CSE 331 Autumn 2020 Homework 1

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)

1. **Forward reasoning with assignment statements.** Write an assertion in each blank space indicating what is known about the *program state*, given the precondition and the previously executed statements. The first assertion in part (a) is supplied as an example.

   Additional rules for this problem:

   - Rewrite your assertions to only refer to the current state of variables (no subscripts).
   - Simplify but **do not weaken** your assertions.

   a. {{ }}
   x = 2;
   {{ _____ }}
   y = 5 * x;
   {{ _____ }}
   z = y - 4;
   {{ _____ }}
   y = z / 3;
   {{ _____ }}
   z = x – y;
   {{ _____ }}

   b. {{ x != 1 }}
   y = x;
   {{ _____ }}
   x = x – 2;
   {{ _____ }}

   c. {{ x >= 20 }}
   x = -x;
   {{ _____ }}
   x = x / 4;
   {{ _____ }}
   x = x – 5;
   {{ _____ }}

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. {{ _____ }}
x = x * 3;
{{ _____ }}
y = 4 + x;
{{ y >= 16 }}

b. {{ _____ }}
y = 8 - x;
{{ _____ }}
x = x * 3;
{{ x <= y }}

c. {{ _____ }}
x = x * 3;
{{ _____ }}
z = x - 2y;
{{ z <= y and z != y }} (another way of writing "z < y")

d. {{ _____ }}
t = 9 – 2*w;
{{ _____ }}
r = w + 3*r;
{{ _____ }}
s = w + 6;
{{ r < s and s <= t }}

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 >= 0 }}
if (x != 0)
        {{ _____________________________________ }}
        x = -x;
        {{ _____________________________________ }}
else
        {{ _____________________________________ }}
        x = x - 1;
        {{ _____________________________________ }}
{{ _____________________________________ }}
```

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)
        {{ _____________________________________ }}
        z = x + 3;
        {{ _____________________________________ }}
else
        {{ _____________________________________ }}
        z = 5 – x;
        {{ _____________________________________ }}
{{ z > 3 }}
```

5. **Hoare triples.** State whether each Hoare triple is valid. If it is invalid, give a counterexample.

a.
```
{{ x >= y + 1 }}
z = x – y;
{{ z >= 2 }}
```

b.
```
{{ x <= 0 }}
y = 3 – x;
{{ y > 0 }}
```

c.
```
{{ }}
if (x < 4)
        y = x / 3;
else
        y = -x + 5;
{{ y <= 1 }}
```

d.
```
{{ x > 0 }}
if (x >= 10)
        x = x % 10;
else
        x = 1;
{{ x > 0 }}
```

6. **Weakest conditions.** Circle the weakest condition in each list.

a. {{ a <= 0 }} {{ a < 2 }} {{ a != 2 }}

b. {{ |b| < 5 }} {{ b < 5 }} {{ b < -5 }}

c. {{ x >= 0 || y > 0 }} {{ if x <= 0, then y > 0 }}

d. {{ |z| > 10 }} {{ if z > 0, then z > 10 }}

7. **Verifying correctness.** For each block of code, 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.

   a. {{ x <= 1 }}
   y = x - 2;
   {{ _____ }}
   z = 3 * y;
   {{ _____ }}
   z = z + 6;
   {{ z <= 0 }}

   b. {{ 2x < w + 4 }}
   y = w + 2;
   {{ _____ }}
   x = x * 2;
   {{ _____ }}
   z = x – 2;
   {{ z < y }}

   c. {{ s != t and t != 0 }}
   if (s > t)
   {{ _____ }}
   s = s / t;
   {{ _____ }}
   else
   {{ _____ }}
   s = t – s;
   {{ _____ }}
   {{ s >= 1 and t != 0 }}