Is this wrong?

```
int y;
if (x < 0) {
  y = 1;
}
if (x >= 0) {
  y = 0;
}
```

- syntax
  - no problems
    - each statement is a legal statement
    - only rules about statement order have to do with defining names and initializing variables before they are used
- logic
  - no problems
    - assuming the intent is that y is 1 if x < 0 and 0 otherwise

Is this wrong?

```
int y;
if (x < 0) {
  y = 1;
}
if (x >= 0) {
  y = 0;
}
```

- structural semantics
  - as written, the two ifs are independent
    - in principle, both conditions can be true, both can be false, or one can be true and the other false
    - y could end up as 1, 0, or uninitialized
  - but that's not the case with these particular conditions
    - exactly one condition will be true
    - exactly one of y = 1 and y = 0 will occur – y will not be left uninitialized
  - it is desirable for the program structure to match the situation
    - use the patterns – do or not do, two alternatives, more than two alternatives (do nothing is one, do nothing is not one)

The Big Picture

Repetition – (loops)

- syntax and semantics
- programming with loops

Loops – Syntax and Semantics

- `while` loop

```
while ( condition ) {
  body
}
```

  - test the condition
    - if it is true, do the body and test the condition again
    - if it is false, end the loop

- `for` loop

```
for ( initialization ; condition ; update ) {
  body
}
```

  - do the initialization
  - test the condition
    - if it is true, do the body followed by the update and test the condition again
    - if it is false, end the loop

Semantics – `while`, `for`

```
int n = 20;
while ( n > 0 ) {
  System.out.println(n);
  n = n/2;
}
```

```
20
10
5
2
1
```

```
for ( int x = 0 ; x < 10 ; x = x+2 ) {
    System.out.println(x/2);
}
```

```
0
1
2
3
4
```


Loops – Syntax and Semantics

- both kinds of loops are equivalent
  - you should be able to read both
  - some people prefer `for` loops for counting loops and `while` loops otherwise
  - it is OK to only write one kind (`for` loops recommended)

Loops – Syntax and Semantics

- `break`
  - only legal in a loop body
  - immediately exit the nearest enclosing loop
- `continue`
  - only legal in a loop body
  - skip the rest of the loop body; go straight to the loop condition (`while`) or update and loop condition (`for`)

`break`, `continue`

What does this code do?

```
Scanner scanner = new Scanner(System.in);
int x, y;
x = 0;
while ( x < 20 ) {
  y = scanner.nextInt();
  if ( y == 0 ) {
    break;
  }
  System.out.println(y);
  x = x+1;
}
```

Prints out (only) the non-zero numbers that the user enters, stopping when 20 numbers have been printed or the user enters a 0 (whichever comes first).

What does this code do?

```
Scanner scanner = new Scanner(System.in);
int x, y;
x = 0;
while ( x < 20 ) {
  y = scanner.nextInt();
  if ( y == 0 ) {
    continue;
  }
  System.out.println(y);
  x = x+1;
}
```

Prints out (only) non-zero numbers that the user enters, stopping when 20 numbers have been printed.

Program Development With Loops

Strategy –

- identify the core structure of a task
  - does this task involve repetition?
- get more specific – identify which variation
  - repetition
    - repeat until condition
    - repeat as long as condition
    - repeat *n* times
    - repeat for values *a* to *b* (with an interval of *c*)

    } counting loops
- fill in the elements in the pattern
  - basic task – what repeats? how many times / how long?
  - loop variables – what changes from one repetition to the next?
  - timing of the loop condition test
  - declare, initialize, update loop variables – where is the value needed? how do things start? how do things change from one repetition to the next?

Program Development With Loops

Code structure –

- repeat until condition

```
while ( !condition ) { body }
for ( ; !condition ; ) { body }
```

- repeat as long as condition

```
while ( condition ) { body }
for ( ; condition ; ) { body }
```

- repeat *n* times (counting loop)

```
for ( int i = 0 ; i < n ; i++ ) { body }
```

- repeat for values *a* to *b* (by an interval of *c*)

```
for ( int i = a ; i <= b ; i += c ) { body }
```

  - < if the value b is not included in the range, <= if it is

Program Development With Loops

- what repeats? → loop body
- how many times / how long? → loop condition
- what changes from one repetition to the next? → loop variables
- what is the timing of the loop condition test? → loop structure
  - before any part of the loop body task → standard
  - in the middle of the loop body task → fencepost
  - after the loop body task → clever initialization or fencepost
- declare, initialize, update loop variables
  - where is the value needed? → location of declaration
    - for no longer than a single repetition → local in body
    - only during the loop repetitions, spans repetitions → in init part of `for` loop or before `while` loop
    - after the loop as well → before loop
  - how do things start? → initialization
  - how do things change from one repetition to the next? → update