Exercise 1.33. You can obtain an even more general version of `accumulate` (exercise 1.32) by introducing the notion of a `filter` on the terms to be combined. That is, combine only those terms derived from values in the range that satisfy a specified condition. The resulting `filtered-accumulate` abstraction takes the same arguments as accumulate, together with an additional predicate of one argument that specifies the filter. Write `filtered-accumulate` as a procedure. Show how to express the following using `filtered-accumulate`:

a. the sum of the squares of the prime numbers in the interval a to b (assuming that you have a `prime?` predicate already written)

b. the product of all the positive integers less than n that are relatively prime to n (i.e., all positive integers $i < n$ such that $\mathrm{GCD}(i, n) = 1$).

Answer. Comparing with `accumulate`, the `filter-accumulate` procedure here with an additional predicate `filter` that check each index to decide whether to combine with that term or not.

```
(define (filter-accumulate filter combiner null-value term a next b)
  (cond ((> a b) null-value)
        ((filter a)
         (combiner (term a)
                   (filter-accumulate filter combiner null-value term (next a) next b)))
        (else (filter-accumulate filter combiner null-value term (next a) next b))))
```

a. Now, we can easily write down a procedure `prime-sum-sq` to compute the sum of the squares of the prime numbers in the interval a to b:

```
(define (prime-sum-sq a b)
  (filter-accumulate prime? + 0 square a inc b))
```

b. Similarly, the product of all the positive integers less than n that are relatively prime to n can be expressed by a procedure called `relat-prime-product`:

```
(define (relat-prime-product n)
  (filter-accumulate relat-prime? * 1 identity inc n))
```