0F-2. While many of Hoare's points still ring true to this day, the rapid advancement of computing technology over the past 50 years has led to some parts of this publication feeling rather outdated

Considering the agreeable aspects, some of Hoare's suggestions are so now ubiquitous as to seems trivial. Comment styles among all popular languages have essentially converged to the C-like comments Hoare recommends, the idea of lacking arithmetic expressions is almost laughable, and program features for structured control flow and blocks have become the norm. Other points, however, have yet to be fully realized, and his criticism are still quite reflective of modern languages in my experience. For example, the benefits of static types as Hoare lays out are quite to obvious to me - they eliminate large classes of errors and make code easier to reason about. While languages like Python and JavaScript are still widespread, I personally find that their lack of static types makes large-scale projects much more difficult to maintain. Others seem to agree, with things like type annotations and typed-variants of these languages (e,g, TypeScript) becoming ever more popular.

However, other aspects of Hoare's suggestions, particularly related to optimization and machine code, now seem quite irrelevant. For example, Hoare argues that optimizing compilers should be avoided in favor of more readable, reasonably efficient object code. Yet, outside of relatively rare use cases, the average programmer now has little care or knowledge for the machine code output of their compiled program, and the speed gain that machine-code level optimization brings is well-worth any such tradeoffs. Similarly, Hoare argues against separate debugging and production compilations, but with ever better compiler optimizations and practically limitless memory, there seems to be little downside to doing so. However, taking this publication for what it is - a set of recommendations from 1973 - perhaps Hoare deserves a pass on these small points.

0F-3. To support a division operator '/', we must add a new inference rule to the big step operational semantics which describes the evaluation of e_1/e_2 for $e_1, e_2 \in \text{Aexp}$. Formally, we add the inference rule

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1/e_2, \sigma\rangle \Downarrow n_3}, \text{ where } n_2 \neq 0 \text{ and } n_1 = n_2 \cdot n_3.$$

Note that this only defines a partial operation i,e, if $n_2 = 0$ or there is no such n_3, then the division operation is undefined.

0F-4. We add the rule

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

0F-5. To account for the `let` command, we extend the redexes and contexts to be

$$r ::= \cdots \mid \texttt{let } x = n \texttt{ in } c$$
$$H ::= \cdots \mid \texttt{let } x = H \texttt{ in } c$$

and add the reduction rule

$$\langle \texttt{let } x = n \texttt{ in } c,\ \sigma\rangle \ \rightarrow\ \langle x := n;\ c;\ x := \sigma(x),\ \sigma\rangle.$$

Peer Review ID: 62782009 — enter this when you fill out your peer evaluation via gradescope

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

- **0 pts** Correct