1 4F-1 Bookkeeping

- **0 pts** Correct

Exercise 4F-2. VCGen for Let The rule given for **Let x = e in c**:

$$\begin{array}{ll} \text{VC}(c_1; c_2, B) & = \text{VC}(c_1, \text{VC}(c_2, B)) \\ \text{VC}(x := e, B) & = [e/x]\ B \\ \text{VC}(\mathsf{let}\ x = e\ \mathsf{in}\ c, B) & = [e/x]\ \text{VC}(c, B) \end{array}$$

First lets identify what is wrong or buggy with the rule given. Take the final VC:
VC(let x = e in c, B) = [e/x] VC(c,B)

This works though only in some cases. Specifically the cases where x is used in some further c_n. The reason one might want to use a "let in" statement is to scope the variable. We want to have a local x := e, but after c we restore x := $\sigma(x)$. The VCGen given above does not restore the value of x.

Recall from our earlier proofs that the "Let in" rule can be viewed as a command tuple of the form: (temp := x; x := e; c; x := temp). The temp step is missing from the rule above, lets add it.

Let x = e in c →
k := x; x := e; c; x := k
VC(Let x = e in c) = VC(k := x; x := e; c; x := k)

As this is just a tuple of commands we know we can compute the VC as follows:
VC(k := x; x := e; c; x := k, B) =
[x/k] VC(x := e; c; x := k, B) =
[x/k] [e/x] VC(c; x := k, B) =
[x/k] [e/x] VC(c, VC(x := k, B)) =
[x/k] [e/x] VC(c, [k/x] B) =

For our final rule we have:

$$[x/k]\ [e/x]\ \text{VC}(c, [k/x]\ B)$$

We introduce a temporary variable, k, which holds the original state. We return x to k which was not done in the original rule.

Exercise 4F-3. VCGen Mistakes [6 points]. Now that we have an idea of what was wrong in the rule given we can consider the actions below in the original rule and point out where it errs. Recall that the rule IS correct if no other commands follow (a broken clock is right twice a day). Lets find those cases where the rule is wrong.

1. a command c and
   Let x = 85 in skip;
   Here we expect the answer to life, the universe, and everything. Instead we get 85. While 85 is the smallest number that can be expressed as a sum of two squares, with all squares greater than 1, in two ways, it's not correct.
2. a post-condition B and
   B = x = 85

2 4F-2 VCGen for Let

- **0 pts** Correct

Exercise 4F-2. VCGen for Let The rule given for **Let x = e in c**:

$$\begin{array}{ll} \text{VC}(c_1; c_2, B) & = \text{VC}(c_1, \text{VC}(c_2, B)) \\ \text{VC}(x := e, B) & = [e/x]\ B \\ \text{VC}(\mathsf{let}\ x = e\ \mathsf{in}\ c, B) & = [e/x]\ \text{VC}(c, B) \end{array}$$

First lets identify what is wrong or buggy with the rule given. Take the final VC:
VC(let x = e in c, B) = [e/x] VC(c,B)

This works though only in some cases. Specifically the cases where x is used in some further c_n. The reason one might want to use a "let in" statement is to scope the variable. We want to have a local x := e, but after c we restore x := $\sigma(x)$. The VCGen given above does not restore the value of x.

Recall from our earlier proofs that the "Let in" rule can be viewed as a command tuple of the form: (temp := x; x := e; c; x := temp). The temp step is missing from the rule above, lets add it.

Let x = e in c →
k := x; x := e; c; x := k
VC(Let x = e in c) = VC(k := x; x := e; c; x := k)

As this is just a tuple of commands we know we can compute the VC as follows:
VC(k := x; x := e; c; x := k, B) =
[x/k] VC(x := e; c; x := k, B) =
[x/k] [e/x] VC(c; x := k, B) =
[x/k] [e/x] VC(c, VC(x := k, B)) =
[x/k] [e/x] VC(c, [k/x] B) =

For our final rule we have:

$$[x/k]\ [e/x]\ \text{VC}(c, [k/x]\ B)$$

We introduce a temporary variable, k, which holds the original state. We return x to k which was not done in the original rule.

Exercise 4F-3. VCGen Mistakes [6 points]. Now that we have an idea of what was wrong in the rule given we can consider the actions below in the original rule and point out where it errs. Recall that the rule IS correct if no other commands follow (a broken clock is right twice a day). Lets find those cases where the rule is wrong.

1. a command c and
   Let x = 85 in skip;
   Here we expect the answer to life, the universe, and everything. Instead we get 85. While 85 is the smallest number that can be expressed as a sum of two squares, with all squares greater than 1, in two ways, it's not correct.
2. a post-condition B and
   B = x = 85

We have a post condition consistent with the wrong way to implement "let in"

3. a state σ such that
   $\sigma(x) = 42$
   This is the prior x which is not restored after our buggy let occurs

4. $\sigma \models \mathrm{VC}(c, B)$ and
   $\mathrm{VC}(c, B)$
   Because we use the buggy role and 42 is not restored $\mathrm{VC}(c, B)$ is $85 = 85$. note that c is Let x = 85 in skip; in our case

5. $\langle c, \sigma \rangle \Downarrow \sigma'$ but
   $\sigma'(x) = 42$ we would expect the result of the non-buggy rule but

6. $\sigma' \not\models B$.

   $\sigma' \not\models x = 85 \neq \sigma'(x) = 42$ alas, we have two conflicting results and we have shown that the rule given is wrong, we therefore use the new rule I describe above.

Exercise 4F-4. Axiomatic Do-While The rule for Do-While can be modeled as a combination of two rules we already know, **sequencing** and the basic **while**. Do-While simply executes c, then executes **while b do c**. This is identical to **c1;c2** where **c1 = c** and **c2 = while b do c**. This results in the following rule for Do-While:

$$\frac{\vdash \{\mathtt{A}\}\ \mathtt{c}\ \{\mathtt{B}\} \qquad \vdash \{\mathtt{B} \wedge \mathtt{b}\}\ \mathtt{c}\ \{\mathtt{B}\}}{\vdash \{\mathtt{A}\}\ \mathsf{do}\ c\ \mathsf{while}\ \ \mathtt{b}\ \{\mathtt{B} \wedge \neg \mathtt{b}\}}$$

We produce a state B, then we loop on B until we (hopefully) terminate with $\{B \wedge \neg b\}$. We always produce B, so even if b is false initially, we still terminate in $\{B \wedge \neg b\}$.

3 4F-3 VCGen Mistakes

- **0 pts** Correct

We have a post condition consistent with the wrong way to implement "let in"

3. a state σ such that
   $\sigma(x) = 42$
   This is the prior x which is not restored after our buggy let occurs

4. $\sigma \models \mathrm{VC}(c, B)$ and
   $\mathrm{VC}(c, B)$
   Because we use the buggy role and 42 is not restored $\mathrm{VC}(c, B)$ is $85 = 85$. note that c is Let x = 85 in skip; in our case

5. $\langle c, \sigma \rangle \Downarrow \sigma'$ but
   $\sigma'(x) = 42$ we would expect the result of the non-buggy rule but

6. $\sigma' \not\models B$.

   $\sigma' \not\models x = 85 \neq \sigma'(x) = 42$ alas, we have two conflicting results and we have shown that the rule given is wrong, we therefore use the new rule I describe above.

Exercise 4F-4. Axiomatic Do-While The rule for Do-While can be modeled as a combination of two rules we already know, **sequencing** and the basic **while**. Do-While simply executes c, then executes **while b do c**. This is identical to **c1;c2** where **c1 = c** and **c2 = while b do c**. This results in the following rule for Do-While:

$$\frac{\vdash \{\mathtt{A}\}\ \mathtt{c}\ \{\mathtt{B}\} \qquad \vdash \{\mathtt{B} \wedge \mathtt{b}\}\ \mathtt{c}\ \{\mathtt{B}\}}{\vdash \{\mathtt{A}\}\ \mathsf{do}\ c\ \mathsf{while}\ \ \mathtt{b}\ \{\mathtt{B} \wedge \neg \mathtt{b}\}}$$

We produce a state B, then we loop on B until we (hopefully) terminate with $\{B \wedge \neg b\}$. We always produce B, so even if b is false initially, we still terminate in $\{B \wedge \neg b\}$.

4 4F-4 Axiomatic Do-While

- **0 pts** Correct