- integer overflow will never occur
- division is truncating integer division (as in Java)

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

a. {{ x <= 0 }}
y = 2*x;
{{ y < 0 }}

Invalid
when x=0, y=0

b. {{ x >= y }}
z = x – y;
{{ z >= 0 }}

valid

c. {{ }}
if (x >= 10)
    y = x % 7;
else
    y = x-1;
{{ y < 9 }}

valid

d. {{ x < 0 }}
if (x < 100)
    x = -1;
else
    x = 1;
{{ x < 0 }}

valid

2. **Weakest conditions.** Circle the weakest condition in each set.

a. {{ x > 10 }} {{ x >= 10 }} {{ x = 20 }}

b. {{ t != 0 }} {{ t = 2 }} {{ t > 0 }}

c. {{ x > 0 and y > 0 }} {{ x > 0 or y > 0 }}

d. {{ |x+y| > w }} {{ x+y > w }}

3. **Forward reasoning with assignment statements.** Write an assertion in each blank sp
what is known about the *program state*, given the precondition and the previously e
statements. Be as specific as possible. The first assertion in part (a) is supplied as an

a. {{ }}

```
x = 10;
```

{{ x = 10 }}

```
y = 3 * x;
```

{{ x=10, y=30 }}

```
z = y + 6;
```

{{ x=10, y=30, z=36 }}

```
x = z / 2;
```

{{ x=18, y=30, z=36 }}

```
y = 0;
```

{{ x=18, y=0, z=36 }}

b. {{ x < 0 }}

```
y = x;
```

{{ x<0, y=x }}

```
y = y + 5;
```

{{ x<0, y=x+5 }}

c. {{ |x| > 8 }}

```
x = -x;
```

{{ x>8 or x<-8 }}

```
x = x / 2;
```

{{ x>4 or x≤-4 }}

```
x = x + 1;
```

{{ x≥5 or x≤-3 }}

d. {{ y > 2*x }}

```
y = y * 3;
```

{{ y>6x }}

```
x = x + 1;
```

{{ y>6x-6 }}

4. **Backward reasoning with assignment statements.** Find the weakest precondition for each s using backward reasoning, and write the appropriate assertion in each blank space.

a. {{ $x > 0$ }}

```
x = x + 5;
```

{{ $x > 5$ }}

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

{{ y > 10 }}

b. {{ $x \geq \frac{1}{2}w - 5$ }}

```
y = w – 10;
```

{{ $x \geq \frac{1}{2}y$ }}

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

{{ x >= y }}

c. {{ $s \leq 2,\ w > 0$ }}

```
t = 2 * s;
```

{{ $s \leq 2,\ 2s + w \geq t$ }}

```
r = w + 4;
```

{{ $r \geq 2s + w,\ 2s + w \geq t$ }}

```
s = 2*s + w;
```

{{ r >= s and s >= t }}

5. **Backward reasoning with if/else statements.** Find the weakest precondition for the follow conditional statement using backward reasoning, inserting the appropriate assertion in eac

{{ $x > 0$ or $(x < 0$ and $x != -1)$ }}

```
if (x >= 0)
```

{{ $x != 0$ }}

```
    z = x;
```

{{ $z != 0$ }}

```
else
```

{{ $x != -1$ }}

```
    z = x + 1;
```

{{ $z != 0$ }}

{{ z != 0 }}

6. **Verifying correctness.** For each block of code, fill in the intermediate assertions (working either forward or backward or some combination), then use them to state whether the code is correct. I.e., whether the Hoare triples for the pre- and post-condition are valid.

a. {{ x >= 1 }}
y = x – 1;
{{ y ≥ 0 }}
z = 2 * y;
{{ z ≥ 0 }}
z = z + 1;
{{ z > 1 }}

Incorrect

z ≥ 1

using forward reasoning
I found postcondition is {z ≥ 1}
which doesn't imply {z > 1} since
z maybe 1, so the code is incorrect

using backward reasoning.
I found precondition {2x ≥ w} which
is the same with precondition that
is given, so code is correct

b. {{ 2x >= w }}
y = w – 2;
{{ 2x – 2 ≥ y }}
x = 2*x;
{{ x – 2 ≥ y }}
z = x – 2;
{{ z >= y }}

2x ≥ w

Correct

c. {{ y >= 0 }}
if (x == y)
{{ y ≥ 0, x = y }}
x= -1;
{{ y ≥ 0, x = -1 }}
else
{{ y ≥ 0, x != y }}
x = y - 1;
{{ y ≥ 0, x = y-1 }}
{{ x < y }}

Correct

{y ≥ 0, x = -1 or x = y-1}

y ≥ 0, if x = -1 x < y
if x = y-1 x < y

So my precondition implies
given precondition

∴ code is correct