CSE 331 Summer 2021 HW1

General Rules:

- For logical operators, you may use words (e.g., “or”) or any standard symbols (e.g., “V”).
- Assume that
  - all numbers are integers
  - integer overflow will never occur
  - integer division rounds toward zero (as in Java) (Example: 5 / 2 = 2)
- Simplify but **do not weaken** your assumptions.

1. **Forward reasoning with assignment statements.** Write an assertion in each blank space indicating what's known about the program state, given the precondition and the previously executed statements. The first assertion in part (a) is supplied as an example. Additionally, you should rewrite your assertions to only refer to the current state of variables (i.e., do not use subscripts).

   a. {{ }}
   a = 0;
   {{ a = 0 }}
   b = a + 3;
   {{ _____ }}
   c = b / 2;
   {{ _____ }}
   a = b - c;
   {{ _____ }}
   d = a * b;
   {{ _____ }}

   b. {{ x >= 2 }}
   y = 4 - x;
   {{ _____ }}
   z = x / 3;
   {{ _____ }}
   y = z - y;
   {{ _____ }}

2. **Backward reasoning with assignment statements.** Find the weakest precondition for each sequence using backward reasoning, writing the appropriate assertion in each blank space.

   a. {{ _____ }}
   a = a + 5;
   {{ _____ }}
   b = a / 2;
   {{ _____ }}
   b = b - 3;
   {{ b <= -1 }}

   b. {{ _____ }}
   q = 1 - r;
   {{ _____ }}
   r = q * q;
   {{ _____ }}
   z = r - q;
   {{ z > q }}

3. **Forward reasoning with if/else statements.** Find the strongest postcondition for the following conditional statement using forward reasoning, inserting the appropriate assertion in each blank.

```
{{ x != 1 }}
if(x >= 1) {
        {{ _____ }}
        y = 1 - x;
        {{ _____ }}
} else {
        {{ _____ }}
        y = 1 + x;
        {{ _____ }}
}
{{ _____ }}
```

4. **Backward reasoning with if/else statements.** Find the weakest precondition for the following conditional statement using backward reasoning, inserting the appropriate assertion in each blank.

```
{{ _____ }}
if(x >= 0) {
        {{ _____ }}
        y = 1 - x;
        {{ _____ }}
} else {
        {{ _____ }}
        y = 1 + x;
        {{ _____ }}
}
{{ y < 1 }}
```

5. **Weakest conditions.** Circle the weakest condition in each row. If there are two equivalent conditions that are the weakest ones, circle both of them. If some of the conditions are incomparable, note that instead.

a.	{{ x < 1 }}	{{ x < 2 }}	{{ x != 2 }}
b.	{{ |y| >= 2 }}	{{ if y <= 2, then y < -2 }}	{{ (y-2) * (y+2) >= 0 }}
c.	{{ a > 0 and b <= 0 }}	{{ if a < 0, then b >= 0 }}	

6. **Hoare triples.** State whether each Hoare triple is valid. If it is invalid, give an initial program state as a counterexample.

    a. {{ x < -2 }}
    y = 1 + 2 * x;
    {{ y < -2 }}

    b. {{ }}
    if(x > 1) {
        y = x / 2;
    } else {
        y = (2 - x) / 2;
    }
    {{ y > 0 }}

    c. {{ y > 0 }}
    if(x != y) {
        x = y + 1;
        y = 3 * y;
    } else {
        y = x - 1;
        x = -1;
    }
    {{ x < y }}

7. **Verifying correctness.** Fill in the intermediate assertions. Indicate with an arrow next to the line of code, whether you are reasoning forward or backward. Finally, state whether the code is correct, i.e., whether the Hoare triples are *all* valid.

```
{{ true }}
if(x <= 4 && -4 <= x) {
        {{ _____ }}
        z = x - 1;
        {{ _____ }}
        y = z * z;
        {{ _____ }}
} else {
        {{ _____ }}
        z = x * x;
        {{ _____ }}
        y = 33 - z;
        {{ _____ }}
}
{{ y <= 16 }}
```