Exercise 3.1

$\lfloor$`(value-of <<x>> `ρ`)`$\rfloor$ = `10`

$\lfloor$`(value-of <<3>> `ρ`)`$\rfloor$ = `3`

$\lfloor$`(value-of <<v>> `ρ`)`$\rfloor$ = `5`

$\lfloor$`(value-of <<i>> `ρ`)`$\rfloor$ = `1`

Exercise 3.2

A $val \in ExpVal$ must be that which is in $Int+Bool$. Then a $val \in ExpVal$ for which $\lceil\lfloor val \rfloor\rceil \neq val$ is where $val \in Bool$, such as $val = true$.

Exercise 3.3

We are able to describe the arithmetic operations in terms of subtraction. We cannot do so if we chose addition.

Exercise 3.4

Let ρ = `[x=`$\lceil$`33`$\rceil$`, y=`$\lceil$`22`$\rceil$`]`.

$$\dfrac{\texttt{(value-of-program <<if zero? (-(x,11)) then -(y,2) else -(y,4)>>)}}{\dfrac{\texttt{(value-of <<if zero? (-(x,11)) then -(y,2) else -(y,4)>> }\rho\texttt{)}}{\dfrac{\texttt{(value-of <<zero? (-(x, 11))>> }\rho\texttt{) = (bool-val \#f)}}{\dfrac{\texttt{(value-of <<-(y,4)>> }\rho\texttt{)}}{\lceil 18 \rceil}}}}$$

Exercise 3.5

$$\frac{\texttt{(value-of <<let x = 7 in let y = 2 in let y = let x = -(x,1) in -(x,y) in -(-(x,8),y)>> } \rho_0\texttt{)}}{skibidi - too - hard}$$

Exercise 3.6

```
;; the-grammar
(expression
 ("minus" "(" expression ")")
 minus-exp)

;; value-of
(minus-exp (exp1)
           (num-val (- 0 (expval->num (value-of exp1 env)))))
```

Exercise 3.7

```
;; the-grammar
(expression
 ("+" "(" expression "," expression ")")
 add-exp)

(expression
 ("*" "(" expression "," expression ")")
 mul-exp)

(expression
 ("/" "(" expression "," expression ")")
 div-exp)
```

```
;; value-of
(add-exp (exp1 exp2)
         (num-val (+ (expval->num (value-of exp1 env))
                     (expval->num (value-of exp2 env)))))
(mul-exp (exp1 exp2)
         (num-val (* (expval->num (value-of exp1 env))
                     (expval->num (value-of exp2 env)))))
(div-exp (exp1 exp2)
         (let ((divisor (expval->num (value-of exp2 env))))
           (if (zero? divisor)
               (eopl:error 'div-exp "division by zero")
               (num-val (/ (expval->num (value-of exp1 env))
                           divisor)))))
```

Exercise 3.8

```
;; the-grammar
(expression
 ("equal?" "(" expression "," expression ")")
 equal?-exp)

(expression
 ("greater?" "(" expression "," expression ")")
 greater?-exp)

(expression
 ("less?" "(" expression "," expression ")")
 less?-exp)

;; value-of
(equal?-exp (exp1 exp2)
            (bool-val (= (expval->num (value-of exp1 env))
                         (expval->num (value-of exp2 env)))))
(greater?-exp (exp1 exp2)
```

```
              (bool-val (> (expval->num (value-of exp1 env))
                           (expval->num (value-of exp2 env)))))
(less?-exp (exp1 exp2)
           (bool-val (< (expval->num (value-of exp1 env))
                        (expval->num (value-of exp2 env)))))
```

Exercise 3.9

```
;; the-grammar
(expression
 ("cons" "(" expression "," expression ")")
 cons-exp)

(expression
 ("car" "(" expression ")")
 car-exp)

(expression
 ("cdr" "(" expression ")")
 cdr-exp)

(expression
 ("null?" "(" expression ")")
 null?-exp)

(expression
 ("emptylist")
 emptylist-exp)

;; value-of
(cons-exp (exp1 exp2)
          (pair-val (value-of exp1 env) (value-of exp2 env)))
(car-exp (exp1)
         (car (expval->pair (value-of exp1 env))))
```

```
(cdr-exp (exp1)
         (cdr (expval->pair (value-of exp1 env))))
(null?-exp (exp1)
           (let ((val1 (value-of exp1 env)))
             (cases expval val1
               (emptylist-val () (bool-val #t))
               (else (bool-val #f)))))
(emptylist-exp () (emptylist-val))

;; expval
(pair-val
 (val1 expval?)
 (val2 expval?))
(emptylist-val)

(define expval->pair
  (lambda (v)
    (cases expval v
      (pair-val (val1 val2) (cons val1 val2))
      (else (expval-extractor-error 'pair v)))))
```

Exercise 3.10

$$Expression \quad ::= \quad \texttt{list}(\{Expression\}^{*(,)})$$

```
;; the-grammar
(expression
 ("list" "(" (separated-list expression ",") ")")
 list-exp)

;; value-of
(list-exp (exps)
          (value-of-list-exp exps env))
```

```
(define value-of-list-exp
  (lambda (exps env)
    (if (null? exps)
        (emptylist-val)
        (pair-val (value-of (car exps) env)
                  (value-of-list-exp (cdr exps) env)))))
```

Exercise 3.11

```
(define value-of
  (lambda (exp env)
    (cases expression exp
      (const-exp (num) (num-val num))
      (var-exp (var) (apply-env env var))
      (diff-exp (exp1 exp2) (value-of-diff-exp exp1 exp2 env))
      (zero?-exp (exp1) (value-of-zero?-exp exp1 env))
      (if-exp (exp1 exp2 exp3) (value-of-if-exp exp1 exp2 exp3 env
      (let-exp (var exp1 body) (value-of-let-exp var exp1 body env
      (minus-exp (exp1) (value-of-minus-exp exp1 env))
      (add-exp (exp1 exp2) (value-of-add-exp exp1 exp2 env))
      (mul-exp (exp1 exp2) (value-of-mul-exp exp1 exp2 env))
      (div-exp (exp1 exp2) (value-of-div-exp exp1 exp2 env))
      (equal?-exp (exp1 exp2) (value-of-equal?-exp exp1 exp2 env))
      (greater?-exp (exp1 exp2) (value-of-greater?-exp exp1 exp2 e
      (less?-exp (exp1 exp2) (value-of-less?-exp exp1 exp2 env))
      (cons-exp (exp1 exp2) (value-of-cons-exp exp1 exp2 env))
      (car-exp (exp1) (value-of-car-exp exp1 env))
      (cdr-exp (exp1) (value-of-cdr-exp exp1 env))
      (null?-exp (exp1) (value-of-null?-exp exp1 env))
      (emptylist-exp () (emptylist-val))
      (list-exp (exps) (value-of-list-exp exps env)))))

;; value-of-diff-exp : Exp * Exp * Env -> ExpVal
```

```
(define value-of-diff-exp
  (lambda (exp1 exp2 env)
    (let ((val1 (value-of exp1 env))
          (val2 (value-of exp2 env)))
      (let ((num1 (expval->num val1))
            (num2 (expval->num val2)))
        (num-val
         (- num1 num2))))))

;; value-of-zero?-exp : Exp * Env -> ExpVal
(define value-of-zero?-exp
  (lambda (exp1 env)
    (let ((val1 (value-of exp1 env)))
      (let ((num1 (expval->num val1)))
        (if (zero? num1)
            (bool-val #t)
            (bool-val #f))))))

;; value-of-if-exp : Exp * Exp * Exp * Env -> ExpVal
(define value-of-if-exp
  (lambda (exp1 exp2 exp3 env)
    (let ((val1 (value-of exp1 env)))
      (if (expval->bool val1)
          (value-of exp2 env)
          (value-of exp3 env)))))

;; value-of-let-exp : Identifier * Exp * Exp * Env -> ExpVal
(define value-of-let-exp
  (lambda (var exp1 body env)
    (let ((val1 (value-of exp1 env)))
      (value-of body
                (extend-env var val1 env)))))
```

```
;; value-of-minus-exp : Exp * Env -> ExpVal
(define value-of-minus-exp
  (lambda (exp1 env)
    (num-val (- 0 (expval->num (value-of exp1 env))))))

;; value-of-add-exp : Exp * Exp * Env -> ExpVal
(define value-of-add-exp
  (lambda (exp1 exp2 env)
    (num-val (+ (expval->num (value-of exp1 env))
                (expval->num (value-of exp2 env))))))

;; value-of-mul-exp : Exp * Exp * Env -> ExpVal
(define value-of-mul-exp
  (lambda (exp1 exp2 env)
    (num-val (* (expval->num (value-of exp1 env))
                (expval->num (value-of exp2 env))))))

  ;; value-of-div-exp : Exp * Exp * Env -> ExpVal
(define value-of-div-exp
  (lambda (exp1 exp2 env)
    (let ((divisor (expval->num (value-of exp2 env))))
      (if (zero? divisor)
          (eopl:error 'div-exp "division by zero")
          (num-val (/ (expval->num (value-of exp1 env))
                      divisor))))))

;; value-of-equal?-exp : Exp * Exp * Env -> ExpVal
(define value-of-equal?-exp
  (lambda (exp1 exp2 env)
    (bool-val (= (expval->num (value-of exp1 env))
                 (expval->num (value-of exp2 env))))))
```

```
;; value-of-greater?-exp : Exp * Exp * Env -> ExpVal
(define value-of-greater?-exp
  (lambda (exp1 exp2 env)
    (bool-val (> (expval->num (value-of exp1 env))
                 (expval->num (value-of exp2 env))))))

;; value-of-less?-exp : Exp * Exp * Env -> ExpVal
(define value-of-less?-exp
  (lambda (exp1 exp2 env)
    (bool-val (< (expval->num (value-of exp1 env))
                 (expval->num (value-of exp2 env))))))

;; value-of-cons-exp : Exp * Exp * Env -> ExpVal
(define value-of-cons-exp
  (lambda (exp1 exp2 env)
    (pair-val (value-of exp1 env) (value-of exp2 env))))

;; value-of-car-exp : Exp * Env -> ExpVal
(define value-of-car-exp
  (lambda (exp1 env)
    (car (expval->pair (value-of exp1 env)))))

;; value-of-cdr-exp : Exp * Env -> ExpVal
(define value-of-cdr-exp
  (lambda (exp1 env)
    (cdr (expval->pair (value-of exp1 env)))))

;; value-of-null?-exp : Exp * Env -> ExpVal
(define value-of-null?-exp
  (lambda (exp1 env)
    (let ((val1 (value-of exp1 env)))
```

```
      (cases expval val1
        (emptylist-val () (bool-val #t))
        (else (bool-val #f))))))

value-of-list-exp : Listof(Exp) * Env -> ExpVal
(define value-of-list-exp
  (lambda (exps env)
    (if (null? exps)
        (emptylist-val)
        (pair-val (value-of (car exps) env)
                  (value-of-list-exp (cdr exps) env)))))
```

Exercise 3.12

```
;; the-grammar
(expression
 ("cond" (arbno expression "==>" expression) "end")
 cond-exp)

;; value-of
(cond-exp (preds exps) (value-of-cond-exp preds exps env))

;; value-of-cond-exp : Listof(Exp) * Listof(Exp) * Env -> ExpVal
(define value-of-cond-exp
  (lambda (predicates consequents env)
    (if (null? predicates)
        (eopl:error 'cond-exp "no tests succeeded")
        (let ((pval (value-of (car predicates) env)))
          (if (expval->bool pval)
              (value-of (car consequents) env)
              (value-of-cond-exp (cdr predicates)
                                 (cdr consequents)
                                 env))))))
```

Exercise 3.13

```
;; value-of-zero?-exp : Exp * Env -> ExpVal
(define value-of-zero?-exp
  (lambda (exp1 env)
    (let ((val1 (value-of exp1 env)))
      (let ((num1 (expval->num val1)))
        (if (zero? num1)
            (num-val 1)
            (num-val 0))))))

;; value-of-if-exp : Exp * Exp * Exp * Env -> ExpVal
(define value-of-if-exp
  (lambda (exp1 exp2 exp3 env)
    (let ((val1 (value-of exp1 env)))
      (if (zero? (expval->num val1))
          (value-of exp3 env)
          (value-of exp2 env)))))

;; value-of-equal?-exp : Exp * Exp * Env -> ExpVal
(define value-of-equal?-exp
  (lambda (exp1 exp2 env)
    (if (= (expval->num (value-of exp1 env))
           (expval->num (value-of exp2 env)))
        (num-val 1)
        (num-val 0))))

;; value-of-greater?-exp : Exp * Exp * Env -> ExpVal
(define value-of-greater?-exp
  (lambda (exp1 exp2 env)
    (if (> (expval->num (value-of exp1 env))
           (expval->num (value-of exp2 env)))
        (num-val 1)
```

```
        (num-val 0))))

;; value-of-less?-exp : Exp * Exp * Env -> ExpVal
(define value-of-less?-exp
  (lambda (exp1 exp2 env)
    (if (< (expval->num (value-of exp1 env))
           (expval->num (value-of exp2 env)))
        (num-val 1)
        (num-val 0))))

;; value-of-null?-exp : Exp * Env -> ExpVal
(define value-of-null?-exp
  (lambda (exp1 env)
    (let ((val1 (value-of exp1 env)))
      (cases expval val1
        (emptylist-val () (num-val 1))
        (else (num-val 0))))))
```

Exercise 3.14

```
 ;; the-grammar
(bool-exp
 ("zero?" "(" expression ")")
 zero?-exp)

(expression
 ("if" bool-exp "then" expression "else" expression)
 if-exp)

(bool-exp
 ("equal?" "(" expression "," expression ")")
 equal?-exp)

(bool-exp
```

```
 ("greater?" "(" expression "," expression ")")
 greater?-exp)

(bool-exp
 ("less?" "(" expression "," expression ")")
 less?-exp)

(bool-exp
 ("null?" "(" expression ")")
 null?-exp)

(expression
 (bool-exp)
 a-bool-exp)

(define value-of
  (lambda (exp env)
    (cases expression exp
      (const-exp (num) (num-val num))
      (var-exp (var) (apply-env env var))
      (diff-exp (exp1 exp2) (value-of-diff-exp exp1 exp2 env))
      (if-exp (exp1 exp2 exp3) (value-of-if-exp exp1 exp2 exp3 env
      (let-exp (var exp1 body) (value-of-let-exp var exp1 body env
      (minus-exp (exp1) (value-of-minus-exp exp1 env))
      (add-exp (exp1 exp2) (value-of-add-exp exp1 exp2 env))
      (mul-exp (exp1 exp2) (value-of-mul-exp exp1 exp2 env))
      (div-exp (exp1 exp2) (value-of-div-exp exp1 exp2 env))
      (cons-exp (exp1 exp2) (value-of-cons-exp exp1 exp2 env))
      (car-exp (exp1) (value-of-car-exp exp1 env))
      (cdr-exp (exp1) (value-of-cdr-exp exp1 env))
      (emptylist-exp () (emptylist-val))
      (list-exp (exps) (value-of-list-exp exps env))
```

```
      (cond-exp (preds exps) (value-of-cond-exp preds exps env))
      (a-bool-exp (exp1) (value-of-bool-exp exp1 env)))))

;; value-of-bool-exp : Bool-exp * Env -> ExpVal
(define value-of-bool-exp
  (lambda (b-exp env)
    (cases bool-exp b-exp
      (zero?-exp (exp1) (value-of-zero?-exp exp1 env))
      (equal?-exp (exp1 exp2) (value-of-equal?-exp exp1 exp2 env))
      (greater?-exp (exp1 exp2) (value-of-greater?-exp exp1 exp2 e
      (less?-exp (exp1 exp2) (value-of-less?-exp exp1 exp2 env))
      (null?-exp (exp1) (value-of-null?-exp exp1 env)))))

;; value-of-if-exp : Exp * Exp * Exp * Env -> ExpVal
(define value-of-if-exp
  (lambda (exp1 exp2 exp3 env)
    (let ((val1 (value-of-bool-exp exp1 env)))
      (if (expval->bool val1)
          (value-of exp2 env)
          (value-of exp3 env)))))
```

Exercise 3.15

We could print the value of the expression in the current environment as an alternative solution.

```
;; the-grammar
(expression
 ("print" "(" expression ")")
 print-exp)

;; value-of
(print-exp (exp1) (value-of-print-exp exp1))

;; value-of-print-exp : Exp -> ExpVal
```

```
(define value-of-print-exp
  (lambda (exp1)
    (display exp1)
    (num-val 1)))
```

Exercise 3.16

```
;; the-grammar
(expression
 ("let" (arbno identifier "=" expression) "in" expression)
 let-exp)

;; value-of
(let-exp (vars exps body) (value-of-let-exp vars exps body env))

;; value-of-let-exp :
;; Listof(Identifier) * Listof(Exp) * Exp * Env -> ExpVal
(define value-of-let-exp
  (lambda (vars exps body env)
    (let ((vals (map (lambda (exp) (value-of exp env)) exps)))
      (value-of body
                (extend-env* vars vals env)))))

(define extend-env*
  (lambda (syms vals env)
    (if (null? syms)
        env
        (extend-env (car syms)
                    (car vals)
                    (extend-env* (cdr syms)
                                 (cdr vals)
                                 env)))))
```

Exercise 3.17

```
;; the-grammar
(expression
 ("let*" (arbno identifier "=" expression) "in" expression)
 let*-exp)

;; value-of
(let*-exp (vars exps body) (value-of-let*-exp vars exps body env))

;; value-of-let*-exp :
;; Listof(Identifier) * Listof(Exp) * Exp * Env -> ExpVal
(define value-of-let*-exp
  (lambda (vars exps body env)
    (if (null? vars)
        (value-of body env)
        (value-of-let*-exp
         (cdr vars)
         (cdr exps)
         body
         (extend-env
          (car vars)
          (value-of (car exps) env)
          env)))))
```

Exercise 3.18

```
;; the-grammar
(expression
 ("unpack" (arbno identifier) "=" expression "in" expression)
 unpack-exp)

;; value-of
(unpack-exp (vars exp1 body)
            (value-of-unpack-exp vars exp1 body env))
```

```
;; value-of-unpack-exp :
;; Listof(Identifier) * Exp * Exp * Env -> ExpVal
(define value-of-unpack-exp
  (lambda (vars exp1 body env)
    (let ((lst (expval->list (value-of exp1 env))))
      (if (= (length vars) (length lst))
          (value-of body (extend-env* vars lst env))
          (eopl:error 'unpack-exp
                      "number of vars not equal to number..."))))))

;; expval->list : ExpVal -> Listof(ExpVal)
(define expval->list
  (lambda (v)
    (cases expval v
      (emptylist-val () '())
      (pair-val (val1 val2) (cons val1 (expval->list val2)))
      (else (expval-extractor-error 'list v)))))
```