

## Question 1 of 3

In lecture, we saw that in order to run a C program (e.g., `hello.c`), we first need to run the command `make hello`, and then run the command `./hello`.

- a) What does running `make hello` do?
- b) What does running `./hello` do?
- c) What might happen if you were to run `./hello` without first running `make hello`?

## Answers

- a) Running `make hello` uses `clang` to compile `hello.c`, a source code file written in C, into `hello`, a file that contains the program represented in machine code: the binary instructions that computers understand more directly.
- b) Running `./hello` runs the machine code
- c) If you had never compiled the program before, you might see an error because there is no file called `hello` in the current directory, since compiling the program generates the `hello` file. If you had compiled the program before, you might end up running an older version of the program, because you haven't yet re-compiled the latest version of the program.

## Question 2 of 3

In the first version of the game [Civilization](#), each character had an "aggression" rating, stored as an 8-bit integer. The programmers of the game gave one of the characters, Gandhi, an initial aggression rating of 1, the lowest of any character in the game. The game also had logic such that, once a character "adopted democracy," the character's aggression rating would decrease by 2.

However, the game had a bug whereby Gandhi would become extremely aggressive, with an aggression rating of 255, if he adopted democracy.

What might explain why Gandhi suddenly became so aggressive in the game?

### Answer

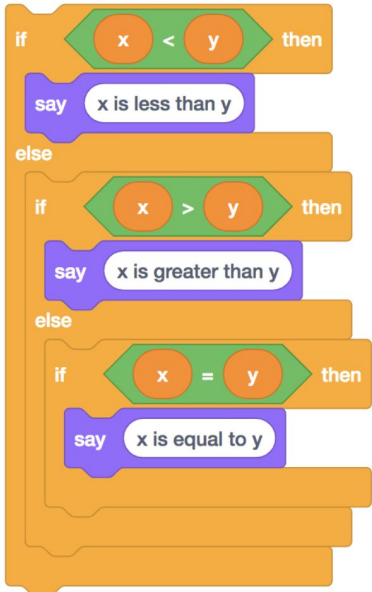
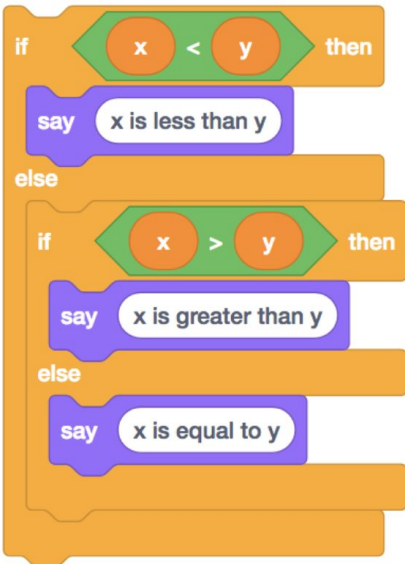
Gandhi's initial aggression rating, 1, might be represented as an 8-bit integer as 00000001. If we decrease his aggression rating by 1, we get 00000000. If we decrease his aggression rating by 1 again (since adopting democracy results in decreasing aggression by 2), Gandhi's aggression rating might wrap around to 11111111 due to integer overflow. The binary number 11111111 is the number 255 in decimal, which would explain why Gandhi ended up with an aggression rating of 255.


## Question 3 of 3

Recall that, in lecture, we saw the following two blocks of code, both of which print the same output.

Version 1	Version 2
<pre>if (x &lt; y) {     printf("x is less than y\n"); } else if (x &gt; y) {     printf("x is greater than y\n"); } else if (x == y) {     printf("x is equal to y\n"); }</pre>	<pre>if (x &lt; y) {     printf("x is less than y\n"); } else if (x &gt; y) {     printf("x is greater than y\n"); } else {     printf("x is equal to y\n"); }</pre>

These are really just the C equivalents of the following two blocks of Scratch code.

Version 1	Version 2
	

- a) Why, in C, do we use two equals signs (==) when we write `else if (x == y)`, whereas in Scratch we use just a single equals sign (=) in  ?
- b) Why is Version 2 of the code, whether implemented in Scratch or in C, marginally more efficient than Version 1?

## Answers

- a) In C, the single equal sign is already used to mean "assignment": it's used when we want to assign a value to a variable. Therefore, to represent a check for equality, we need to use something else, so we use two equals signs instead of one.
- b) Version 2 of the code avoids the need to evaluate a third Boolean expression in the case where  $x$  is equal to  $y$ . In Version 1, if  $x$  is equal to  $y$ , three Boolean expressions are evaluated: first checking if  $x$  is less than  $y$ , then checking if  $x$  is greater than  $y$ , and finally checking if  $x$  is equal to  $y$ . Version 2 only checks the first two Boolean expressions, recognizing that if the first two expressions are false, then the third one must be true.