Exercise 1F-2. Language Design

In his paper, *Hints on Programming Language Design*, C.A.R. Hoare summarizes what he believes to be the most important design principles for modern programming languages. He also gives a passionate commentary on specific language features. Although many of his critiques are clearly coming from a 1960-70s perspective on the state of PL, I was nonetheless surprised by the relevancy of his discussion on program documentation. Whereas many 21st century programming languages have overcome Hoare's primary complaints (see simplicity, security, and fast translation) I cannot help but agree with Hoare that, even today, program readability and documentation is often treated as a fleeting afterthought. I found Hoare's assertion that "The view that documentation is something that is added to a program after it has been commissioned seems to be wrong in principle and counterproductive in practice" particularly insightful. In the time since Hoare's paper, there have been many novel attempts at encouraging good documentation during the primary development phase. These include programatic documentation generators like `doxygen`, source-level doc strings, and/or rigorous program style guides. Despite these developments, lack of quality documentaion is still a pervasive problem for even widely popular software projects.

Near the end of his paper, in his discussion on variables in programming languages, Hoare gives a scathing rebuke of reference variables. His justification for this is twofold. First, reference variables are solely dependent on and have no meaning outside of a particular program execution. Secondly, references to data have the potential to drastically harm performance should they need to be stored back or otherwise output. Admittedly, I do not know much about the particular intricacies and implementation of reference variables in Hoare's era, but to me these remarks seem to be ignore most if not all of the practical advantages that references offer. First and foremost of which is decreased memory footprint in stack-based languages with call-by-value parameter conventions (not to mention the time savings from the lack of additional copies.) Perhaps these benefits were less impactful until the development of C-like languages (which appeared shortly after Hoare's publication?) In any case, I have to disagree with Hoare on this point, but all-in-all I found *Hints on Programming Language Design* insightful and ripe for comparison to modern language design philosophy.

Exercise 1F-3. Simple Operational Semantics

The following two rules extend the Aexpr sub-language to deal with division. Note: "divided by" refers to mathematical division and $\lfloor x \rfloor$ refers to the greatest integer less than or equal to x (i.e. the floor function.) If e_2 evaluates to 0, the result is undefined and the program should terminate.

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \div e_2, \sigma\rangle \Downarrow \lfloor n_1 \text{ divided by } n_2 \rfloor} \text{ Where } n_2 \neq 0$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow 0}{\langle e_1 \div e_2, \sigma\rangle \Downarrow \text{undefined}}$$

Exercise 1F-4. Language Feature Design, Large Step

The following big step operational semantics judgement satisfies the requirements of the new `let` statement. Notice e is evaluated in the original state σ, c is evaluated in a state with the additional binding, and the original x binding is restored in whatever state c produces.

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

Exercise 1F-5. Language Feature Design, Small Step

The `let` command can be rewritten in terms of existing redexes in the small step IMP specification. Because of this and the fact that the sequence operator ensures proper ordering, no additional contexts are needed. The new redex and corresponding rule are written below:

$$r ::= \ldots \mid \text{let } x = e \text{ in } c$$
$$\langle \text{let } x = e \text{ in } c, \sigma \rangle \to \langle x := e; c; x := \sigma(x), \sigma \rangle$$

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

- **0 pts** Correct