Exercise 3.18

```
(letproc-exp
 (name identifier?)
 (var identifier?)
 (proc-body expression?)
 (body expression?))

(letproc-exp (name var proc-body body)
             (value-of body (extend-env name
                                        (proc-val (procedure var p
                                        env)))
```

Exercise 3.19

```
proc (x) proc (y) -(x, -(0, y))
```

Exercise 3.20

```
(define-datatype proc proc?
  (procedure
   (vars (list-of identifier?))
   (body expression?)
   (saved-env environment?)))
(define apply-procedure
  (lambda (proc1 vals)
    (cases proc proc1
      (procedure (vars body saved-env)
                 (value-of body (extend-env* vars vals saved-env))

(proc-exp
 (formal-parameters (list-of identifier?))
 (body expression?))
(call-exp
 (rator expression?)
```

```
 (rands (list-of expression?)))

(proc-exp (params body)
          (proc-val (procedure params 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)))
```

Exercise 3.21

```
(define-datatype proc proc?
  (procedure
   (vars (list-of identifier?))
   (body expression?)
   (saved-env environment?)))
(define apply-procedure
  (lambda (proc1 vals)
    (cases proc proc1
      (procedure (vars body saved-env)
                 (value-of body (extend-env* vars vals saved-env))))))

(proc-exp
 (formal-parameters (list-of identifier?))
 (body expression?))
(call-exp
 (rator expression?)
 (rands (list-of expression?)))

(proc-exp (params body)
          (proc-val (procedure params 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)))
```