Exercise 3.30

The value in an `extend-env` is an expressed value, so the value of `saved-val` in `apply-env` is an expressed value.

Exercise 3.31

```
(letrec-exp
 (p-name identifier?)
 (b-vars (list-of identifier?))
 (p-body expression?)
 (letrec-body expression?))
(call-exp
 (rator expression?)
 (rands (list-of expression?)))

(define-datatype environment environment?
  (empty-env)
  (extend-env
   (var identifier?)
   (val expval?)
   (env environment?))
  (extend-env-rec
   (p-name identifier?)
   (b-vars (list-of identifier?))
   (body expression?)
   (env environment?)))

(letrec-exp (p-name b-vars p-body letrec-body)
            (value-of letrec-body
                      (extend-env-rec p-name b-vars p-body env)))
(call-exp (rator rands)
          (let ((proc (expval->proc (value-of rator env)))
                (args (map (lambda (rand) (value-of rand env))
                           rands)))
```

```
            (apply-procedure proc args)))

(define apply-procedure
  (lambda (proc1 vals)
    (cases proc proc1
      (procedure (vars body saved-env)
                 (value-of body
                           (extend-env* vars
                                        vals
                                        saved-env))))))
```

Exercise 3.32

```
(letrec-exp
 (p-names (list-of identifier?))
 (b-vars (list-of identifier?))
 (p-bodies (list-of expression?))
 (letrec-body expression?))

(letrec-exp (p-names b-vars p-bodies letrec-body)
            (value-of letrec-body
                      (extend-env-rec* p-names
                                       b-vars
                                       p-bodies
                                       env)))

(define extend-env-rec*
  (lambda (p-names b-vars p-bodies env)
    (if (null? p-names)
        env
        (extend-env-rec (car p-names)
                        (car b-vars)
                        (car p-bodies)
                        (extend-env-rec* (cdr p-names)
```

```
                                        (cdr b-vars)
                                        (cdr p-bodies)
                                        env)))))
```

Exercise 3.33

There is little difference when combining the solutions to Exercise 3.31 and Exercise 3.32.

```
(letrec-exp
 (p-names (list-of identifier?))
 (list-b-vars (list-of (list-of identifier?)))
 (p-bodies (list-of expression?))
 (letrec-body expression?))
```

Exercise 3.34

```
(define extend-env-rec
  (lambda (p-name b-var body env)
    (lambda (search-var)
      (if (eqv? search-var p-name)
          (proc-val (procedure b-var body env))
          (apply-env env search-var)))))

(define apply-env
  (lambda (env search-var)
    (env search-var)))
```

Exercise 3.35

```
(define-datatype environment environment?
  (empty-env)
  (extend-env
   (var identifier?)
   (val (or vector? expval?))
   (env environment?)))
```

```
(define apply-env
  (lambda (env search-var)
    (cases environment env
      (empty-env () (report-no-binding-found search-var))
      (extend-env (saved-var saved-val saved-env)
                  (if (vector? saved-val)
                      (vector-ref saved-val 0)
                      (if (eqv? saved-var search-var)
                          saved-val
                          (apply-env saved-env search-var)))))))
```

Exercise 3.36

No change from the solution to Exercise 3.32.

Exercise 3.37

```
let even(x) = if zero?(x) then 1 else (odd -(x,1))
in let odd(x) = if zero?(x) then 0 else (even -(x,1))
in (odd 3)
```