Exercises

Exercise 1.15

```
(define duple
  (lambda (n x)
    (if (zero? n)
        '()
        (cons x (duple (- n 1) x)))))
```

Exercise 1.16

```
(define invert
  (lambda (lst)
    (if (null? lst)
        '()
        (cons (list (cadar lst) (caar lst))
              (invert (cdr lst))))))

(define down
  (lambda (lst)
    (if (null? lst)
        '()
        (cons (list (car lst))
              (down (cdr lst))))))

(define swap-in-s-exp
  (lambda (s1 s2 sexp)
    (if (symbol? sexp)
        (cond ((eqv? sexp s1) s2)
              ((eqv? sexp s2) s1)
              (else sexp))
        (swapper s1 s2 sexp))))

(define swapper
  (lambda (s1 s2 slist)
```

```
    (if (null? slist)
        '()
        (cons (swap-in-s-exp s1 s2 (car slist))
              (swapper s1 s2 (cdr slist))))))

(define report-list-too-short
  (lambda (n proc-name)
    (eopl:error proc-name
                "List too short by ~s elements.~%" (+ n 1))))

(define list-set
  (lambda (lst n x)
    (cond ((null? lst) (report-list-too-short n 'list-set))
          ((zero? n) (cons x (cdr lst)))
          (else (cons (car lst) (list-set (cdr lst) (- n 1) x))))))

(define count-occurrences
  (lambda (s slist)
    (if (null? slist)
        0
        (+ (count-occurrences-in-s-exp s (car slist))
           (count-occurrences s (cdr slist))))))

(define count-occurrences-in-s-exp
  (lambda (s sexp)
    (if (symbol? sexp)
        (if (eqv? sexp s) 1 0)
        (count-occurrences s sexp))))
```