           m)))
        (lambda (x) "Incorrect password")))
  dispatch)

; basically this procedure works with dispatch, which takes the
arguments the user gives, and refers it to a procedure. This is
why “incorrect password” must be in the form of a procedure, or
use the special form error
```

3.4

```
; add call the cops
; note: once call the cops is invoked, the account can not be
accessed anymore
; create a counter at the beginning using let
; first, if password is equal and counter is not 7 —> invoke
; if not equal password and counter is not 7 —> add to counter
```

```
; else, invoke call the cops
(define (make-account balance password)
  (let ((counter 0))
    (define (withdraw amount)
      (if (>= balance amount)
          (begin (set! balance (- balance amount))
                 balance)
          "Insufficient funds"))
    (define (deposit amount)
      (set! balance (+ balance amount))
      balance)
    (define (call-the-cops x) '911)
    (define (dispatch password m)
      (cond ((and (equal? password 'secret-password) (not
(equal? counter 7)))
             (begin (set! counter 0)
                    (cond ((eq? m 'withdraw) withdraw)
                          ((eq? m 'deposit) deposit)
                          (else (error "Unknown request --
MAKE-ACCOUNT"
                                       m)))))
            ((and (not (equal? password 'secret-password)) (not
(equal? counter 7)))
             (begin (set! counter (+ counter 1)) (lambda (x)
"Incorrect password")))
            (else call-the-cops)))
    dispatch))
```

3.7

```
; write joint account, which creates a joint account that
accesses the parent account but with a different password
; add a conditional to dispatch so that it processes the request
get-password and gets the password of the parent account (used
in joint-account)
(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 given-password m)
    (if (equal? password given-password)
        (cond ((eq? m 'withdraw) withdraw)
              ((eq? m 'deposit) deposit)
              ((eq? m 'get-password) password)
              (else (error "Unknown request -- MAKE-ACCOUNT"
                           m)))
        (lambda (x) "Incorrect password")))
  dispatch)

; define a second dispatch which checks if the given password
matches the joint password. If so, invoke the dispatch for the
parent account. If not, return error
; invoke dispatch2, but first make sure the password matches the
parent password when creating the joint
(define (make-joint parent-acc parent-pass joint-pass)
  (define (dispatch2 given-password f)
    (if (equal? joint-pass given-password)
        (parent-acc parent-pass f)
        (lambda (x) "Incorrect password")))
  (if (equal? parent-pass (parent-acc parent-pass
'get-password))
      dispatch2
      (error "Incorrect password - can not create joint")))
```

3.8

```
; create a method that works differently for (+ (f 0) (f 1)) and
(+ (f 1) (f 0))
; first tried this way
; if counter is 1, return the number, else return 0
; this does not work because it creates separate counters for
each instance so counter is always zero
(define (f num)
  (let ((counter 0))
    (if (equal? counter 0)
```

```
        (begin (set! counter (+ counter 1)) num)
        0)))

; this method works because there is a lambda which takes the
counter from the first instance and applies it in the second
instance
(define f
  (let ((counter 0))
    (lambda (x)
      (if (equal? counter 0)
          (begin (set! counter (+ counter 1)) x)
          0))))
```