1F-2

I completely agree with Hoare's views on readability of programming languages. Since the primary activity in Software Engineering is reading software, the readability of the software is much more valuable than the ease of writing it. Of course, the language should be easy enough for the programmer to write in, but even more important is how readable the code is once it is written. I believe that a sufficiently experienced programmer should be able to understand the code even if they are completely unfamiliar with the language. This is very important these days because developers use a variety of tools and switch around often. The ability to quickly scan through the code and understand its meaning is paramount in such an environment.

I disagree with Hoare's views on "complete avoidance of any form of automatic type transfer…". Even though the automatic type casting rules (int to double, bool to int, int to bool, char to int, int to char, etc) can be a little confusing to learn at first, they greatly simplify code that works with many datatypes. Once learned, these rules appear fairly intuitive and allow the programmer to efficiently write correct code without having to look up the rules every time they want to cast data types. Automatic casting also significantly improves the readability of the code by avoiding verbose typecasting syntax.

1F-3

Let "//" represent the mathematical operation of integer division. Since division by zero is undefined mathematically, I arbitrarily define the result of division by zero to be zero. With these two assumptions, integer division can be represented by the following rules of inference:

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \langle e_2, \sigma\rangle \Downarrow n_2 \langle e_2 = 0, \sigma\rangle \Downarrow \text{false}}{\langle e_1 / e_2, \sigma\rangle \Downarrow n_1 // n_2}$$

$$\frac{\langle e_2 = 0, \sigma\rangle \Downarrow \text{true}}{\langle e_1 / e_2, \sigma\rangle \Downarrow 0}$$

1F-4

$$\frac{\langle x := e; c, \sigma\rangle \Downarrow \sigma' \; \langle x := \sigma(x), \sigma'\rangle \Downarrow \sigma''}{\langle \text{let } x = e \text{ in } c, \sigma\rangle \Downarrow \sigma''}$$

The first hypothesis evaluates the statements c with x set to e and the resulting state is stored as σ'. The second hypothesis sets the value of x back to the original value in σ and saves the state as σ''.

1F-5

I define a new context rule:

$H ::= \ldots \mid \text{let } x = H \text{ in } c$

This context tells us to evaluate the RHS of the assignment when a let command appears:

I define a new redex:

$r ::= \ldots \mid \text{let } x = n \text{ in } c, \text{ where } n \in Z$

with the following reduction rule:

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

This rule tells us to replace the let command with a sequence of three statement: assigning a new value n to x, evaluating c with the new value, restoring the old value $\sigma(x)$.

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

- **0 pts** Correct