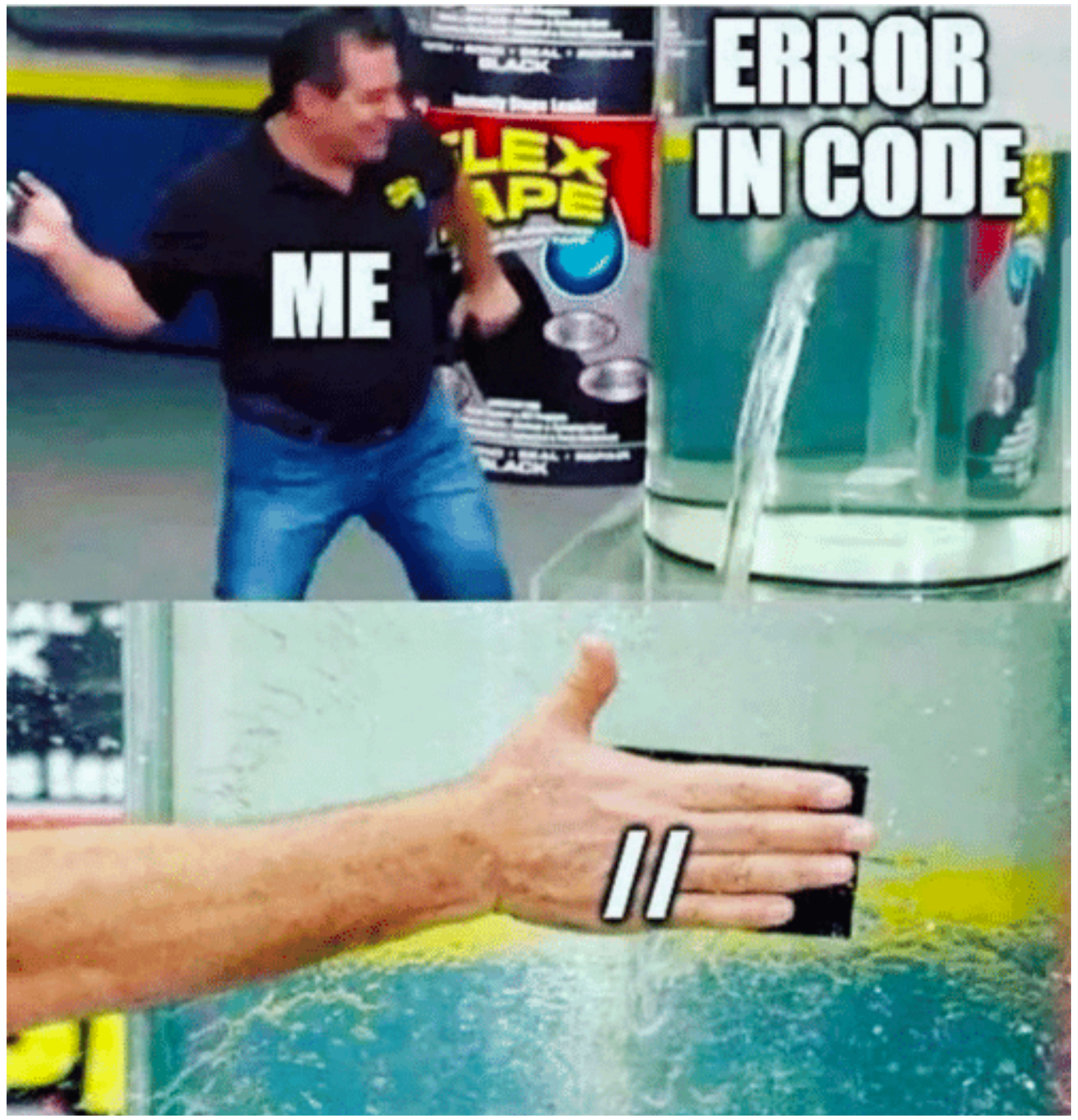
COMP1511 Week 3

structs and while-loops

Joanna Lin

What we'll cover today

- **structs**
  - what are **user-defined** types?
  - what are structs?
  - how do we work with structs?
- **while loops**
  - what are loops and why do we need them?
  - how do we structure a loop
  - how do they work?
  - nested (2D) while loops



Structs

User-defined variable type

- Structs serve to represent some kind of 'thing' whose features will be represented by using the variables that you put inside the struct. We can model the real world a little closer.
- Structs are user-defined types.
  - **user-defined**: You create a new variable type out of a combination of existing types (`int`, `double`s, `char`s).
  - Unlike `int`, `char` and `double`, there is an additional step of **defining** a variable type.
  - Only once you have defined your custom variable type can you actually declare a `struct` variable

Defining a struct type

What do we want to represent?

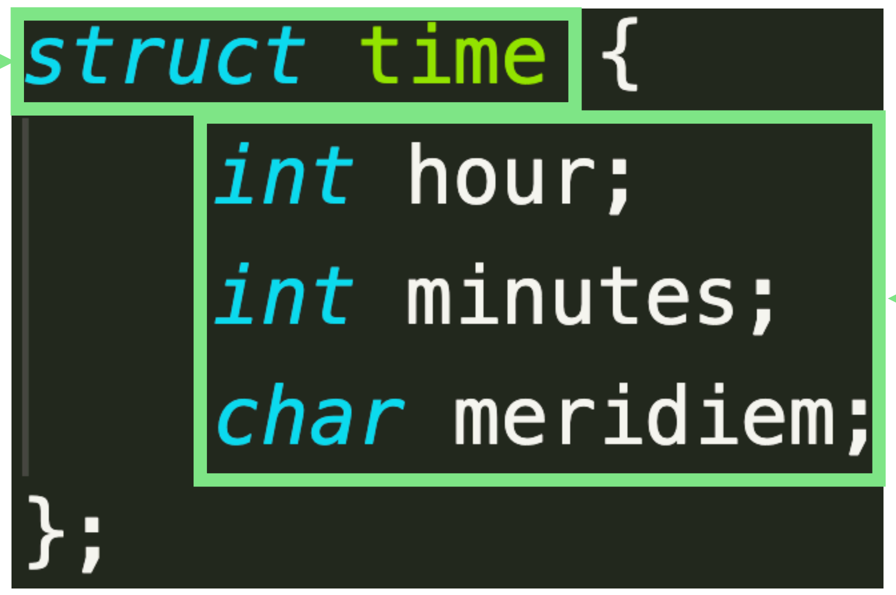
- Let's say we wanted to represent 'time' in using a variable.

3:14 PM

hour minute merīdiem

- We'd do this in C through structs.
- Before **`main`**, we would write:

State the name of the variable type (**`struct`** plus the thing you want to represent)



```c
struct time {
        int hour;
        int minutes;
        char meridiem;
};
```

* (don't forget the semicolon!)

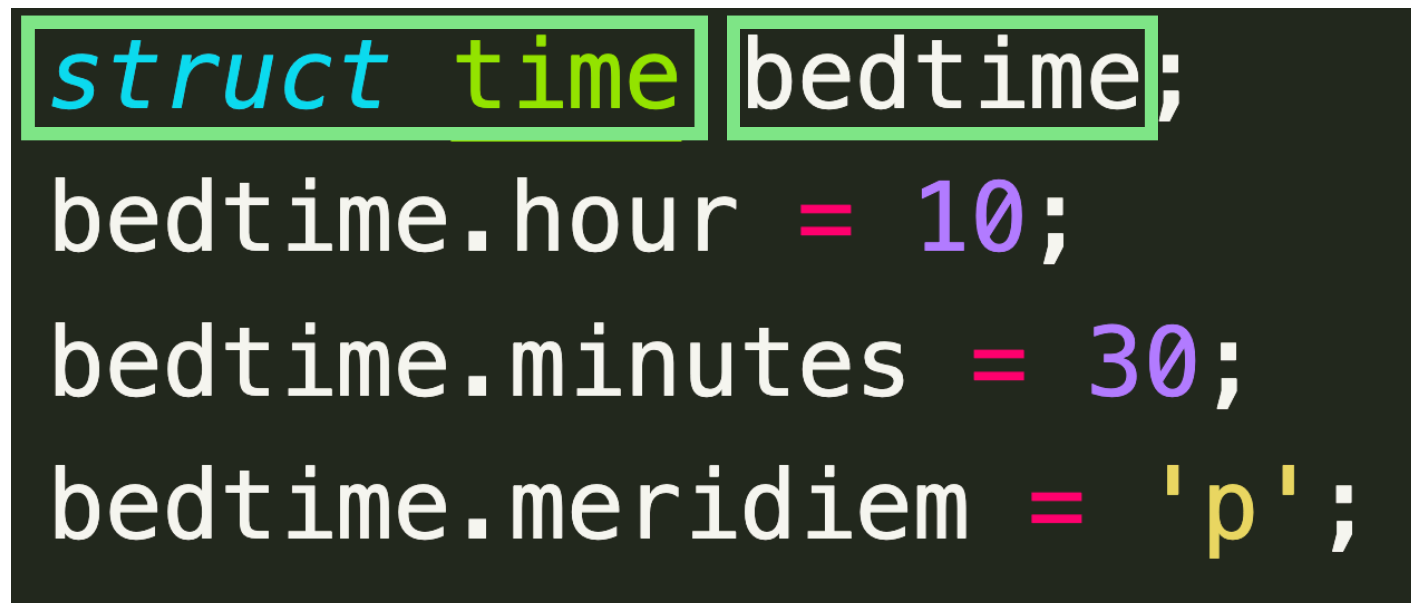
State what variables the struct will contain. These variables are called fields.

Working with Structs

Putting our type definition to use

- We now treat **`struct time`** as a type, and use it like **`int`**, **`double`**, **`char`** when declaring variables.
- We use '**`bedtime.hour`**' to access the field called 'hour' in our **`struct time`** called **`bedtime`**

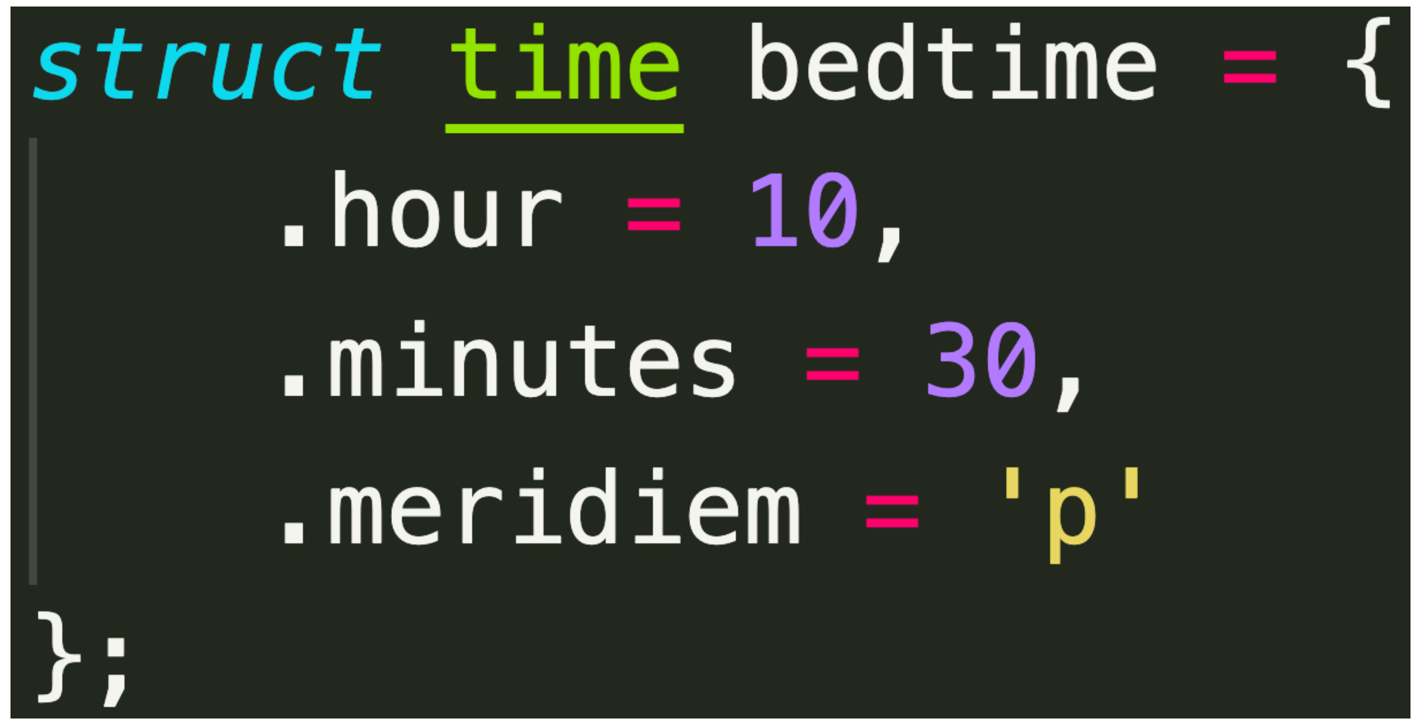
variable type variable name



```c
struct time bedtime;
bedtime.hour = 10;
bedtime.minutes = 30;
bedtime.meridiem = 'p';
```

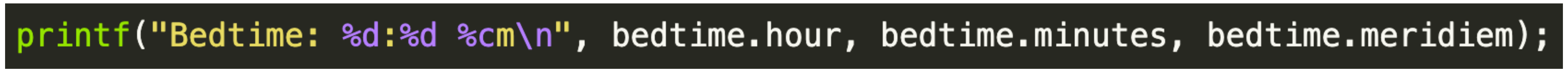
Remember the fields we can access are in our type definition

is the same as



```c
struct time bedtime = {
    .hour = 10,
    .minutes = 30,
    .meridiem 'p'
};
```

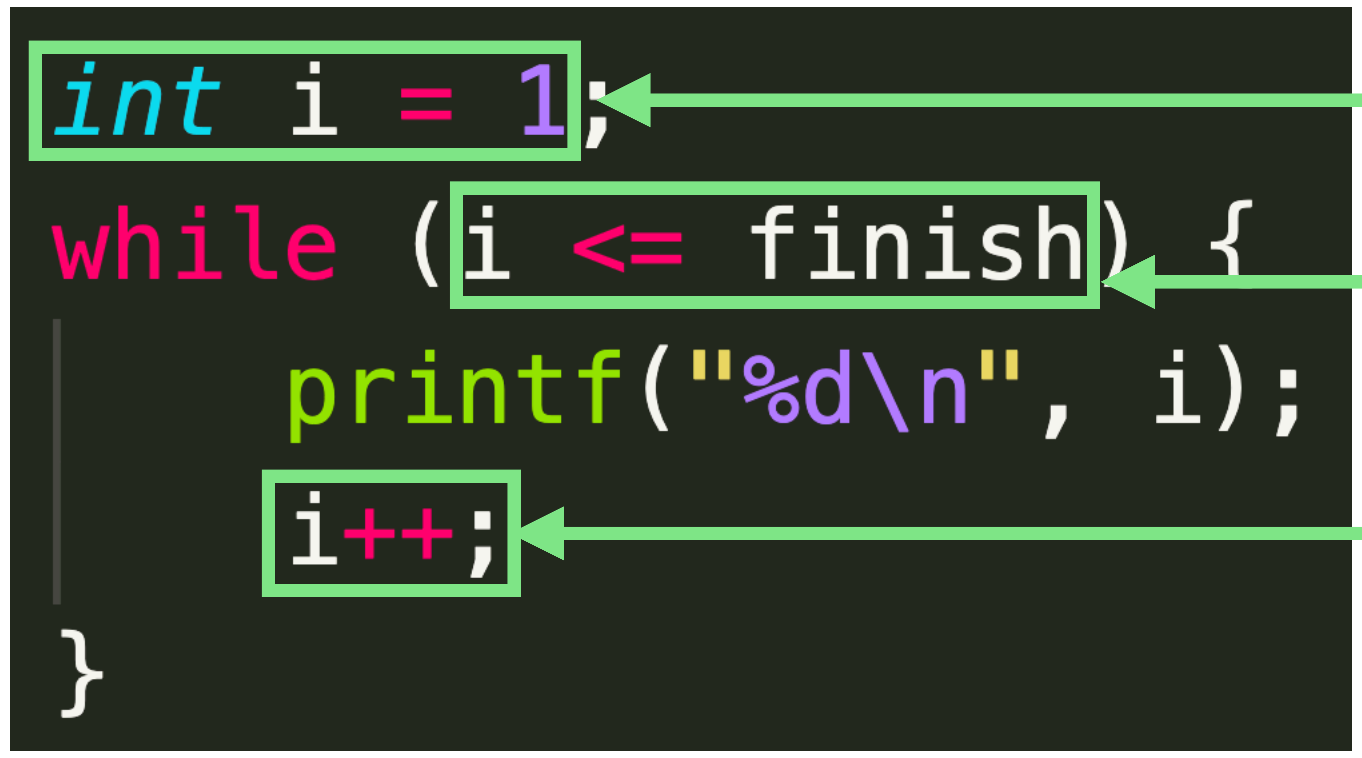
- We print the fields of a struct using this notation as well:



```c
printf("Bedtime: %d:%d %cm\n", bedtime.hour, bedtime.minutes, bedtime.meridiem);
```

While Loops

- **`while`** loops allow us to condense repeated logic
- Almost all **`while`** loops consist of 3 components:

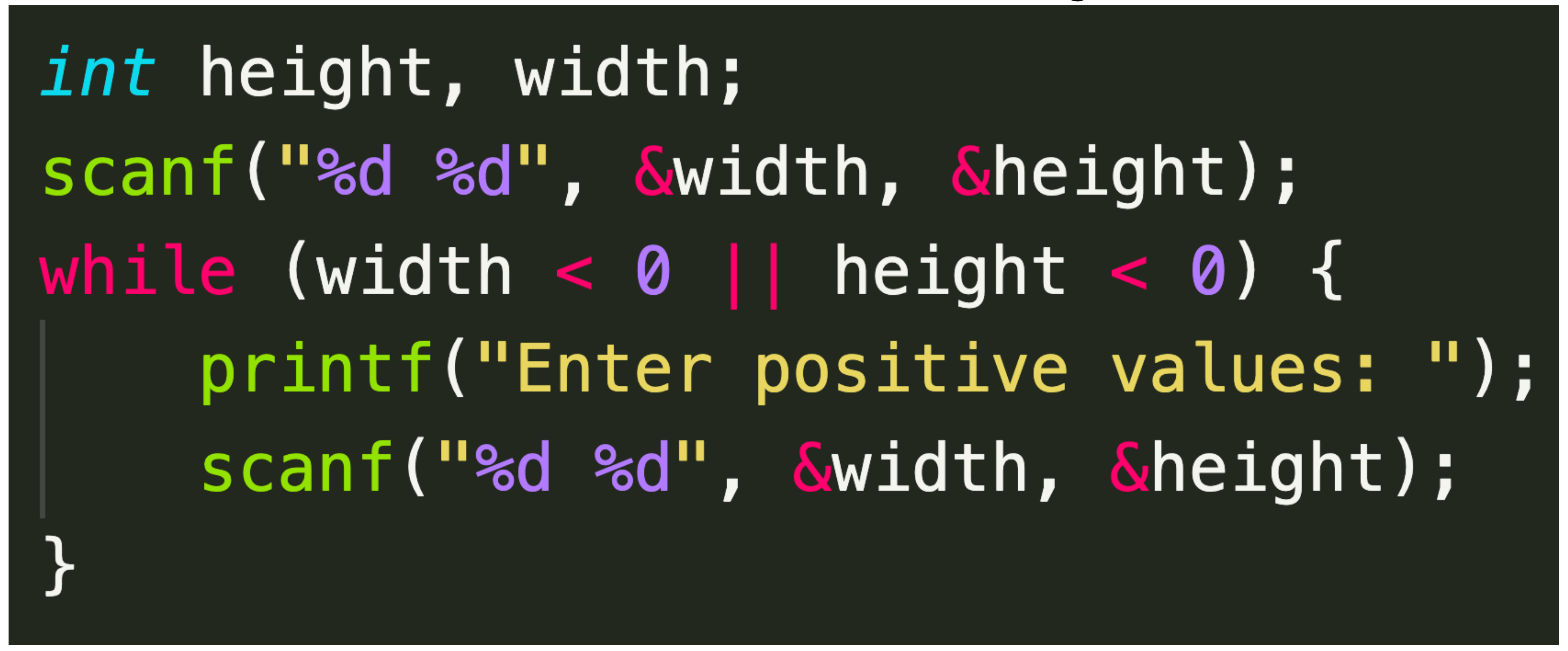


```c
int i = 1;
while (i <= finish) {
    printf("%d\n", i);
    i++;
}
```

1. A starting value(s) for the variable(s) controlling the loop
2. Condition for loop to continue
3. An iterating step. Note that: `i++`, `i += 1` and `i = i + 1` are all the same

A while loop that iterates through the values from 1 to finish, and prints them
Assume here that 'finish' is a positive int

Another example: Can you identify the 3 components?



```c
int height, width;
scanf("%d %d", &width, &height);
while (width < 0 || height < 0) {
    printf("Enter positive values: ");
    scanf("%d %d", &width, &height);
}
```

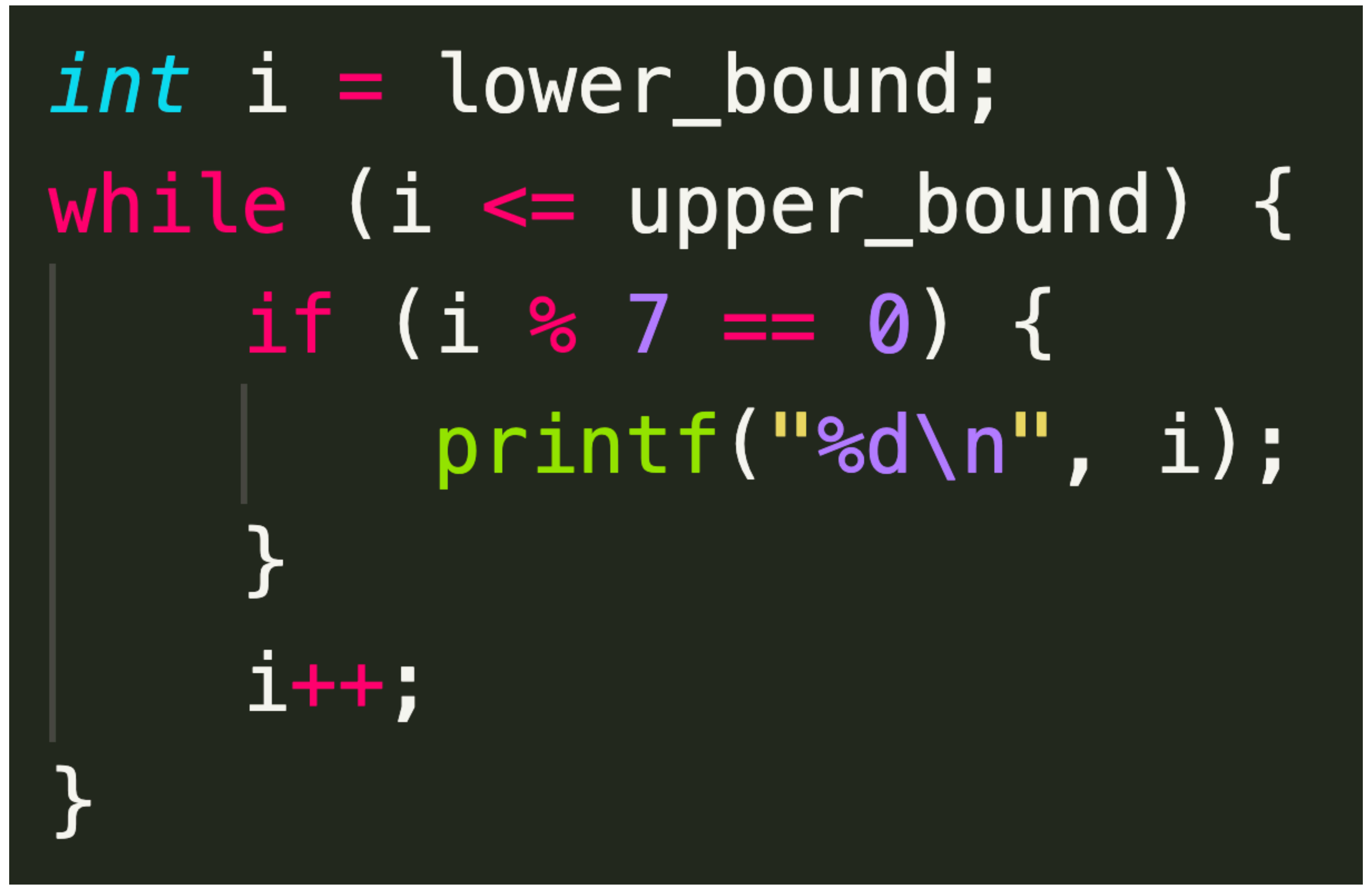
A while loop that keeps getting user input until the input is valid.

- What's happening in a while loop?
  - The condition is checked and, if true, the code inside the curly braces corresponding to the while loop is executed
  - Upon reaching the end of the code within the curly braces, the program goes back to check whether the condition is true or false. If true, the code within the curly braces is executed again.
  - This process repeats until the condition is false upon being checked.

if statements and loops

Putting them together

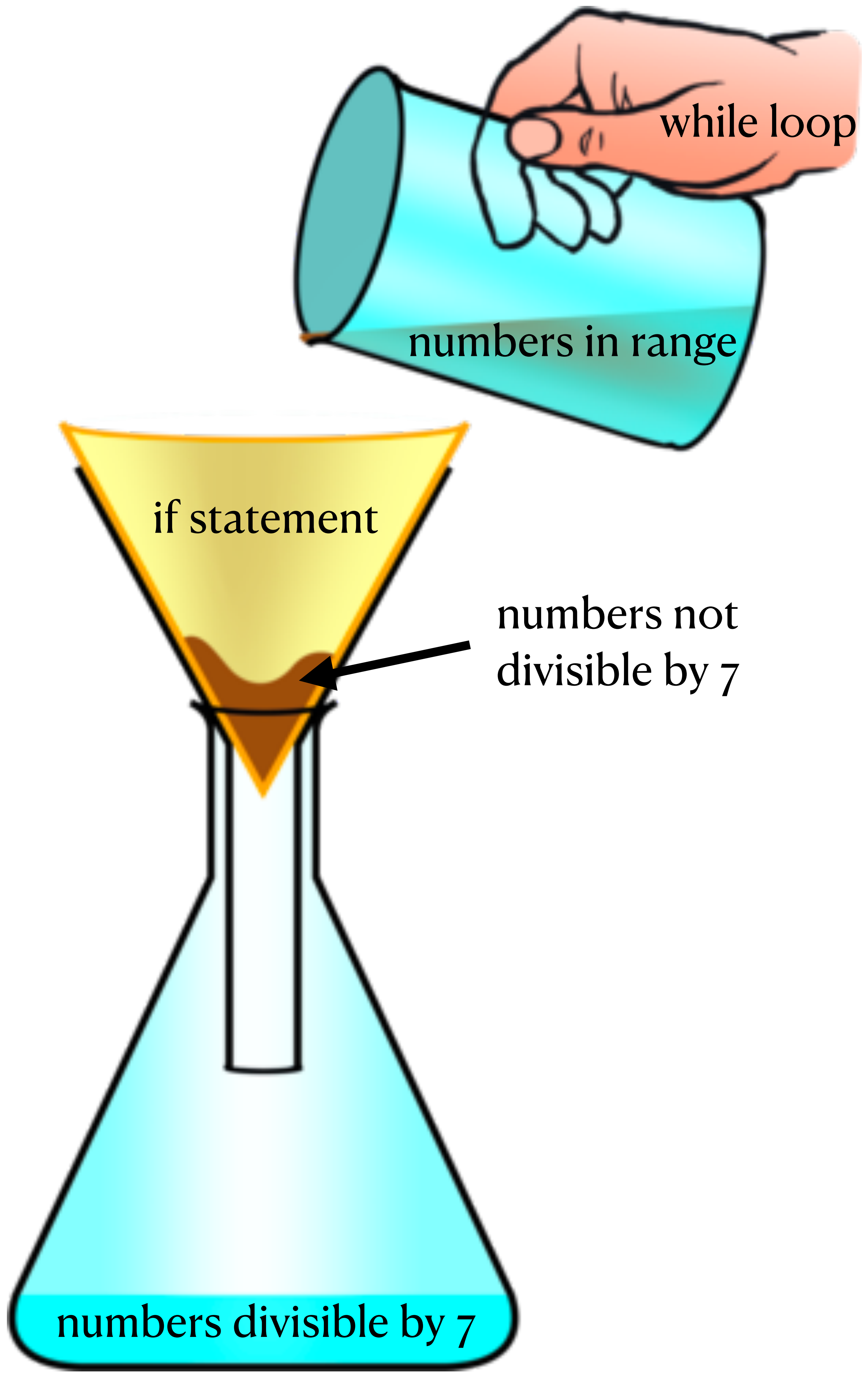
- Suppose we wanted to print only numbers divisible by 7 in a range of numbers. We'd write:



```
int i = lower_bound;
while (i <= upper_bound) {
    if (i % 7 == 0) {
        printf("%d\n", i);
    }
    i++;
}
```

Assume lower_bound and upper_bound are integers and lower_bound < upper_bound

Note that `i` is the variable that cycles through all the numbers, which is why the if-statement condition is on `i`



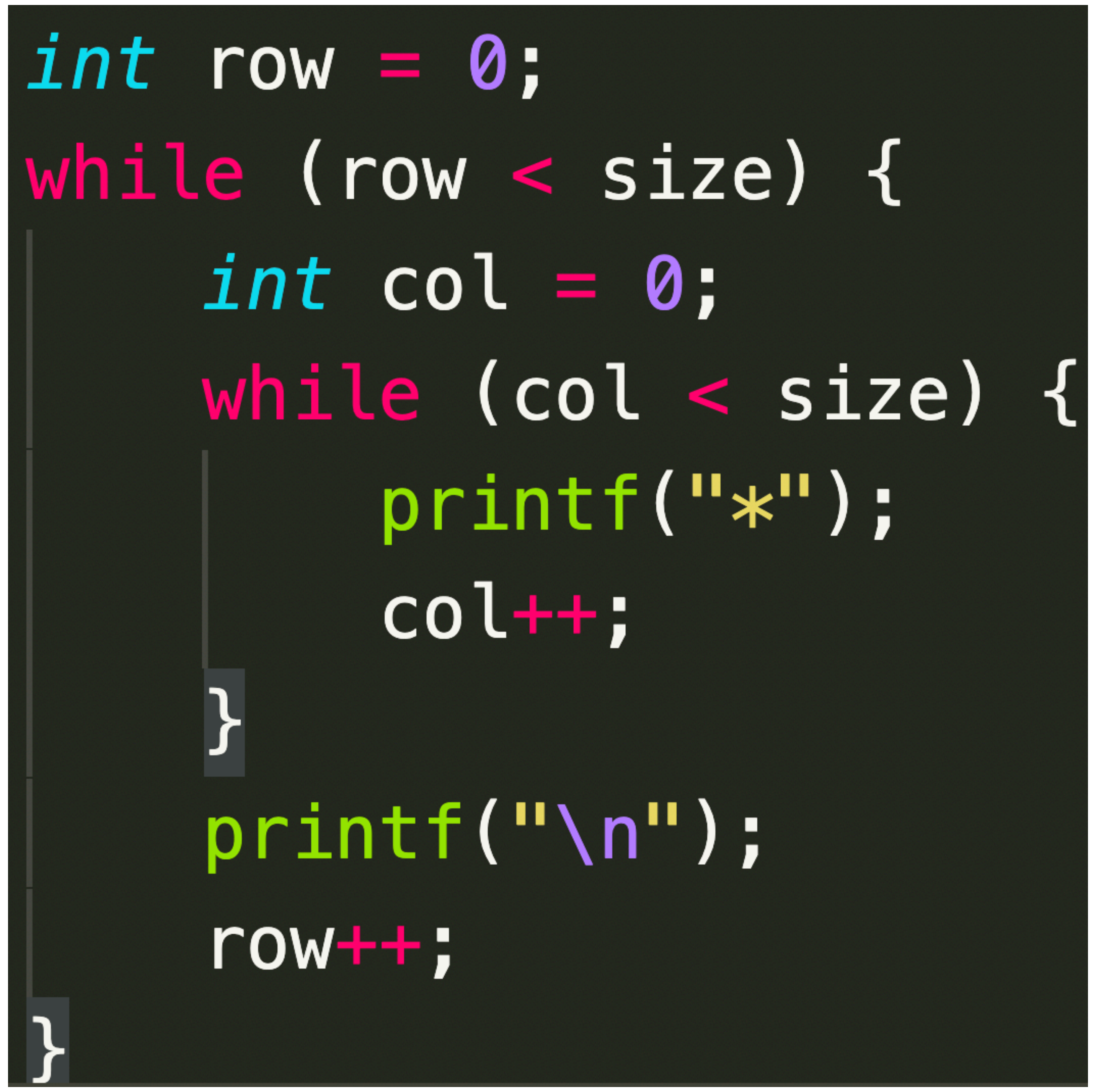
- We use the **`while`** loop to iterate over every possible candidate value, then use the **`if`** statement to 'filter out' what we don't want.

Nesting While Loops

- We sometimes need to put while loops inside while loops.
- Example, printing a ‘square’ of asterisks. Notice that the loop variable for the inner loop goes inside the outer loop

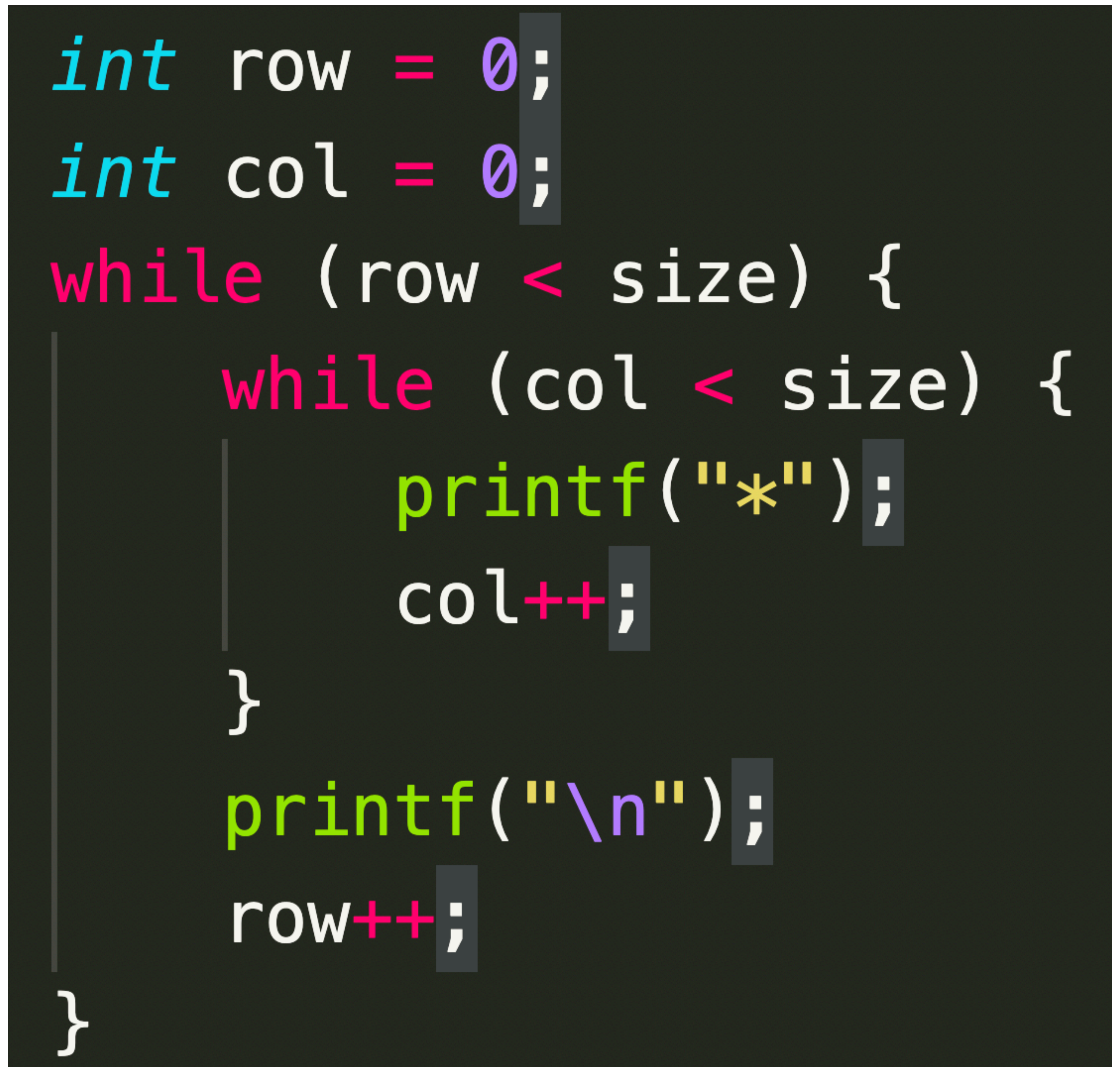
Sample output when size is 5:

```
*****
*****
*****
*****
*****
```



```
int row = 0;
while (row < size) {
    int col = 0;
    while (col < size) {
        printf("*");
        col++;
    }
    printf("\n");
    row++;
}
```

✓



```
int row = 0;
int col = 0;
while (row < size) {
    while (col < size) {
        printf("*");
        col++;
    }
    printf("\n");
    row++;
}
```

✗

Why is this wrong? What will it do instead?