Exercise 1.6. Alyssa P. Hacker doesn't see why if needs to be provided as a special form. ``Why can't I just define it as an ordinary procedure in terms of cond?'' she asks. Alyssa's friend Eva Lu Ator claims this can indeed be done, and she defines a new version of if:

```
(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
        (else else-clause)))
```

Delighted, Alyssa uses new-if to rewrite the square-root program:

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

What happens when Alyssa attempts to use this to compute square roots? Explain.

Solution

In this excercise I will use applicative order of substitution, but in this case the result would be very similar for normal order.

```
(sqrt 4)
```

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

```
(sqrt-iter 1.0 4)
```

```
(new-if (good-enough? guess x)
                guess
          (sqrt-iter (improve guess x)
                     x)))
```

```
(new-if (good-enough? 1.0 4)
                1.0
          (sqrt-iter (improve 1.0 4)
                     4)))
```

We can stop here, because the substitution will now try to evaluate/expand both branches of the new-if. This is because new-if is a function whereas standard if seems to be a special form. This is

explained directly in the book *“To evaluate an if expression, the interpreter starts by evaluating the* <predicate> *part of the expression. If the* <predicate>*evaluates to a true value, the interpreter then evaluates the* <consequent> *and returns its value. Otherwise it evaluates the* <alternative> *and returns its value”.* This means if will only evaluate one branch based on the predicate.

Thie means that in case of new-if, substitution needs to expand/evaluate all operands before being able to proceed. However the else branch is recursive and therefore the substitution will never end, even if the recursion termination condition is met.

To demonstrate this, we can use a bit simpler example. For example we can use very simple recursive function that was presented earlier in the book.

(define (p) (p))

And then apply in the following.

(if (= 1 1) 1 (p))

(new-if (= 1 1) 1 (p))

This will have exactly the same result as described above. In case of if, the code works fine. In case of new-if, it will never stop expanding. When used in REPL, this obviously resulted in infinite cycle and subsequent timeout.