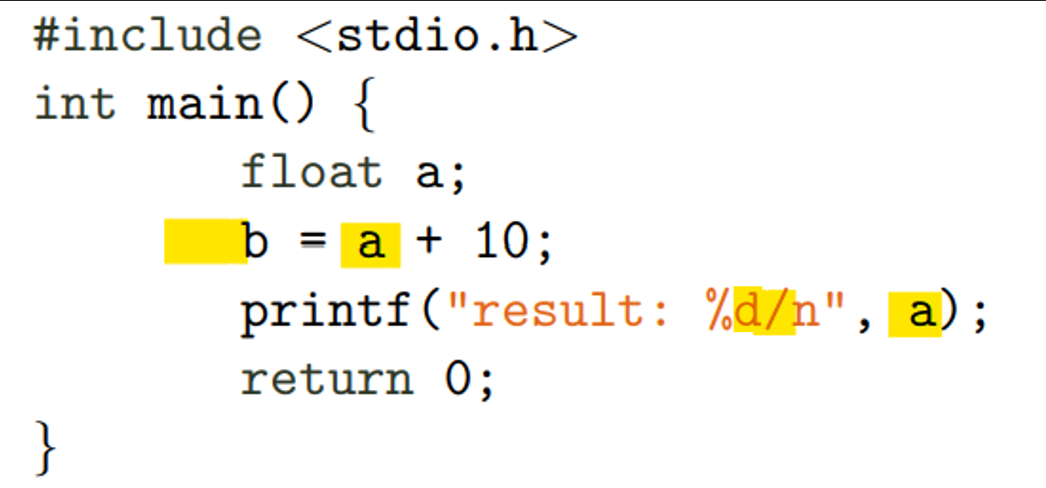
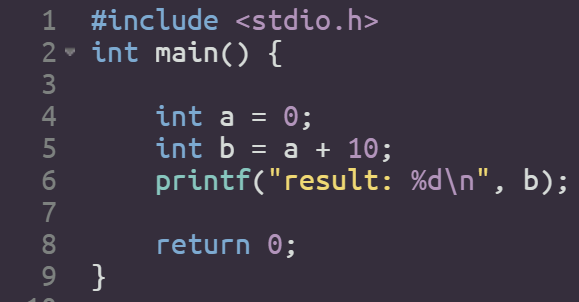
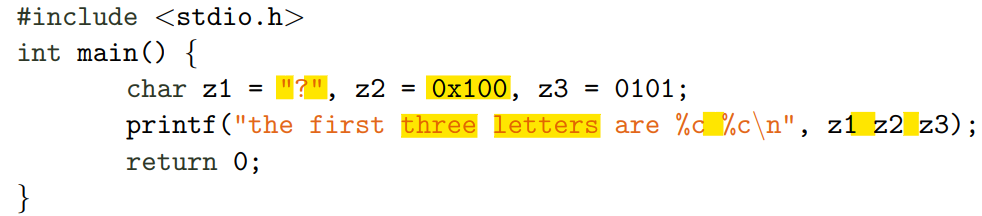
**Warmup (optional):**

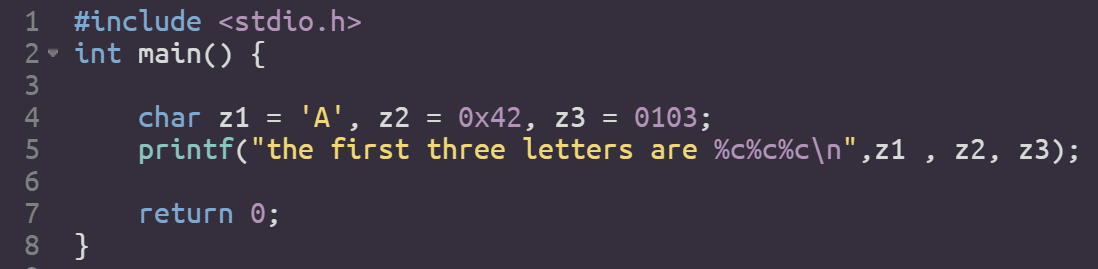
****

* **b is not declared yet, but it’s trying to be defined.**
* **a is not defined, but it’s trying to be used.**
* **To make a new line, it’s required to use a backslash, not a normal slash.**
* **The result is not in a. It’s not an error that would prevent the program from running, but it won’t give the desired outcome.**
* **The print method has a format specifier for an integer, while a is a float. Because we are just adding 10, I would change a to be an integer.**

****

****

* **To define a char you need to use the single quotes.**
* **The ASCII code have exactly 256 characters, but it begins in 0, so the last one is 255. Hence, you cannot define a char to be the character 256 (100 in hexadecimal), as it´s done here.**
* **It’s saying that they are three, but then there are only two “%c”.**
* **To add multiple variables to the function prinft, commas are required.**
* **It´s saying that the characters are letters, but they are not. z1 is two behind (63, it should be 65), z2 is non-existent and z3 is A, so it´s either not in order, or z3 is two behind.**

****

**Exercise 3.1.**

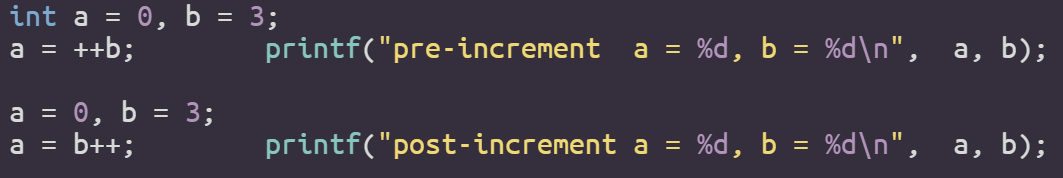
1. If you change **b--** to be **--b**, it will first take one from **b**, and **then** it will put that in **a** as well, while **b--** first puts **b** in **a**, and then takes one from **b**.

However, if you change it to be **b – 1**, it won’t change **b** at all.

1. It first adds one to **a** (9 🡪 10), takes that result and adds something else. To calculate that else, it takes **a** and adds one again (10 🡪11). It then adds the first 10 to 11 and gives as an answer a 21.

If we however, change that to be **a-- + --a**, it takes first **a** and will use it to the addition. Then it takes **a** again, subtracts one, and leave it. Then it takes **a** again, subtracts one and it **uses** **that**, after the subtraction, for the addition. Lastly, it adds **a** before operations (9) to **a** after them (7).

1. The pre-increment operator first adds one and then uses that, while the post-increment operator first uses that and then adds one.





**Exercise 3.2**

* char c = ’B’;
* short s = −1;
* unsigned int ui = 10;
* c != ’X’
* c + s

Here, the char c is casted into a short, since short is larger than char.

* ui > s

The short is casted into an unsigned int.

* ui ∗= 2.0

The 2.0 is casted into an unsigned int.