Exercise 4F-2. VCGen for Let [6 points]. The problem here is that the current VC rule for `let` treats it like a literal assignment `x := e`, and doesn't account for scoping.

$$\mathrm{VC}(c[x \mapsto x_{local}, B)[x_{local} \mapsto e]]$$

Here, we rename the fresh *local* x in `c` to a new variable (x_{local}) to avoid capturing the original scoping, and account for possible shadowing. The key idea is that the old x is restored after the let.

Peer Review ID: 316639216 — enter this when you fill out your peer evaluation via gradescope

Exercise 4F-3. VCGen Mistakes [6 points]. Given $\{A\}c\{B\}$ we desire that $A \implies \text{VC}(c, B) \implies \text{WP}(c, B)$. We say that our VC rules are *sound* if $\models \{\text{VC}(c, B)\}\ c\ \{B\}$. Demonstrate the unsoundness of the buggy let rule by giving the following six things:

1. a command c - let $x = 5$ in skip
2. a post-condition B - $(x = 5)$
3. a state σ such that - $\sigma(x) = 10$
4. $\sigma \models \text{VC}(c, B)$ - $[5/x]\text{VC}(\text{skip}, B)$, $\text{VC}(\text{skip}, B) = B$, $[5/x](x = 5)$, which is true and satisfied by every σ. So, $\sigma \models \text{VC}(c, B)$.
5. $\langle c, \sigma\rangle \Downarrow \sigma'$ - $\langle$let $x = 5$ in skip, σ[x := 10]$\rangle$. The skip does nothing, and restores x to 10 after execution.
6. $\sigma' \not\models B$. - but $\sigma' \neq$5, so $\sigma' \not\models B$ as $\sigma'(x) = 10$

Problem arises from treating let binding like an assignment (:=).

Peer Review ID: 316639216 — enter this when you fill out your peer evaluation via gradescope

Exercise 4F-4. Axiomatic Do-While [6 points]. Write a sound and complete Hoare rule for $\mathsf{do}\ c\ \mathsf{while}\ b$. This statement has the standard semantics (e.g., c is executed at least once, before b is tested).

$$\begin{array}{l}\textsc{Do-While}\\ \dfrac{\{A\}c\{B\} \qquad \{B \land b\}c\{B\}}{\{A\}\ \mathsf{do}\ c\ \mathsf{while}\ b\ \{B \land \neg b\}}\end{array}$$

The idea is that command c is first executed unconditionally, which establishes another assertion B, from the initial pre-condition A. Then, the loop behaves similar to a `while` loop as discussed in class, which established a loop invariant B to continue running, and finally exits when the loop guard (b) is false (with B still holding).