Exercise 4.2.

Louis Reasoner plans to reorder the `cond` clauses in `eval` so that the clause for procedure applications appears before the clause for assignments. He argues that this will make the interpreter more efficient: Since programs usually contain more applications than assignments, definitions, and so on, his modified eval will usually check fewer clauses than the original eval before identifying the type of an expression.

a. What is wrong with Louis's plan? (Hint: What will Louis's evaluator do with the expression `(define x 3)`?)

b. Louis is upset that his plan didn't work. He is willing to go to any lengths to make his evaluator recognize procedure applications before it checks for most other kinds of expressions. Help him by changing the syntax of the evaluated language so that procedure applications start with call. For example, instead of `(factorial 3)` we will now have to write `(call factorial 3)` and instead of `(+ 1 2)` we will have to write `(call + 1 2)`.

Answer.

a. A procedure application is any compound expression that is not one of the above expression types. Also observe that all the special forms (quoted expressions, assignments, definitions etc.) can pass the test for procedure application. So it's appropriate to examine the procedure application after all the other cases have been checked.

In Louis's plan, where the clause for procedure application is placed at the very beginning of the `cond` clauses in `eval`, all the special forms will be regarded as procedure applications by the evaluator. Hence, whenever it encounters a special form, the evaluator will evaluate the procedure of that special form as well as all its arguments, even if one of its arguments is undefined. For example, when the expression `(define x 3)` is evaluated, the evaluator first evaluate `define` as well as `x` and `3` and applies the value of `define` to those of the latter ones. However, in terms of the evaluator, the variable `x` has not been defined. This traps the evaluator into bewildment.

b. As the evaluator is constructed using data abstraction in which we decouple the general rules of operation from the details of how expressions are represented. So, in order to make Louis's proposal fit into the new environment, all we need to do is to modify the underlying representation for procedure application:

```
(define (application? exp) (tagged-list? exp 'call))
(define (operator exp) (cadr exp))
(define (operands exp) (cddr exp))
```

*.
Email address: informlarry@gmail.com