Exercise 1.7. The `good-enough?` test used in computing square roots will not be very effective for finding the square roots of very small numbers. Also, in real computers, arithmetic operations are almost always performed with limited precision. This makes our test inadequate for very large numbers. Explain these statements, with examples showing how the test fails for small and large numbers. An alternative strategy for implementing `good-enough?` is to watch how `guess` changes from one iteration to the next and to stop when the change is a very small fraction of the guess. Design a square-root procedure that uses this kind of end test. Does this work better for small and large numbers?

Answer. It is obvious that the absolute tolerance of 0.001 is extremly rough when it comes to compute the square root of a relatively small number. Let's take 0.00001 for example:

```
(sqrt 0.00001)
> 0.03135649010771716
```

Instead of putting out 0.0031622776601683794, which is legitimate. The original compound procedure `sqrt` gives a wrong answer with a tolerance over 900%.

On the other hand, the interpreter might fail to terminate its evaluation while dealing with a radicand whose magnitude becomes larger and larger except some special form like 10^{2n} and so on. Take a glance at the following example:

```
(sqrt 88888888888888888888888888888)
>
```

The odd thing happens here is because when the radicand becomes relatively giant, the interpreter might fail to represent the difference between `guess` and `x` within the tolerance of 0.001 with its limited precision. But the interpreter won't jump out the compound procedure `sqrt-iter` until the predicate `good-enough?` is sufficed. Thus this is contradictory and the interpreter drops into an endless sequence of recursive calls.

As is mentioned in the problem, in order to get an alternative strategy for implementing `good-enough?` with our limited precision, we should be able to represent the change with a limited proportion of `guess` itself. Well, let's go:

```
(define (good-enough? guess prev-guess x)
  (< (abs (- (improve guess x) guess))(* guess 0.001)))
```

Having the new `good-enough?` conditional defined above, we can easily rewrite the original square-root procedure as the following:

```
(define (sqrt-iter guess pre-guess x)
  (if (good-enough? guess prev-guess x)
      guess
      (sqrt-iter (improve guess x) guess x)))

(define (improve guess x)
  (average guess (/ x guess)))

(define (average x y)
  (/ (+ x y) 2))

(define (sqrt x)
  (sqrt-iter 1.0 0 x))
```

Now let's do some test on the new `sqrt` procedure as before:

```
(sqrt 0.00001)
> 0.0031622926477232706

(sqrt 88888888888888888888888888888)
> 298142561136219.9
```

By comparing the performance of the new `sqrt` procedure with the original version from the book. We are now sure that the new vertion of `sqrt` procedure works better for both small and large number.