Exercise 2

I agree with the author's opinion that replacing the principles of simplicity with that of modularity in language designs is questionable. Although a lot of programmers, myself included, work on languages that we partially understand on many occasions, it doesn't mean we can always get by that. The unfamiliar parts of languages always give me a sense of insecurity while working with them. Just like Hoare said, I got into serious trouble while working with LLVM, a C++ like language, which I had little experience with. I was stuck for a substantial amount of time to achieve a simple functionality that involved the understanding of some unfamiliar classes in LLVM. Upon the unfamiliarity, the complexity of the documentation of those classes and the implementation differences between different versions totally threw me off. Thus, I strongly agree with Hoare that simplicity is one of the most important characteristics that a successful language should have.

However, I hold doubts on the idea that "it seems especially necessary in the design of a new programming language, intended to attract programmers away from their current high level language, to pursue the goal of simplicity to an extreme. " Simplicity should be a principle for designing a new programming language, but it is wrong to phrase the importance of it as if people will change the tool they used as long as the new language is simple enough. Human nature is hardwired to resist change. No matter how simple a new language is, it would still require time to fully understand its usage. Moreover, simplicity is not the only factor that contributes to a programmer's degree of understanding towards a language. Time and practices would also help a complicated language to become handy. Therefore, I found this statement particularly misleading.

Exercise 3

In order to deal with the special case in the division rules, where an integer is divided by 0, we introduce NaN to represent the undefined result. (If necessary, let NaN be a quiet NaN, which doesn't raise any additional exceptions. Thus, the integer division won't lead the program into a new state.) Below are the two rules for integer division:

$$\frac{<e_2 = 0, \sigma> \Downarrow false \quad <e_1, \sigma> \Downarrow n_1 \quad <e_2, \sigma> \Downarrow n_2 \quad <n_2 * n_3 + n_4, \sigma> \Downarrow n_1}{<e_1 / e_2, \sigma> \Downarrow n_3}$$

$$\frac{<e_2 = 0, \sigma> \Downarrow true}{<e_1 / e_2, \sigma> \Downarrow NaN}$$

Peer Review ID: 63251233 — enter this when you fill out your peer evaluation via gradescope

Exercise 4

In the command '$let\ x\ =\ e\ in\ c$', c requires a temporary state where only the value of x is different from the original state. Thus, we update the original state with $x = e$. Then after the execution of c, we keep the new state and modify the value of x back to the original value.

$$\frac{< e,\ \sigma > \Downarrow n \quad < c,\ \sigma[\ x :=\ n] > \Downarrow \sigma'}{< let\ x\ =\ e\ in\ c,\ \sigma > \Downarrow \sigma'[\ x :=\ \sigma(x)]}$$

Exercise 5

To express the command '$let\ x\ =\ e\ in\ c$' in small steps, we need to evaluate expression e before creating the temporary state for the execution of c. Thus, we could extend the contexts as follows:

$H\ ::=\ ...\ |\ let\ x := H\ in\ c$

According to the context above, we could extend the redexes to:

$r\ ::=\ ...\ |\ let\ x\ :=\ n\ in\ c$

Finally, the local reduction rule would be:

$< let\ x\ :=\ n\ \ in\ \ c,\ \sigma > \ \rightarrow\ < skip;\ c;\ \ x :=\ \sigma(x),\ \sigma[\ x :=\ n] >$

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

- **0 pts** Correct