Consider the following source code program written in a high level programming language. Assume that multiplication is left-associative and has higher precedence than addition.

```
int w, x, y;
x = 1;
y = 2;
w = x * y * x + 7 * x
print y
```

Problem 1. Convert the program to three-address code, introducing temporary variables as necessary.

One possible solution:

$$\begin{aligned} &x \leftarrow 1 \\ &y \leftarrow 2 \\ &t_1 \leftarrow x * y \\ &t_2 \leftarrow t_1 * x \\ &t_3 \leftarrow 7 * x \\ &w \leftarrow t_2 + t_3 \\ &print\ y \end{aligned}$$

Most students submitted correct IR for the source code, with the following variations, all of which are acceptable:

- Different students had different (legal) orderings of IR statements, e.g., some students generated IR for the left operand (`x*y*x`) first, whereas some generated IR for the right operand (`7*x`) first.
- Some students reused temporary variables to reduce the number of temporaries, whereas others created a new temporary in each instruction.

One common mistake among the (few) incorrect solutions was that some students calculated multiplication in a non-left-associative manner.

Problem 2. Assuming that all print instructions are critical instructions, show the output IR after dead code elimination is performed.

Solution:

$$\begin{gathered} y \leftarrow 2 \\ print\ y \end{gathered}$$

Almost all students answered this problem correctly. A few students were not sure what *critical instruction* meant in the context of dead code elimination. (Recall from the lecture that critical instructions are roots of computations that are necessary.)