

### 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)))

```

### Exercise 3.23

```

(times 4 3)
((makemult makemult) 3)
((proc (x) if zero?(x) then 0 else -(((makemult makemult) -(x,1)),
-(((makemult makemult) 2), -4)
-((proc (x) if zero?(x) then 0 else -(((makemult makemult) -(x,1))
-(-(((makemult makemult) 1), -4), -4)
-(-((proc (x) if zero?(x) then 0 else -(((makemult makemult) -(x,1)
-(-(-(((makemult makemult) 0), -4), -4), -4)
-(-(-(proc (x) if zero?(x) then 0 else -(((makemult makemult) -(x,1)
-(-(-(-0, -4), -4), -4)
-(-(-4, -4), -4)
-(8, -4)
12

```

Let  $a$  be any nonnegative integer.

```

let times = proc (maker)
  proc (y)
    proc (x)
      if zero?(x)
      then 0
      else -((((maker maker) y) -(x,1)), -(0, y))
in let fact = proc (maker)
  proc (x)
    if zero?(x)
    then 1
    else (((times times) ((maker maker) -(x, 1))) x)
in ((fact fact) a)

```