Exercise 1F-2

One thing that I found very interesting in Hoare's paper was the discussion he had on the various different commenting styles that existed at the time. While this section came across as extremely dated, I found it incredibly relatable to see this discussion of how the seemingly minor decision of how the programming language handles commenting can have ripple effects in how people actually interface with the language. Personally, I'm a big fan of the central idea Knuth's literate programming style that documentation should be as much a part of the program as the code is. And, while the commenting section does come across as dated, I think it is still quite relevant, as there are still programming languages today that seem to discourage organization and documentation. One example that comes to mind is that, in Matlab, functions must occupy separate files. In my mind, all this ever seems to do is put a barrier in the way of me making my code properly organized.

One thing that I disagreed with among Hoare's points was the way he approached the concept of "simplicity" in programming language design. In his paper, Hoare held up many low-level hardware instruction sets as exceedingly simple. He claimed that this was good because it allowed the users to have every single aspect of the system memorized. While agree that this sort of simplicity would have this effect, I don't agree that this is the type of simplicity that should be desired in a programming language. Simplicity should be measured in terms of *distance to familiarity*, that is how many new concepts should a programmer have to understand in order to make use of the language. In these low-level instruction sets, the user must be familiar with all of the gory details of the underlying implementation in order to get anything done. In the design of Python there was a goal to have many ways to do any one thing, but only one *idomatic* way. While it's debatable whether modern Python actually achieves this goal, I think that it is a perfect basis for a goal of language simplicity.

Exercise 1F-3

There are a number of possible ways to extend IMP with a division operator. There are two problems that must be solved in order to define division in IMP. The first is that, in the best case, division is only a partial function because it is undefined when the denominator is zero. We will solve this problem by introducing the symbol $\perp$ to indicate undefined and altering our operational semantics so that arithmetic expressions may take values in $\mathbb{Z} \cup \{\perp\}$ rather than just $\mathbb{Z}$. The second problem is that division is not closed in the integers, that is, the quotient of two integers need not be an integer. While we could solve this problem by considering n/m to be undefined (by returning $\perp$) whenever m does not divide n, we find it more useful to preserve some information, so we will define integer division by truncation.

With the particulars decided, we now formally write out the necessary new rules of inference. Since arithmetic expressions can now be undefined, we first need to introduce rules to determine the behavior of all operations on undefined input. We define the following eight rules of inference in an abbreviated style: for $i \in \{1, 2\}$, and $\diamond \in \{+, -, *, /\}$ define the rules

$$\frac{\langle e_i, \sigma \rangle \Downarrow \perp}{\langle e_1 \diamond e_2, \sigma \rangle \Downarrow \perp}.$$

Similarly we define four more rules to work with boolean expressions: for $i \in \{1, 2\}$, and $\diamond \in \{\leq, =\}$ define the rules

$$\frac{\langle e_i, \sigma\rangle \Downarrow \bot}{\langle e_1 \diamond e_2, \sigma\rangle \Downarrow \texttt{false}}.$$

Finally we define the rules for division itself. To handle the case when the denominator is zero, we define the rule

$$\frac{\langle e_2, \sigma\rangle \Downarrow 0}{\langle e_1/e_2, \sigma\rangle \Downarrow \bot}.$$

And in any other case we define integer division by truncation:

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow n_2 \quad n_2 \neq 0}{\langle e_1/e_2, \sigma\rangle \Downarrow \lfloor n_1/n_2 \rfloor}.$$

Exercise 1F-4

In order to handle the `let` command we introduce the following rule

$$\frac{\langle e, \sigma\rangle \Downarrow n \quad \langle c, \sigma[x := n]\rangle \Downarrow \sigma'}{\langle \texttt{let } x = e \texttt{ in } c, \sigma\rangle \Downarrow \sigma'[x := \sigma(x)]}.$$

Note that with this rule, the command c is run with a state where x is bound to the result of e, and any bindings c makes to a location $y \in L, y \neq x$ are preserved, but the original binding of x is carried forward regardless of any assignments c makes.

Exercise 1F-5

We start by extending the set of redexes so that `let` commands may be reduced:

$$r ::= \ldots \mid \texttt{let } x = n \texttt{ in } c.$$

We define an additional reduction rule for this redex as follows:

$$\langle \texttt{let } x = n' \texttt{ in } c, \sigma\rangle \to \langle x := n'; c; x := n, \sigma\rangle,$$

where $n = \sigma(x)$ was the original value of x. Finally we extend the set of global contexts:

$$H ::= \ldots \mid \texttt{let } x = H \texttt{ in } c \mid \texttt{let } x = n \texttt{ in } H.$$

This set of contexts indicates that the expression in the let statement must be evaluated before the command can be executed.

1 HW1 (select all pages: your first page has your name and bookkeeping, and all others are anonymous))

- **0 pts** Correct