1. Translate the following C program fragment into a statement in our programming language.
   a. $x = a * ++z$
   $z \coloneqq z + 1;\ x \coloneqq a * z$

   b. $x = a * z++$
   $x \coloneqq a * z; z \coloneqq z + 1$

   c. $\mathbf{while}\ (--x\ >= 0)\ \ z *= x;$
   $x \coloneqq x - 1; \mathbf{while}\ x \geq 0\ \mathbf{do}\ z \coloneqq z * x; x \coloneqq x - 1\ \mathbf{od}$

   d. $\mathbf{while}\ (x-- >= 0)\ \ z *= x;$
   $\mathbf{while}\ x \geq 0\ \mathbf{do}\ x \coloneqq x - 1; z \coloneqq z * x\ \mathbf{od}; x \coloneqq x - 1$

<u>Operational Semantics</u>

To analyze how a program works, we'll start with operational semantics: We'll model execution as a sequence of "configurations" — snapshots of the program and memory state — over time. The semantics rules describe how step–by–step execution of the program changes memory. When the program is complete, we have the final memory state of execution.

- A **configuration** $\langle S, \sigma \rangle$ is an ordered pair of a statement and state.
- The **operational semantics** of programs is given by a relation on configurations: $\langle S, \sigma \rangle \to \langle S_1, \sigma_1 \rangle$ means that executing S in state σ for one step yields $\langle S_1, \sigma_1 \rangle$. S_1 is the **continuation** of S. The "→" here doesn't mean implication, it simply means from one configuration to another.
  - For example, $\langle x \coloneqq x + 1; y \coloneqq x, \{x = 5\} \rangle \to \langle y \coloneqq x, \{x = 6\} \rangle$. In this example, $y \coloneqq x$ is the continuation of $x \coloneqq x + 1; y \coloneqq x$.

2. Evaluate configuration $\langle x \coloneqq x + 1; y \coloneqq x; z \coloneqq y + 3, \{x = 5\} \rangle$

$$\begin{aligned} \langle x \coloneqq x + 1; y \coloneqq x; z \coloneqq y + 3, \{x = 5\} \rangle &\to \langle y \coloneqq x; z \coloneqq y + 3, \{x = 6\} \rangle \\ &\to \langle z \coloneqq y + 3, \{x = 6,\ y = 6\} \rangle \\ &\to \langle E, \{x = 6,\ y = 6,\ z = 9\} \rangle \end{aligned}$$

- "E" represents an empty program, we use it to represent that we finish the execution.

- If there is a sequence of configurations $\langle S, \sigma \rangle \to \cdots \to \langle E, \tau \rangle$, then we say **execution of S starting in state σ converges to (or terminates in) state τ.** Based on which part we want to emphasize, we can simplify this sentence in different ways: "S converges", "σ terminates in state τ" or "σ terminates".

- The opposite of convergence is **divergence**, which means the program doesn't finish. For us, that means infinite loops or an infinitely long sequence of calculations. For example, $\langle \mathbf{while}\ x = 0\ \mathbf{do}\ x \coloneqq\ 0\ \mathbf{od}, \{x = 0\} \rangle$ and $\langle \mathbf{while}\ x \geq 0\ \mathbf{do}\ x \coloneqq x + 1\ \mathbf{od}, \{x = 0\} \rangle$ diverge.

(Operational Semantics Rules)

- For a no-op statement: $\langle \mathbf{skip}, \sigma \rangle \to \langle E, \sigma \rangle$

- For an assignment statement:
  - $\langle v \coloneqq e, \sigma\rangle \to \langle E, \sigma[v \mapsto \sigma(e)]\rangle$
  - $\langle\, b[e_1] \coloneqq e, \sigma\rangle \to \langle E, \sigma[b[\sigma(e_1)] \mapsto \sigma(e)]\rangle$

3. Evaluate the following configurations.
   a. $\langle x \coloneqq 2 * x * x + 5 * x + 6\,,\ \{x = 1, y = 2\}\rangle \to \langle E, \{x = 13, y = 2\}\rangle$

   b. With $\sigma(x) = 8$, $\langle b[x + 1] \coloneqq x * 5, \sigma\rangle \to \langle E, \sigma[b[9] \mapsto 40]\rangle$

- For a conditional statement:
  - $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \begin{cases} \langle S_1, \sigma\rangle, & \text{if } \sigma(B) = T \\ \langle S_2, \sigma\rangle, & \text{if } \sigma(B) = F \end{cases}$

  - $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ fi}, \sigma\rangle \to \begin{cases} \langle S_1, \sigma\rangle, & \text{if } \sigma(B) = T \\ \langle \textbf{skip}, \sigma\rangle, & \text{if } \sigma(B) = F \end{cases}$

4. Let $S \equiv \textbf{if } x > 0 \textbf{ then } y \coloneqq 0 \textbf{ fi}$ and $\sigma(x) = 5$, evaluate S in state σ.
   $\langle \textbf{if } x > 0 \textbf{ then } y \coloneqq 0 \textbf{ fi}, \sigma\rangle \to \langle y \coloneqq 0, \sigma\rangle \to \langle E, \sigma[y \mapsto 0]\rangle$

- For an iterative statement:
  - Let $W \equiv \textbf{while } B \textbf{ do } S \textbf{ od}$, then $\langle W, \sigma\rangle \to \begin{cases} \langle E, \sigma\rangle, & \text{if } \sigma(B) = F \\ \langle S; W, \sigma\rangle, & \text{if } \sigma(B) = T \end{cases}$

Here, the second case is the only operational semantic rule that produces a continuation that is larger than the starting statement. Thus, if the loop is infinite, we will get infinite configurations.
So far, all the rules we have seen are considered as axioms. For sequence statements, it is slightly more complicated to describe the rules, but it is also easy to understand.

- For a sequence statement:
  - If $\langle S_1, \sigma\rangle \to \langle U, \sigma_1\rangle$ then $\langle S_1; S_2, \sigma\rangle \to \langle U; S_2\,, \sigma_1\rangle$
  - If $\langle S_1, \sigma\rangle \to \langle E, \sigma_1\rangle$ then $\langle S_1; S_2, \sigma\rangle \to \langle S_2, \sigma_1\rangle$

5. Let $S \equiv s \coloneqq 0; k \coloneqq 0; W$, where $W \equiv \textbf{while } k < n \textbf{ do } S_1 \textbf{ od}$ and $S_1 \equiv s \coloneqq s + k + 1; k \coloneqq k + 1$, evaluate S in state σ with $\sigma(n) = 1$.

$$
\begin{aligned}
\langle S, \sigma\rangle &= \langle s \coloneqq 0; k \coloneqq 0; W, \sigma\rangle \\
&\to \langle k \coloneqq 0; W, \sigma[s \mapsto 0]\rangle \\
&\to \langle W, \sigma[s \mapsto 0][k \mapsto 0]\rangle \\
&= \langle \textbf{while } k < n \textbf{ do } S_1 \textbf{ od}\,, \sigma[s \mapsto 0][k \mapsto 0]\rangle \\
&\to \langle S_1; W, \sigma[s \mapsto 0][k \mapsto 0]\rangle && \text{// since } \sigma[s \mapsto 0][k \mapsto 0](k < n) = T \\
&= \langle s \coloneqq s + k + 1; k \coloneqq k + 1;\ W, \sigma[s \mapsto 0][k \mapsto 0]\rangle \\
&\to \langle k \coloneqq k + 1; W, \sigma[s \mapsto 1][k \mapsto 0]\rangle \\
&\to \langle W, \sigma[s \mapsto 1][k \mapsto 1]\rangle \\
&= \langle \textbf{while } k < n \textbf{ do } S_1 \textbf{ od}\,, \sigma[s \mapsto 1][k \mapsto 1]\rangle \\
&\to \langle E, \sigma[s \mapsto 1][k \mapsto 1]\rangle && \text{// since } \sigma[s \mapsto 1][k \mapsto 1](k < n) = F
\end{aligned}
$$

<u>Denotational Semantics</u>

We have seen the step-by-step operational semantics for our program now, but we sometimes do not need that many details. We want to learn some ways to simplify this process so that we can focus on the most important configurations.

- If we have a sequence of configurations $\langle S_0, \sigma_0 \rangle \to \langle S_1, \sigma_1 \rangle \to \langle S_2, \sigma_2 \rangle \to \cdots \to \langle S_n, \sigma_n \rangle$, then we say $\langle S_0, \sigma_0 \rangle$ **evaluates to $\langle S_n, \sigma_n \rangle$ in n steps**, which is denoted as $\langle S_0, \sigma_0 \rangle \to^n \langle S_n, \sigma_n \rangle$.
    - "n" can even be 0 $\langle S_0, \sigma_0 \rangle \to^0 \langle S_0, \sigma_0 \rangle$.

Under many circumstances, we do not really care the value of n in the above definition, we can use "$\to^*$" to represent any number of steps.

- Program S starting in σ terminates in (converges to) τ if $\langle S, \sigma \rangle \to^* \langle E, \tau \rangle$.

6. Let $S \equiv s \coloneqq 0; k \coloneqq 0; W$, where $W \equiv \textbf{while } k < n \textbf{ do } S_1 \textbf{ od}$ and $S_1 \equiv s \coloneqq s + k + 1;\ k \coloneqq k + 1$, evaluate S in state σ with $\sigma(n) = 4$.

$$\begin{aligned} \langle S, \sigma \rangle &= \langle s \coloneqq 0; k \coloneqq 0; W, \sigma \rangle \\ &\to^2 \langle W, \sigma[s \mapsto 0][k \mapsto 0] \rangle \\ &\to^* \langle W, \sigma[s \mapsto 1][k \mapsto 1] \rangle \\ &\to^* \langle W, \sigma[s \mapsto 3][k \mapsto 2] \rangle \\ &\to^* \langle W, \sigma[s \mapsto 6][k \mapsto 3] \rangle \\ &\to^* \langle W, \sigma[s \mapsto 10][k \mapsto 4] \rangle \\ &\to \langle E, \sigma[s \mapsto 10][k \mapsto 4] \rangle \end{aligned}$$

- If $\langle S, \sigma \rangle \to^* \langle E, \tau \rangle$ (or in other words, if program S in state σ terminate in τ) then we say τ is the **denotational semantics of S in σ**, and we write $M(S, \sigma) = \{\tau\}$.
    - We can consider the denotational semantics of a program as the final evaluation of a program. It is like we skip all details in the operational semantics.
    - We can simplify the simplify $\{\tau\}$ to τ, but technically it is not correct, because a configuration might end up with a set of different states (we will see that next week). To sum up, we can simplify $\{\tau\}$ to τ, unless $\tau = \emptyset$ or τ is more than one states.

(Denotational Semantics Rules)
Denotational semantics rules are basically just operational semantics rules written in a different way.

- $M(\textbf{skip}, \sigma) = \{\sigma\}$
- $M(v \coloneqq e, \sigma) = \{\sigma[v \mapsto \sigma(e)]\}$
- $M(b[e_1] \coloneqq e, \sigma) \to \{\sigma[b[\sigma(e_1)] \mapsto \sigma(e)]\}$
- $M(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma) = \begin{cases} M(S_1, \sigma), & \text{if } \sigma(B) = T \\ M(S_2, \sigma), & \text{if } \sigma(B) = F \end{cases}$
- Let $W \equiv \textbf{while } B \textbf{ do } S \textbf{ od}$, then $M(W, \sigma) = \begin{cases} \{\sigma\}, & \text{if } \sigma(B) = F \\ M(S; W, \sigma) = M\big(W, M(S, \sigma)\big), & \text{if } \sigma(B) = T \end{cases}$
- $M(S_1; S_2, \sigma) = M\big(S_2, M(S_1, \sigma)\big)$
    - In the last two rules, we have the nested M. We assume the inner M returns some state τ (instead of $\{\tau\}$).

7. Let $W \equiv \textbf{while } x < n \textbf{ do } S \textbf{ od}$, where the loop body $S \equiv x \coloneqq x + 1; y \coloneqq y + y$. Calculate $M(W, \sigma)$ where $\sigma = \{x = 0, n = 3, y = 1\}$.

$$\begin{aligned} M(W, \sigma) &= M(W, \{x = 0, n = 3, y = 1\}) \\ &= M\big(W, M(S, \{x = 0, n = 3, y = 1\})\big) \\ &= M(W, \{x = 1, n = 3, y = 2\}) \\ &= M(W, \{x = 2, n = 3, y = 4\}) \end{aligned}$$

$= M(W, \{x = 3, n = 3, y = 8\})$
$= \{\{x = 3, n = 3, y = 8\}\}$

Divergence

We learned that if S in σ converges to τ, we have $M(S, \sigma) = \tau$; but what if S diverges?

- We define pseudo-state "⊥" (reads "bottom") to represent a program cannot terminate successfully.
  - ⊥ is not a real state in memory.
  - There can be multiple reasons a program cannot terminate successfully, such as the program diverges, or the program meets some runtime error and halts.

- Denotationally, we write $M(S, \sigma) = \{\bot_d\}$ to represent S diverges in σ. Operationally, $\langle S, \sigma\rangle \to^* \langle E, \bot_d\rangle$ means that S starting in σ diverges.

- Here, we present two situations that we can say S diverges in σ. However, if program S and state σ doesn't satisfy the following situation, it is still possible that S diverges in σ; in general, "Whether arbitrary S converges in arbitrary σ?" is an undecidable problem.

  - In the sequence of configurations $\langle S_0, \sigma_0\rangle \to \langle S_1, \sigma_1\rangle \to \langle S_2, \sigma_2\rangle \to \cdots$, if $\exists i. \exists j. i \neq j \land S_i = S_j \land \sigma_i = \sigma_j$, then S_0 starting in σ_0 diverges.

8. Evaluate $W \equiv \textbf{while } T \textbf{ do skip od}$ in σ.
   Since $\langle W, \sigma\rangle \to \langle \textbf{skip}; W, \sigma\rangle \to \langle W, \sigma\rangle$, thus $M(W, \sigma) = \{\bot_d\}$.

   - While evaluating an iterative statement $W \equiv \textbf{while } B \textbf{ do } S \textbf{ od}$, and get a sequence $\langle W, \sigma_0\rangle \to^* \langle W, \sigma_1\rangle \to^* \langle W, \sigma_2\rangle \to \cdots$, if $\neg\exists i. \sigma_i(B) = F$, then W in σ_0 diverges.

9. Calculate $M(W, \sigma_0)$ where $W \equiv \textbf{while } x \neq n \textbf{ do } x := x - 1 \textbf{ od}$ and $\sigma_0 = \{x = -1, n = 0\}$.
   $M(W, \sigma_0)$ $= M(W, \sigma_0 = \{x = -1, n = 0\})$
   $= M(W, \sigma_1 = \sigma_0[x \mapsto -2] = \{x = -2, n = 0\})$
   $= M(W, \sigma_2 = \sigma_1[x \mapsto -3] = \{x = -3, n = 0\})$
   $= \cdots$

   - We start with $\sigma_0(x) < \sigma_0(n)$; after each iteration of W, the value bind to x can only be updated to a smaller number and the value of n never changes. Thus $\forall i. \sigma_i(x) < \sigma_i(n)$, and $M(W, \sigma_0) = \{\bot_d\}$.