Problem 1: Condition Strength

Given three implication chains of the form

$$\text{Condition}_1 \longleftarrow \text{Condition}_2 \longleftarrow \text{Condition}_3 \longleftarrow \cdots \longleftarrow \text{Condition}_n,$$

which is to be read as "Condition$_i$ is implied by Condition$_{i+1}$." If the rightmost condition holds, then all conditions to its left must also hold. In a correct chain every implication

$$(\text{Right}) \implies (\text{Left})$$

must be true.

Chain (1)

$\{x \text{ is a Monday}\} \longleftarrow \{x \text{ is a day in January 2025}\} \longleftarrow \{x \text{ is a Monday in January}\} \longleftarrow \{x = \text{Monda}$

Label the conditions from right to left as:

- $D : \{x = \text{Monday, January 27, 2025}\}$
- $C : \{x \text{ is a Monday in January}\}$
- $B : \{x \text{ is a day in January 2025}\}$
- $A : \{x \text{ is a Monday}\}$

We check each implication:

1. $D \implies C$**:** If x is exactly Monday, January 27, 2025, then x is certainly a Monday in January. **Conclusion: True.**
2. $C \implies B$**:** C states "x is a Monday in January" but does not specify the year. For example, if

   $$x = \text{Monday, January 6, 2015},$$

   then C holds but x is not in January 2025. **Conclusion: False.**
3. $B \implies A$**:** B states "x is a day in January 2025", yet many days in January 2025 are not Mondays (e.g., January 1, 2025 is a Wednesday). **Conclusion: False.**

Since at least two implications fail, **Chain (1) is false**.

Chain (2)

$\{u = 10k \wedge v = y+7 \wedge u+v = 7k\} \longleftarrow \{10x+y \text{ is divisible by } 7\} \longleftarrow \{x-2y \text{ is divisible by } 7 \vee (x-2y) =$

Label these from right to left:

- $E : \{10x + y = 609\}$
- $D : \{x = 60 \wedge y = 9\}$
- $C : \{x - 2y \text{ is divisible by } 7 \vee (x - 2y) = 0\}$
- $B : \{10x + y \text{ is divisible by } 7\}$
- $A : \{u = 10k \wedge v = y + 7 \wedge u + v = 7k\}$

Check each implication:

1. $E \implies D$**:** E states $10x + y = 609$. In general, this linear equation has many solutions. For instance, if $x = 59$ then

$$y = 609 - 10 \cdot 59 = 19,$$

so $10x + y = 609$ holds but $(x, y) \neq (60, 9)$. **Conclusion: False.**

2. $D \implies C$**:** With $x = 60$ and $y = 9$, compute:

$$x - 2y = 60 - 18 = 42,$$

and since 42 is divisible by 7, the disjunction holds. **Conclusion: True.**

3. $C \implies B$**:** If $x - 2y$ is divisible by 7 (or equals 0), then in particular $x - 2y = 7n$ for some integer n (noting that $0 = 7 \cdot 0$). Write $x = 7n + 2y$ and substitute into $10x + y$:

$$10(7n + 2y) + y = 70n + 21y = 7(10n + 3y),$$

which is divisible by 7. **Conclusion: True.**

4. $B \implies A$**:** B states that $10x + y$ is divisible by 7. However, A involves different variables u, v, k with the conditions

$$u = 10k, \quad v = y + 7, \quad u + v = 7k.$$

There is no necessary connection between $10x + y$ being divisible by 7 and these equations holding. For example, take

$$x = 1, \quad y = -3 \quad (\text{so } 10x + y = 7).$$

Then even if one sets $k = 1$, choosing u arbitrarily (say $u = 0$) will cause $u = 10k$ to fail. **Conclusion: False.**

Since the implications $E \implies D$ and $B \implies A$ fail, **Chain (2) is false**.

Chain (3)

$$\{\text{true}\} \longleftarrow \{|x| = x\} \longleftarrow \{x > 0\} \longleftarrow \{10 < x < 50 \wedge x < 0\} \longleftarrow \{\text{false}\}$$

Label the conditions from right to left:

- E : {false}
- D : $\{10 < x < 50 \wedge x < 0\}$
- C : $\{x > 0\}$
- B : $\{|x| = x\}$
- A : {true}

Check the implications:

1. $E \implies D$**:** Since E is false, the implication is vacuously **true**.
2. $D \implies C$**:** The condition D requires both $10 < x < 50$ (i.e., $x > 10$) and $x < 0$, which is impossible. An unsatisfiable antecedent implies any statement; hence, this is vacuously **true**.
3. $C \implies B$**:** If $x > 0$, then by definition $|x| = x$. **Conclusion: True.**
4. $B \implies A$**:** Since A is the constant **true** statement, any statement implies **true**. **Conclusion: True.**

Thus, every implication in Chain (3) holds (some vacuously), and **Chain (3) is true**.

Part (b): Ordering Conditions by Strength

In all parts, an implication

$$A \leftarrow B$$

means that if condition B holds then condition A also holds (i.e. $B \implies A$). In set language, $\text{Sol}(B) \subseteq \text{Sol}(A)$. "Weakest" conditions (largest solution sets) appear on the left, "strongest" (smallest solution sets) on the right. If two conditions are equivalent, they are placed together.

B(1)

Given Conditions:

1. $\{\, 6 \leq k < 6 \wedge k = 6\}$
2. {false}
3. $\{\, k = 6n\}$
4. $\{\, 12 \leq k \leq 12\}$

Step 1. Simplification:

- In (1), $6 \le k < 6$ is always false. Adding $k = 6$ does not salvage it. Hence (1) has no solutions.
- (2) is explicitly false (no solutions).
- (3) means k is a multiple of 6. (Infinitely many solutions: $\ldots, -6, 0, 6, 12, \ldots$)
- (4) means $12 \le k \le 12$, so $k = 12$ (exactly one solution).

Step 2. Implications and Counterexamples:

- Comparing (4) and (3): If $k = 12$ then certainly $k = 6n$ (take $n = 2$); that is, (4) $\implies$ (3). However, (3) $\not\Longrightarrow$ (4): for example, if $k = 6$ then $k = 6n$ holds (with $n = 1$) but $k \neq 12$.
- Any condition implying false is impossible unless the condition itself has no solutions. Thus (1) and (2) are equivalent and are stronger than any nonempty condition.

Step 3. Final Ordering:

$$\{\, k = 6n\} \quad \leftarrow \quad \{\, 12 \le k \le 12\} \quad \leftarrow \quad \{\, 6 \le k < 6 \wedge k = 6\} \equiv \{\text{false}\}.$$

B(2)

Given Conditions:

1. $\{\, x = 3k + 3 \wedge x \text{ is even}\}$
2. $\{\, x \text{ is divisible by } 6\}$
3. $\{\, y = (\text{sum of digits of } x) \wedge y\%3 = 0 \wedge x = 2k\}$
4. $\{\, x = x + 1 \wedge y = 12\}$

Step 1. Analysis of (1), (2), (3):

- In (1) write $x = 3(k + 1)$. For x to be even, $k + 1$ must be even; let $k + 1 = 2m$. Then $x = 6m$. Thus (1) forces x to be divisible by 6.
- (2) directly states that x is divisible by 6.
- In (3), $x = 2k$ (so x is even) and y is the sum of the digits of x with $y\%3 = 0$, which is the divisibility test for 3. Hence (3) also says x is divisible by 6.

Thus (1), (2), and (3) are logically equivalent.

Step 2. Analysis of (4):

- (4) contains $x = x + 1$, which is impossible. Hence (4) is always false.

Step 3. Final Ordering:

$\{x = 3k+3 \wedge x \text{ is even}\} \equiv \{x \text{ is divisible by } 6\} \equiv \{y = (\text{sum of digits of } x),\ y\%3 = 0,\ x = 2k\} \quad \leftarrow \quad \{x =$

B(3)

Assume `result` is a `double`. Given Conditions:

1. $\{\ \texttt{result} = \sqrt{x}\}$
2. $\{\ |\texttt{result}^2 - x| \geq 0.0\}$
3. $\{\ |\texttt{result}^2 - x| < -10^{-10}\}$
4. $\{\ |\texttt{result}^2 - x| < 0.001\}$

Step 1. Simplification:

- (2): For any real number, $|\texttt{result}^2 - x| \geq 0$ is always true. So (2) is the weakest (no restriction).
- (3): Since -10^{-10} is negative, no nonnegative number (as an absolute value) can be less than it. Thus (3) is always false.
- (1): If $\texttt{result} = \sqrt{x}$ (the principal square root) then $\texttt{result}^2 = x$.
- (4): Demands $|\texttt{result}^2 - x| < 0.001$, a bound that is weaker than equality.

Step 2. Implications and Counterexamples:

- (1) $\implies$ (4): If $\texttt{result} = \sqrt{x}$, then $\texttt{result}^2 - x = 0 < 0.001$.
- (4) does not imply (1) because $|\texttt{result}^2 - x| < 0.001$ can hold even if `result` is not exactly $\sqrt{x}$. For example, with rounding errors one might have $\texttt{result}^2$ very close to x without equality.
- (3) is always false so it implies every condition (vacuously) but no non-false condition implies (3).

Step 3. Final Ordering:

$\{\ |\texttt{result}^2 - x| \geq 0.0\} \quad \leftarrow \quad \{\ |\texttt{result}^2 - x| < 0.001\} \quad \leftarrow \quad \{\texttt{result} = \sqrt{x}\} \quad \leftarrow \quad \{\ |\texttt{result}^2 - x| < -1$

Summary of Reasoning

1. A condition C is *stronger* than D if $C \implies D$ (i.e., $\text{Sol}(C) \subseteq \text{Sol}(D)$).
2. The always-false conditions (empty solution sets) are the strongest; the always-true conditions (no restriction) are the weakest.
3. When conditions are equivalent, they are grouped together.

Problem 2 (4 pts., 1 pt. each): Hoare Triples

(1)

$$\{x \geq -\tfrac{1}{2}\}$$

```
y = 2 * x;
```

$$\{y \geq 0 \vee y = 1\}$$

- Since x is an integer, the precondition $x \geq -\frac{1}{2}$ forces $x \geq 0$ (because there is no integer between $-\frac{1}{2}$ and 0).
- The assignment $y = 2 * x$ then gives $y = 2x$.
- With $x \geq 0$, we have $y \geq 0$. Hence the disjunction $y \geq 0 \vee y = 1$ holds (indeed, $y \geq 0$ is always true).

Answer (1): Valid.

(2)

$$\{\sqrt{x-1} \leq k\}$$

```
x = x - 1;
```

$$\{k < 0\}$$

- The expression $\sqrt{x-1}$ is defined only when $x - 1 \geq 0$, so we must have $x \geq 1$. Also, $\sqrt{x-1} \geq 0$ for all $x \geq 1$.

- The precondition $\sqrt{x-1} \le k$ therefore forces $k \ge 0$.
- However, the postcondition is $k < 0$, which contradicts the fact that k must be nonnegative.
- **Thus, the triple is invalid.**

Now, to obtain the strongest postcondition after, the assignment changes x as follows:

- Let the old value of x be x_{old}. The precondition is $\sqrt{x_{\text{old}} - 1} \le k$.
- After executing $x = x - 1$, the new value is $x_{\text{new}} = x_{\text{old}} - 1$.
- Rewriting the precondition in terms of x_{new}:

$$\sqrt{(x_{\text{new}} + 1) - 1} = \sqrt{x_{\text{new}}} \le k.$$

Modified (Strongest) Postcondition:

$$\{\sqrt{x} \le k\}$$

Answer (2): Invalid; the strongest postcondition is $\{\sqrt{x} \le k\}$.

(3)

$$\{i + j \neq 0 \wedge i \cdot j = 0\}$$

```
i = j + 1;
k = i * j;
j = i - 1;
```

$$\{i = 0 \vee i = j \vee k = i \cdot j\}$$

- The precondition $i \cdot j = 0$ together with $i + j \neq 0$ tells us that exactly one of i or j is 0.
- **Step 1:** The assignment `i = j + 1` sets

$$i_{\text{new}} = j_{\text{old}} + 1.$$

- **Step 2:** Then `k = i * j` computes

$$k = (j_{\text{old}} + 1) \cdot j_{\text{old}}.$$

- **Step 3:** Finally, `j = i - 1` updates j to

$$j_{\text{new}} = (j_{\text{old}} + 1) - 1 = j_{\text{old}}.$$

- The final state is:

$$i = j_{\text{old}} + 1, \quad j = j_{\text{old}}, \quad k = (j_{\text{old}} + 1) \cdot j_{\text{old}}.$$

- Notice that in the final state we always have

$$k = i \cdot j.$$

- Since the postcondition is a disjunction that includes $k = i \cdot j$, the postcondition is satisfied regardless of the values of i and j.

Answer (3): Valid.

(4)

$$\{\text{false}\}$$

```
if (m < n)
    x = n;
else
    x = m;
```

$$\{x = \min(n, m)\}$$

- The precondition is false, which means no initial state satisfies the precondition.
- In Hoare logic, if the precondition is false, the Hoare triple is *vacuously valid* (since there are no executions that start in a state satisfying false; formally, from false anything follows).

Answer (4): Valid.

Problem 3 (4 pts., 1 pt. each): General Hoare Triples

1. $\{E\}$ code $\{B\}$ is *possibly invalid.*

Counterexample:

- **Let:**
  - B : “$x = 1$”
  - E : “$x > 0$”

  $B \to E$ holds since if $x = 1$ then $x > 0$, but E does not imply B.

- **Define the code as:**

```
if (x == 1) {
    x = 1;
} else {
    // do nothing (i.e., x remains unchanged)
}
```

Analysis:

- If the initial state satisfies B (i.e., $x = 1$), then the "if" branch executes and x remains 1. Hence, the postcondition E (and even B) holds.
- However, if the initial state satisfies E but not B (say, $x = 2$, so $2 > 0$ holds but $x \neq 1$), the "else" branch is taken and x remains 2. In this case, the postcondition B (i.e., $x = 1$) fails.

Thus, the Hoare triple $\{E\}$ code $\{B\}$ is **possibly invalid**.

2. $\{B\}$ code $\{E\}$ is valid.

This triple is given as a fact in the problem statement.

3. $\{C\}$ code $\{D\}$ is *possibly invalid.*

Counterexample:

- **Let:**
  - B : “$x = 1$”
  - C : “$x = 1 \lor x = 2$” (so that $B \to C$)

- D : "$x = 1$"
- E : "$x = 1$"

(Also, since $C \to D$ is assumed given, but here it is not a logical necessity for the code's behavior.)

- **Define the code as:**

```
if (x == 1) {
    x = 1;
} else {
    x = 2;
}
```

Analysis:

- For any state satisfying B (i.e., $x = 1$), the "if" branch executes and x remains 1, so E holds (and D holds since $x = 1$).
- However, for a state where $x = 2$ (which satisfies C but not B), the "else" branch executes, and x remains 2. This violates D (since $2 \neq 1$).

Thus, the triple $\{C\}$ code $\{D\}$ is **possibly invalid**.

4. $\{E\}$ code $\{E\}$ is *possibly invalid.*

Counterexample:

- **Let:**
  - B : "$x = 1$"
  - E : "$x > 0$"

  ($B \to E$ holds, but the converse does not.)

- **Define the code as:**

```
if (x == 1) {
    x = 1;
} else {
    x = -1;
}
```

Analysis:

- When $x = 1$ (i.e., B holds), the "if" branch executes and x remains 1. So the postcondition E (since $1 > 0$) holds.
- When $x \neq 1$ but still E holds (for example, $x = 2$ so $2 > 0$), the "else" branch executes and sets $x = -1$. This final state does not satisfy E (since $-1 \not> 0$).

Thus, the Hoare triple $\{E\}$ code $\{E\}$ is **possibly invalid**.

Explanation

1. **For $\{E\}$ code $\{B\}$:**
   We are given that $\{B\}$ code $\{E\}$ holds. This guarantees that if the initial state satisfies the stronger condition B, then after executing the code the state will satisfy the weaker condition E. However, if we start from a state that satisfies E but not B, there is no guarantee that the code "recovers" B because E is a strictly weaker condition than B. The provided counterexample shows that starting with E (e.g., $x = 2$) might lead to a post-state that still does not satisfy B.

2. **For $\{B\}$ code $\{E\}$:**
   This triple is given to be true. The chain $A \to B \to C \to D \to E \to F$ along with the fact $\{B\}$ code $\{E\}$ ensures that whenever B holds initially, the final state will satisfy E.

3. **For $\{C\}$ code $\{D\}$:**
   Although we know $C \to D$ holds as a logical implication, the correctness of a Hoare triple depends on the behavior of the code. The triple $\{B\}$ code $\{E\}$ tells us nothing about what happens when the precondition is C (which is weaker than B); the code may behave arbitrarily when B is not met. The counterexample shows a case where starting from a state satisfying C but not B (i.e., $x = 2$) leads to a final state that violates D.

4. **For $\{E\}$ code $\{E\}$:**
   Similar reasoning applies. Although the postcondition is the same as the precondition, the code's guarantee was only provided for states that satisfy B. For states that satisfy E but not B, the code might not preserve E. The counterexample demonstrates that if the initial state is such that $x > 0$ but $x \neq 1$, the code might change the state so that E no longer holds.

Final Answers

1. $\{E\}$ code $\{B\}$: **Possibly Invalid**
2. $\{B\}$ code $\{E\}$: **Valid**
3. $\{C\}$ code $\{D\}$: **Possibly Invalid**
4. $\{E\}$ code $\{E\}$: **Possibly Invalid**

Problem 4 (8 pts., 1.5 pts. for each condition): Forward reasoning

Note on notation: In assignments where a variable is updated we can "remember" its former value by writing, "y_{old}" to denote the value of y before the assignment. PLEASE IGNORE strange formatting or red arrows as I'm not sure why my latex is not compiling properly.

(1) Precondition: $\{ z = 0 \}$

```
// Precondition: { z = 0 }
x = 10;
// Postcondition after x = 10: { x = 10 /\ z = 0 }
y = y - x;
// Here the "old" value of y is denoted by y_old.
// After assignment: x remains 10, z still 0, and new y = y_old - 10.
{ x = 10 /\ z = 0 /\ y = y_old - 10 }
z = x - y;
// Now, using x = 10 and y = y_old - 10, we get:
// z = 10 - (y_old - 10) = 20 - y_old.
{ x = 10 /\ y = y_old - 10 /\ z = 20 - y_old }
y = 0;
// This assignment resets y to 0 (x and z are unchanged).
{ x = 10 /\ y = 0 /\ z = 20 - y_old }
z = 2 * k;
// Finally, z is overwritten by 2*k; x and y remain.
{ x = 10 /\ y = 0 /\ z = 2 * k }
```

(2) Precondition: $\{ |x| > 4 \}$

```
y = x;
// Let the original value of x be denoted by x_0.
// After y = x, we have: x = x_0, y = x_0, and |x_0| > 4.
{ x = x_0 /\ y = x_0 /\ |x_0| > 4 }
x = -x * y;
// With x = x_0 and y = x_0, the new x becomes -x_0 * x_0 = -x_0^2.
{ x = -x_0^2 /\ y = x_0 /\ |x_0| > 4 }
x = x + y;
```

```
// Now, new x = (-x_0^2) + x_0 = x_0 - x_0^2, and y is still x_0.
{ x = x_0 - x_0^2 /\ y = x_0 /\ |x_0| > 4 }
```

(3) Precondition: $\{ x \cdot y = 0 \}$

```
if (x > 0 || y > 0) {
    // Under the precondition x*y = 0 and branch condition (x > 0 || y > 0)
    // only one of x or y can be nonzero.
    // Thus the possible states are either (x > 0 /\ y = 0) or (x = 0 /\ y > 0).
    { (x > 0 /\ y = 0) \/ (x = 0 /\ y > 0) }
    y = y * x;
    // In the first case: if x > 0 and y = 0 then y becomes 0 * x = 0.
    // In the second case: if x = 0 and y > 0 then y becomes y * 0 = 0.
    // Hence, in either case the new value of y is 0; x is unchanged.
    { y = 0 }
} else {
    // In the else branch the condition not (x > 0 or y > 0) holds, i.e. x <= 0 and y <= 0.
    // With the precondition x*y = 0, the possibilities are:
    // either (x < 0 /\ y = 0) or (x = 0 /\ y < 0) or (x = 0 /\ y = 0).
    // We express this as:
    { (x < 0 /\ y = 0) \/ (x = 0 /\ y < 0) }
    x = x + y;
    // Now consider the two cases:
    // -- If (x < 0 /\ y = 0): then x becomes x+0 = x (< 0) and y remains 0.
    // -- If (x = 0 /\ y < 0): then x becomes 0 + y = y (< 0) and y remains unchanged.
    // In the (x = 0, y = 0) case the result is (0, 0).
    // Thus, after the assignment we have:
    { (x < 0 /\ y = 0) \/ (x = y /\ y < 0) }
}
// After the if--else, we combine the outcomes from both branches.
// In the if branch, we had (with x originally >= 0): x remains either > 0 or = 0 and y = 0.
// In the else branch, the result is either (x < 0 with y = 0) or (x = y with y < 0).
// Combining these, if y = 0 (whether x is positive or negative) that fact holds,
// and the only "extra" information in the else branch is when x = y (with y < 0).
{ (y = 0) \/ (x = y /\ y < 0) }
```

Problem 5 (12 pts., 0.5 pts. each condition): Backward reasoning

```
{ y ≤(10-(x+k))/2 }
x = x + k;
{ wp("z = 2 * y + x;", z ≤10) } = { 2y + x ≤10 } =
{ 2y ≤10 - x } = { y ≤(10 - x) / 2 }
z = 2 * y + x;
{ z ≤10 }
```

Given the computation of

$$wp(z = 2 * y + x,\ z \leq 10) = \{\, y \leq \frac{10 - x}{2} \,\}$$

it applies *after* the assignment

$$x = x + k.$$

Thus, when chaining the assignments, we must substitute the new value of x (i.e. $x + k$) into the condition. In other words, replace every occurrence of x with $x + k$ in

$$y \leq \frac{10 - x}{2},$$

obtaining

$$y \leq \frac{10 - (x + k)}{2}.$$

Thus, the missing precondition at the very top should be

$$\{\, y \leq \frac{10 - (x + k)}{2} \,\}.$$

2)

```
{ x(1-x) > 0 }
y = x;
{ y(1-x) > 0 }
x = -x * y;
{ x+y > 0 }
x = x + y;
{ x > 0 }
```

Step 1: For the final assignment:

$$wp(\texttt{x = x+y;}, x > 0) = \{ x + y > 0 \}.$$

Step 2: For the second assignment:

$$wp(\texttt{x = -x*y;}, \{x + y > 0\}) = \{ (-x * y) + y > 0 \}.$$

Factor y:

$$= \{ y(1 - x) > 0 \}.$$

Step 3: For the first assignment:

$$wp(\texttt{y = x;}, \{ y(1 - x) > 0 \}) = \{ x(1 - x) > 0 \}.$$

Thus,

- **Before:** $\{ x(1 - x) > 0 \}$
- **After** `y = x;`**:** $\{ y(1 - x) > 0 \}$
- **After** `x = -x*y;`**:** $\{ x + y > 0 \}$
- **After** `x = x+y;`**:** $\{ x > 0 \}$

In "wp() notation":

$$\begin{aligned} wp(\texttt{x = x+y;}, x > 0) &= \{ x + y > 0 \}, \\ wp(\texttt{x = -x*y;}, \{x + y > 0\}) &= \{ y(1 - x) > 0 \}, \\ wp(\texttt{y = x;}, \{y(1 - x) > 0\}) &= \{ x(1 - x) > 0 \}. \end{aligned}$$

3

```
{ y<10 ∧x≤-10 }
if (y >= 10) {
    { (y/10)<0 ∧(y%10)≤-10 }
    x = y / 10; // integer division
    { x<0 ∧(y%10)≤-10 }
    y = y % 10;
    { x<0 ∧y≤-10 }
} else {
    { x≤-10 }
    y = x;
    { x<0 ∧y≤-10 }
}
{ x < 0 ∧y ≤-10 }
```

Let the final postcondition be

$$Q \equiv \{ x < 0 \wedge y \leq -10 \}.$$

Then–branch (when $y \geq 10$):

Step 1. For `y = y % 10;`:

$$wp(\texttt{y = y \% 10;}, Q) = \{\, x < 0 \wedge (y\%10) \leq -10 \,\}.$$

Step 2. For `x = y/10;`:

$$wp(\texttt{x = y/10;}, \{x < 0 \wedge (y\%10) \leq -10\}) = \{\, (y/10) < 0 \wedge (y\%10) \leq -10 \,\}.$$

Thus, the then–branch requires:

$$\{\, (y/10) < 0 \wedge (y\%10) \leq -10 \,\}$$

when entered under the guard $y \geq 10$.

Else–branch (when $y < 10$):
For `y = x;`:

$$wp(\texttt{y = x;}, Q) = \{\, x < 0 \wedge x \leq -10 \,\},$$

which is equivalent to $\{\, x \leq -10 \,\}$ (since $x \leq -10$ implies $x < 0$).

Overall if–statement:
The wp for the entire if–statement is:

$$\Big[(y \geq 10 \wedge \{(y/10) < 0 \wedge (y\%10) \leq -10\}) \vee (y < 10 \wedge \{x \leq -10\})\Big].$$

Since the then–branch is unsatisfiable for $y \geq 10$, the overall wp simplifies to:

$$\{\, y < 10 \wedge x \leq -10 \,\}.$$

therefore:

- **Before if:** $\{\, y < 10 \wedge x \leq -10 \,\}$
- **Then–branch:**
  - Before `x = y/10;`: $\{\, (y/10) < 0 \wedge (y\%10) \leq -10 \,\}$
  - After `x = y/10;`: $\{\, x < 0 \wedge (y\%10) \leq -10 \,\}$
  - After `y = y % 10;`: $\{\, x < 0 \wedge y \leq -10 \,\}$
- **Else–branch:**
  - Before `y = x;`: $\{\, x \leq -10 \,\}$
  - After `y = x;`: $\{\, x < 0 \wedge y \leq -10 \,\}$
- **Final postcondition:** $\{\, x < 0 \wedge y \leq -10 \,\}$

4

```
{ (|x|<=5 ∧-7≤x≤0 ∧z<0) ∨(x>5 ∧-5/2≤z<0) }
if (Math.abs(x) <= 5) {
    { -7≤x≤0 ∧z<0 }
    x = x + 2;
    { -5≤x≤2 ∧z<0 }
} else {
    { x>5 ∧-5/2≤z<0 }
    if (x <= -5) {
        { -5≤x≤2 ∧x<-6 }
        z = x + 6;
        { -5≤x≤2 ∧z<0 }
    } else {
        { -5/2≤z<0 }
        x = 2 * z;
        { -5≤x≤2 ∧z<0 }
    }
}
{ -5≤x≤2 ∧z<0 }
```

Let the final postcondition be

$$Q \equiv \{\,-5 \le x \le 2 \land z < 0\,\}.$$

First branch (when $|x| \le 5$):

For `x = x+2;`:

$$wp(\texttt{x = x+2;}, Q) = \{\,-5 \le x+2 \le 2 \land z < 0\,\}.$$

Simplify by subtracting 2:

$$\iff \{\,-7 \le x \le 0 \land z < 0\,\}.$$

Thus, in the then–branch the required precondition before the assignment is:

$$\{\,-7 \le x \le 0 \land z < 0\,\},$$

and the branch is executed under the guard $|x| \le 5$.

Else branch (when $|x| > 5$):

There is a nested if–statement.

Inner if (when $x \le -5$):

For `z = x+6;`:

$$wp(\texttt{z = x+6;}, Q) = \{\, -5 \le x \le 2 \wedge (x+6) < 0 \,\} = \{\, -5 \le x \le 2 \wedge x < -6 \,\}.$$

Since x cannot simultaneously satisfy $-5 \le x$ and $x < -6$, this branch is unsatisfiable.

Inner else (when $x > -5$):

For `x = 2*z;`:

$$wp(\texttt{x = 2*z;}, Q) = \{\, -5 \le 2z \le 2 \wedge z < 0 \,\}.$$

Dividing the inequality by 2:

$$\iff \{\, -\frac{5}{2} \le z \le 1 \wedge z < 0 \,\}.$$

Since $z < 0$ is already required, we simplify to:

$$\{\, -\frac{5}{2} \le z < 0 \,\}.$$

The inner else is executed under the guard $x > -5$.

Furthermore, the outer else is entered when $|x| > 5$. Since $|x| > 5$ and $x > -5$ force $x > 5$, the overall else–branch requires:

$$\{\, x > 5 \wedge -\frac{5}{2} \le z < 0 \,\}.$$

Overall if–statement:

Thus, the wp for the entire if–statement is:

$$\Big[(|x| \le 5 \wedge -7 \le x \le 0 \wedge z < 0) \vee (x > 5 \wedge -\frac{5}{2} \le z < 0) \Big].$$

A full annotated answer is:

- **Before the if:**

$$\{\, (|x| \le 5 \wedge -7 \le x \le 0 \wedge z < 0) \vee (x > 5 \wedge -\frac{5}{2} \le z < 0) \,\}$$

- **Then–branch (when $|x| \le 5$):**
  - Before `x = x+2;`: $\{\, -7 \le x \le 0 \wedge z < 0 \,\}$
  - After `x = x+2;`: $\{\, -5 \le x \le 2 \wedge z < 0 \,\}$

- **Else–branch (when $|x| > 5$):**
  - Before inner if: $\{\, x > 5 \wedge -\frac{5}{2} \le z < 0 \,\}$

  - Inner if ($x \leq -5$) is unsatisfiable:
    - Before `z = x+6;`: $\{ -5 \leq x \leq 2 \wedge x < -6 \}$
    - After `z = x+6;`: $\{ -5 \leq x \leq 2 \wedge z < 0 \}$
  - Inner else (when $x > -5$):
    - Before `x = 2*z;`: $\{ -\frac{5}{2} \leq z < 0 \}$
    - After `x = 2*z;`: $\{ -5 \leq x \leq 2 \wedge z < 0 \}$

- **Final postcondition:** $\{ -5 \leq x \leq 2 \wedge z < 0 \}$

5

```
{ (x<10 ∧((z<0 ∧((Math.max(z,x)≠0 ∧y≥0) ∨(x+y≥0)))
   ∨(z≥0 ∧(((x+10)≠0 ∧y≥0) ∨(x+y≥0)))) )
   ∨(x≥10) }
if (x < 10) {
    { (z<0 ∧((Math.max(z,x)≠0 ∧y≥0) ∨(x+y≥0)))
       ∨(z≥0 ∧(((x+10)≠0 ∧y≥0) ∨(x+y≥0))) }
    z = (z < 0) ? Math.max(z,x) : x+10;
    { (z≠0 ∧y≥0) ∨(x+y≥0) }
    x = x+y;
    { (z≠0 ∧y≥0) ∨x≥0 }
}
{ (z≠0 ∧y≥0) ∨x≥0 }
```

The final postcondition be

$$Q \equiv \{ (z \neq 0 \wedge y \geq 0) \vee x \geq 0 \}.$$

work in two parts.

When the if–branch is executed (i.e. $x < 10$):

Step 1: For `x = x+y;`:

$$wp(\texttt{x = x+y;}, Q) = \{ (z \neq 0 \wedge y \geq 0) \vee (x + y \geq 0) \}.$$

Denote this intermediate postcondition by Q_1.

Step 2: For the assignment with the ternary operator:
The assignment is:

`z = (z<0)? Math.max(z,x): x+10;`.

We treat this as a conditional assignment with two cases:

- **Case A (when $z < 0$):** After the assignment, z becomes Math.max(z, x). The resulting condition is:

$$Q_{1A} = \{\, (\text{Math.max}(z, x) \neq 0 \wedge y \geq 0) \vee (x + y \geq 0) \,\}.$$

- **Case B (when $z \geq 0$):** After the assignment, z becomes $x + 10$. The resulting condition is:

$$Q_{1B} = \{\, ((x + 10) \neq 0 \wedge y \geq 0) \vee (x + y \geq 0) \,\}.$$

Thus, the wp for the ternary assignment is:

$$wp\big(\texttt{z = (z<0)? Math.max(z,x): x+10;},\, Q_1\big) = \{\, (z < 0 \wedge Q_{1A}) \vee (z \geq 0 \wedge Q_{1B}) \,\}.$$

When the if–statement is not executed (i.e. $x \geq 10$):

No assignments occur so the state remains unchanged. However, if $x \geq 10$ then $x \geq 0$ holds automatically. Thus the postcondition Q is satisfied and the wp is *true*.

Overall:

The wp for the entire if–statement is:

$$\Big[(x < 10 \wedge wp(\texttt{z = (z<0)? Math.max(z,x): x+10;},\, wp(\texttt{x=x+y;},\, Q))) \vee (x \geq 10)\Big],$$

where Q is the final postcondition. Since Q is automatically true when $x \geq 10$,

$$\{\, (x < 10 \wedge \Big[(z < 0 \wedge Q_{1A}) \vee (z \geq 0 \wedge Q_{1B})\Big]) \vee (x \geq 10) \,\}.$$

A fully annotated answer is then:

- **Before the if:**

  $$\{\, (x < 10 \wedge wp(\texttt{z = (z<0)? Math.max(z,x): x+10;},\, wp(\texttt{x=x+y;},\, Q))) \vee (x \geq 10) \,\},$$

  where

  $$wp(\texttt{x=x+y;},\, Q) = \{\, (z \neq 0 \wedge y \geq 0) \vee (x + y \geq 0) \,\},$$

  and

  $$wp(\texttt{z = (z<0)? Math.max(z,x): x+10;},\, \cdot) = \{\, (z < 0 \wedge Q_{1A}) \vee (z \geq 0 \wedge Q_{1B}) \,\}.$$

- **Inside the if (when $x < 10$):**
  - Before the `z`-assignment: $\{\, (z < 0 \wedge ((Math.max(z, x) \neq 0 \wedge y \geq 0) \vee (x + y \geq 0))) \vee (z \geq 0 \wedge (((x + 10) \neq 0 \wedge y \geq 0) \vee (x + y \geq 0))) \,\}$
  - After the `z`-assignment: $\{\, (z \neq 0 \wedge y \geq 0) \vee (x + y \geq 0) \,\}$
  - After the `x`-assignment: $\{\, (z \neq 0 \wedge y \geq 0) \vee x \geq 0 \,\}$

- **After the if:** $\{\, (z \neq 0 \wedge y \geq 0) \vee x \geq 0 \,\}$

Problem 6

(1) Code Block

Given precondition:

$$\{ x < 2 \}$$

Code with filled conditions:

```
{ x < -1 } // (A) weakest precondition for z = x * 2 to ensure z < -2
z = x * 2;
{ z < -2 } // (B) follows from (A) since 2*x < -2 ⇔x < -1
w = -z;
{ w > 2 } // (C) follows from (B) because if z < -2 then -z > 2
w = w - 1;
{ w > 1 } // (D) since (w > 2) implies (w - 1 > 1)
```

Sufficiency Analysis: The weakest precondition required to guarantee the final postcondition is $\{x < -1\}$. However, the given precondition is $\{x < 2\}$, which is much weaker. For example, if $x = 1$ (which satisfies $x < 2$), then the code executes as follows:

- $z = 1 \times 2 = 2$, so the condition $z < -2$ fails.
- Then $w = -2$ and after $w = w - 1$ we get $w = -3$, so the final postcondition $w > 1$ is not met.

Thus, the given precondition $\{x < 2\}$ is **Insufficient**.

(2) Code Block

Given precondition:

$$\{ x = y \wedge y > 0 \vee y \neq x \}$$

Backward Reasoning for the "if" Statement

The final postcondition is:

$$\{ x > y \wedge y > 0 \}$$

We consider the two branches of the conditional.

1. Then Branch (when $x == y$):

- **At entry:** We must have $x == y$.
- **Before executing `x++`:** We need to ensure that after the increment, the final condition holds. After `x++`, we have $x = \text{old\_}x + 1 = y + 1$. The final condition requires $(y + 1) > y$ (which is always true) and $y > 0$. Therefore, the branch must start with:

$$\{ x == y \wedge y > 0 \}$$

- **After** `x++`**:**

$$\{\, x = y + 1 \land y > 0 \,\}$$

2. Else Branch (when $x \neq y$):

- **At entry:** We know $x \neq y$.
- **Before executing** `x = y + 2`**:** In order to satisfy the final postcondition, we need the updated x to be greater than y and also $y > 0$. After the assignment, $x = y+2$ guarantees $x > y$ automatically, but to meet the postcondition we must have $y > 0$. Hence, the branch must start with:

$$\{\, x \neq y \land y > 0 \,\}$$

- **After assignment:**

$$\{\, x = y + 2 \land y > 0 \,\}$$

Since the conditional splits on $x == y$ and $x \neq y$, the overall weakest precondition for the entire if statement is the disjunction of the branch preconditions:

$$\{\, (x == y \land y > 0) \lor (x \neq y \land y > 0) \,\}$$

which simplifies to:

$$\{\, y > 0 \,\}$$

since regardless of whether $x == y$ or $x \neq y$, we require $y > 0$.

Filled Code:

```
{ y > 0 } // (E) weakest precondition for the if-statement
if (x == y) {
    { x == y ∧y > 0 } // (F) branch precondition for then branch
    x++;
    { x = y + 1 ∧y > 0 } // (G) postcondition for then branch
} else {
    { x ≠y ∧y > 0 } // (H) branch precondition for else branch
    x = y + 2;
    { x = y + 2 ∧y > 0 } // (I) postcondition for else branch
}
{ x > y ∧y > 0 } // (J) final postcondition (since in the then branch, x = y+1 > y, and in the else branch, x = y+2 > y)
```

Sufficiency Analysis: The weakest precondition required for the if-statement is $\{y > 0\}$. The given precondition is

$$\{\, x = y \land y > 0 \lor y \neq x \,\}$$

is equivalent to

$$\{\, (x == y \land y > 0) \lor (x \neq y) \,\}.$$

In the second disjunct ($x \neq y$), there is no guarantee that $y > 0$. For instance, if $x \neq y$ but $y \leq 0$, the branch precondition $\{x \neq y \wedge y > 0\}$ is not met. Thus, the given precondition does not ensure $y > 0$ in all cases and is therefore **Insufficient**.

Problem 7 (4 pts.): Finding Input Values

We will show that the following Java code

```
if (x >= y - b) {
    y = y + b * x;
    b = b + x + y;
} else {
    b = 1 - x;
    x = y - x;
}
System.out.printf("%b %b %b\n", b < 0, x > y, y < 0);
```

prints

`true false true`

(i.e.,

- $b < 0$ is `true`,
- $x > y$ is `false` (so $x \leq y$), and
- $y < 0$ is `true`),

if and only if the initial inputs (the three integers x, y, b) satisfy one of the following two sets of conditions.

Case 1: The `if` Branch is Executed

This branch is taken when

$$x \geq y - b.$$

Inside the branch the following assignments occur:

- y is updated to
$$y' = y + b \cdot x.$$
- Then b is updated to
$$b' = b + x + y',$$

$$b' = b + x + y + b\,x.$$

The output conditions must then hold on the updated variables:

1. $b' < 0$,
2. $x > y'$ is false, i.e., $x \le y'$,
3. $y' < 0$.

Thus, the necessary and sufficient conditions on the original inputs for the `if`-branch are:

$$\boxed{\begin{array}{rcl} x & \ge & y - b, \\ y + b\,x & < & 0, \\ x & \le & y + b\,x, \\ b + x + y + b\,x & < & 0. \end{array}}$$

(the inequality $x \le y + b\,x$ can also be written as $x(1-b) \le y$ when $1 - b > 0$ or reversed when $1 - b < 0$;)

Case 2: The `else` Branch is Executed

This branch is taken when

$$x < y - b.$$

Here the assignments are:

- b is updated to
$$b' = 1 - x.$$
- x is updated to
$$x' = y - x.$$

(y remains unchanged.)

The printed conditions are then:

1. $b' < 0 \implies 1 - x < 0 \implies x > 1$.
2. $x' > y$ is false, i.e., $x' \le y$. Since
$$x' = y - x,$$
the inequality $y - x \le y$ simplifies to $-x \le 0$ (i.e., $x \ge 0$); this is automatically true once $x > 1$.
3. $y < 0$.

Additionally, the branch condition

$$x < y - b$$

must hold.

Thus, the necessary and sufficient conditions for the `else`-branch are:

$$\boxed{\begin{array}{lcl} x & < & y - b, \\ x & > & 1, \\ y & < & 0. \end{array}}$$

Final Answer

The output

`true false true`

is produced if and only if the initial inputs (x, y, b) satisfy **either** of the following two sets of conditions:

Case 1 (if branch is taken):

$$\boxed{x \geq y - b \quad \wedge \quad (y + b\,x < 0) \quad \wedge \quad (x \leq y + b\,x) \quad \wedge \quad (b + x + y + b\,x < 0).}$$

Case 2 (else branch is taken):

$$\boxed{x < y - b \quad \wedge \quad x > 1 \quad \wedge \quad y < 0.}$$

Any triple (x, y, b) of integers satisfying one of these sets of conditions will cause the code to print `true false true`.