`nth-element`

Exercise 1.6

If we check whether $n = 0$ before determining that *lst* is not empty, then we might take the car or cdr of an empty list in the recursive call.

Exercise 1.7

```
(define informative-report-list-too-short
  (lambda (lst n)
    (eopl:error 'nth-element
                "~s does not have ~s elements.~%"
                lst
                (+ n 1))))

(define nth-element
  (lambda (lst n)
    (define iter
      (lambda (l m)
        (if (null? l)
            (informative-report-list-too-short lst n)
            (if (zero? m)
                (car l)
                (iter (cdr l) (- m 1))))))
    (iter lst n)))
```

`remove-first`

Exercise 1.8

remove-first : $Sym \times Listof(Sym) \rightarrow Listof(Sym)$
usage: `(remove-first` *s los*`)` `returns a list with the elements of` *los* `arranged in the same order, except that all of the elements before and including the first occurrence of` *s* `are removed.`

Exercise 1.9

```
(define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eqv? (car los) s)
            (remove s (cdr los))
            (cons (car los) (remove s (cdr los)))))))
```

`occurs-free?`

Exercise 1.10

It can mean one and only one of two, or at least one of two.

Exercise 1.11

The recursion is guaranteed to halt on members of $S - list$, which in the last line of `subst-in-s-exp`, we know that `sexp` $\in S - list$.

Exercise 1.12

```
(define subst
  (lambda (new old slist)
    (if (null? slist)
        '()
        (cons (if (symbol? (car slist))
                  (if (eqv? (car slist) old)
                      new
                      (car slist))
                  (subst new old (car slist)))
              (subst new old (cdr slist))))))
```

Exercise 1.13

```
(define subst
```

```
(lambda (new old slist)
  (map (lambda (sexp)
         (subst-in-s-exp new old sexp))
       slist)))
```