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)))
```

Eva demonstrates the program for Alyssa:

```
(new-if (= 2 3) 0 5)
5

(new-if (= 1 1) 0 5)
0
```

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.

Answer. The interpreter turns out to never terminate without any result ouput when Alyssa attempts to compute square root in this way.

Note that we have been informed the application process for compound procedures in 1.1.5. That is, to apply a compound procedure to arguments, evaluate the body of the procedure with each formal parameter replaced by the corresponding argument.

After receiving the combination `(sqrt-iter guess x)`, the interpreter extracts it into:

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

Obvious, in order to get the compound procedure `new-if` applied, all the arguments in the latter expression above have to be evaluated. That is, the `(good-enough? guess x)`, `guess` and `(sqrt-iter (improve guess x) x)`. To evaluate `(sqrt-iter (improve guess x) x)`, the interpreter has to call `new-if` again. Continue in this process, we can see that the argument `(sqrt-iter (improve guess x) x)` would be evaluated recursively with `new-if` being called infinitely. Thus, a borderless recursion was formed and the interpreter would never terminate with nothing output.