Exercise 1.20. The process that a procedure generates is of course dependent on the rules used by the interpreter. As an example, consider the iterative `gcd` procedure given above. Suppose we were to interpret this procedure using normal-order evaluation, as discussed in section 1.1.5. (The normal-order evaluation rule for `if` is described in exercise 1.5.) Using the substitution method (for normal order), illustrate the process generated in evaluating `(gcd 206 40)` and indicate the `remainder` operations that are actually performed. How many `remainder` operations are actually performed in the normal-order evaluation of `(gcd 206 40)`? In the applicative-order evaluation?

Answer. The following process is generated by the interpreter while evaluating `(gcd 206 40)` in normal-order evaluation rule:

```
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b))))

(gcd 206 40)
;;(if (= 40 0) ...)
(gcd 40 (remainder 206 40))
;;(if (= (remainder 206 40) 0) ...)
;;(if (= 6 0) ...)
(gcd (remainder 206 40)
     (remainder 40 (remainder 206 40)))
;;(if (= (remainder 40 (remainder 206 40))
;;        0) ...)
;;(if (= 4 0) ...)
(gcd (remainder 40 (remainder 206 40))
     (remainder (remainder 206 40)
                (remainder 40 (remainder 206 40))))
;;(if (= (remainder (remainder 206 40)
;;                  (remainder 40 (remainder 206 40)))
;;        0) ...)
;;(if (= 2 0) ...)
(gcd (remainder (remainder 206 40)
                (remainder 40 (remainder 206 40)))
     (remainder (remainder 40 (remainder 206 40))
                (remainder (remainder 206 40)
                           (remainder 40 (remainder 206 40)))))
;;(if (= (remainder (remainder 40 (remainder 206 40))
;;                  (remainder (remainder 206 40)
;;                             (remainder 40 (remainder 206 40))))
;;        0) ...)
;;(if (= 0 0) ...)
(remainder (remainder 206 40)
           (remainder 40 (remainder 206 40)))
(remainder (remainder 206 40)
           (remainder 40 6))
(remainder (remainder 206 40) 4)
(remainder 6 4)
2
```

As is shown above, it takes the interpreter 14 times to perform the `remainder` operations while evaluating the predicates in those `if` conditionals. By comparison, the `remainder` operations are only performed for 4 times when it come to reduce the expression. Thus, the interpreter totally spends 18 times to perform the `remainder` operations in the normal-order evaluation of `(gcd 206 40)`.

The applicative-order evaluation, however, generates a pretty concise process in evaluating `(gcd 206 40)`:

```
(gcd 206 40)
;;(if (= 40 0) ...)
(gcd 40 (remainder 206 40))
(gcd 40 6)
;;(if (= 6 0) ...)
(gcd 6 (remainder 40 6))
(gcd 6 4)
```

*. Creative Commons 2013, Lawrence R. Amlord(颜世敏).

```
;;(if (= 4 0) ...)
(gcd 4 (remainder 6 4))
(gcd 4 2)
;;(if (= 2 0) ...)
(gcd 2 (remainder 4 2))
(gcd 2 0)
;;(if (= 0 0) ...)
2
```

Obviously, the `remainder` operations here are only performed for 4 times, they all take place in the reduction.