Exercise 2.39.

Complete the following definitions of `reverse` (exercise 2.18) in terms of `fold-right` and `fold-left` from exercise 2.38:

```
(define (reverse sequence)
  (fold-right (lambda (x y) <??>) nil sequence))

(define (reverse sequence)
  (fold-left (lambda (x y) <??>) nil sequence))
```

Answer.

Recall that `fold-right` composes the first element of the sequence with the `fold-right` of all the rest elements. How these elements are composed into the result is specified by the parameter `op`.

On the other hand, we can reverse a sequence by swap the `car` of the sequence with the `reverse` of the `cdr` of the sequence. And this can be done by `append`ing the reverse of the `cdr` of the sequence by a list which merely contains the `car` of the sequence:[1]

```
(define (reverse sequence)
  (fold-right (lambda (x y)
                (append y (list x)))
              nil
              sequence))
```

Now consider the problem of reversing a sequence by means of `left-fold`ing. We begin by establishing an empty sequence, then extend it with the first element of the original sequence adding to the head of it. Also notice that head-insertion is in fact `cons`ing, this reveals another implementation of `reverse`:

```
(define (reverse sequence)
  (fold-left (lambda (x y)
               (append (list y) x))
             nil
             sequence))
```

*. Creative Commons  2013, Lawrence X. Amlord (颜世敏, aka 颜序).
Email address: informlarry@gmail.com

1. Someone might come up with an idea of `cons`ing the reversion of the `cdr` of the sequence onto the `car` of the sequence `(e1 e2 e3 ... en)`. Unfortunately, this strategy fails by producing the reversed sequence in a strange nested way:

```
(cons (cons (cons (... (cons nil en) ...)
                  e3)
            e2)
      e1)
```

Another hasty practice would be `list`ing the reversion of the `cdr` of the sequence with the `car` subsequent to it. Similarily, trying this plan also generates a nested sequence, though the elements are in a reversed order:

```
(list (list (list (... (list nil en) ...)
                  e3)
            e2)
      e1)
```