COMP1511 Week 3

structs and while-loops

Joanna Lin

Reminders

- Assignment 0 and lab 3 due **Monday 8pm Week 4 (next week!)**
- Answer this survey if you ever find tutorial or lab content ambiguous, confusing, misleading, too difficult etc… or if you want to provide us with some nice feedback :)
- This week, we've scheduled many more help sessions. Check the schedule to see the full list. If you want to ensure you're seen at a help session, you should use Hale, our fast-pass booking system. Watch this video to learn more.

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 fields that you put inside the struct. We can model the real world a little closer.
- Structs are user-defined types.
  - You create a new variable type out of a combination of existing types (**`int`**s, **`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?

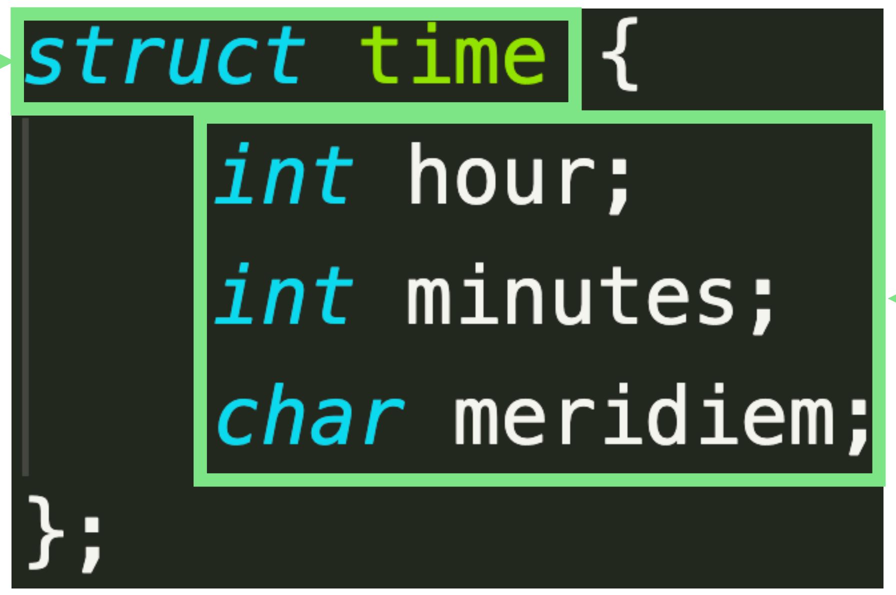
- Let’s say we wanted to represent ‘time’ using a variable.

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;
};
```

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

* (don’t forget the semicolon!)

Working with Structs

Putting our type definition to use

- We now treat **`struct time`** as a type, and use it in the same way we use **`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 'dot' notation as well:

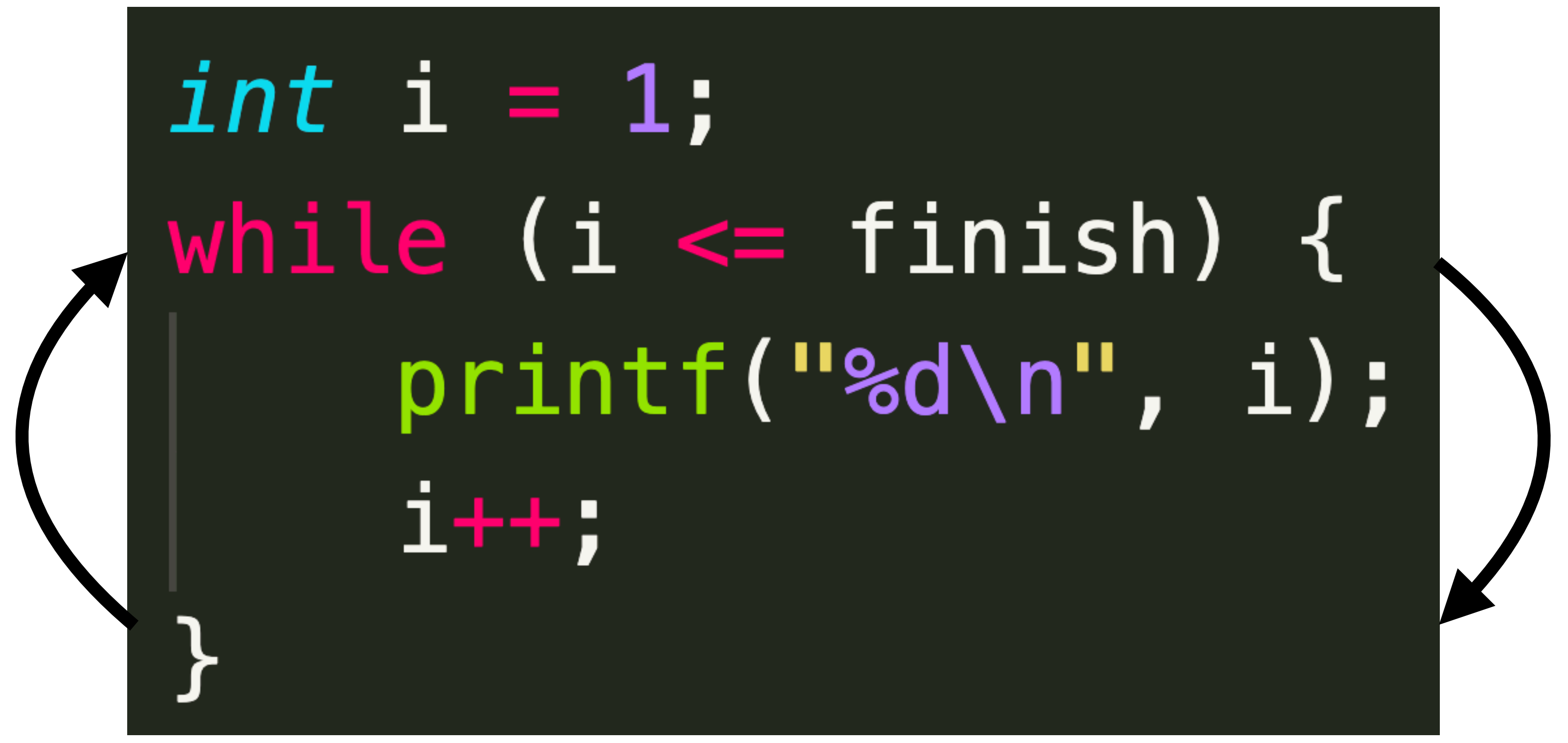
```c
printf("Bedtime: %d:%d %cm\n", bedtime.hour, bedtime.minutes, bedtime.meridiem);
```

While Loops

What are they and how do they work?

- **`while`** loops allow us to run the same piece of code repeatedly, without needing to write the code more than once.

Upon reaching the end of the code within the curly braces, the program goes back to check whether the condition is true or false



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

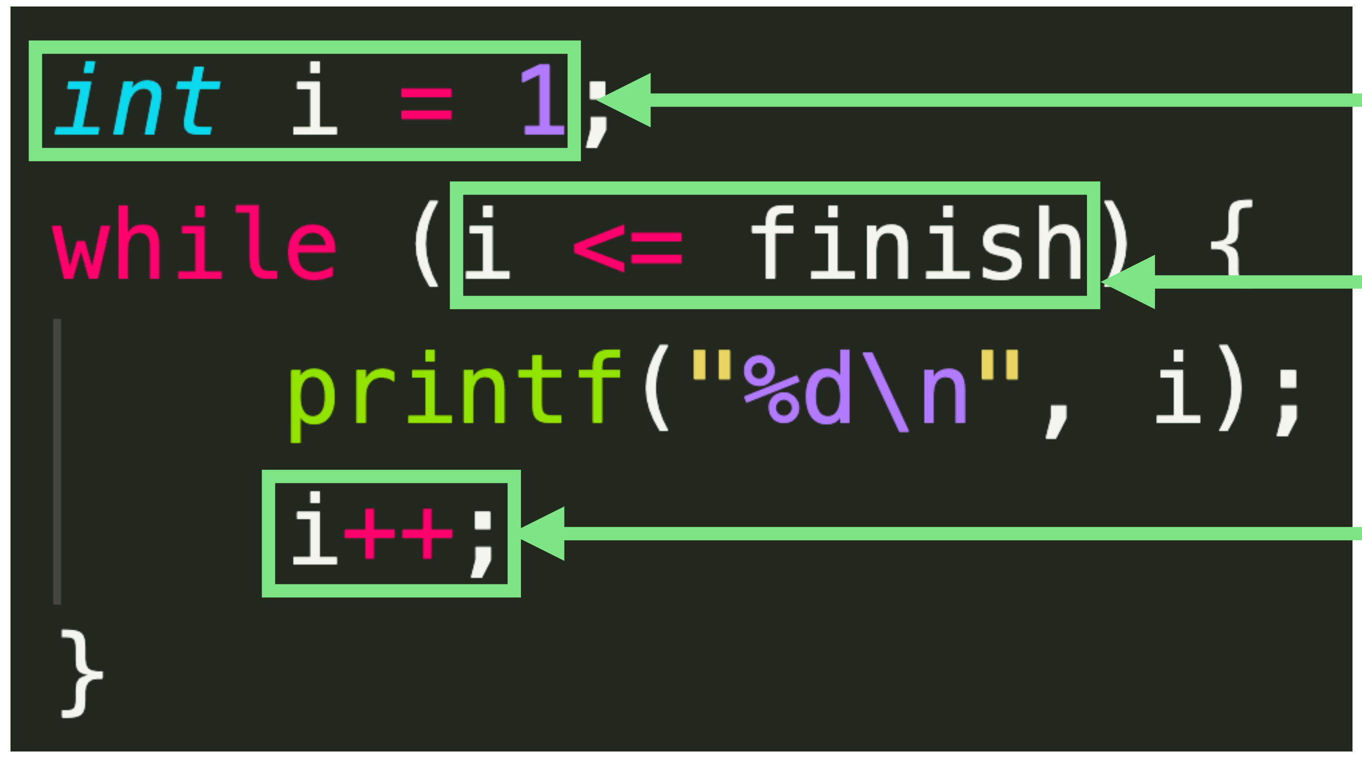
The condition is checked and, if true, the code inside the curly braces corresponding to the while loop is executed

This process repeats until the condition is false upon being checked.

While Loops

Basic Structure

- Almost all **`while`** loops consist of 3 components

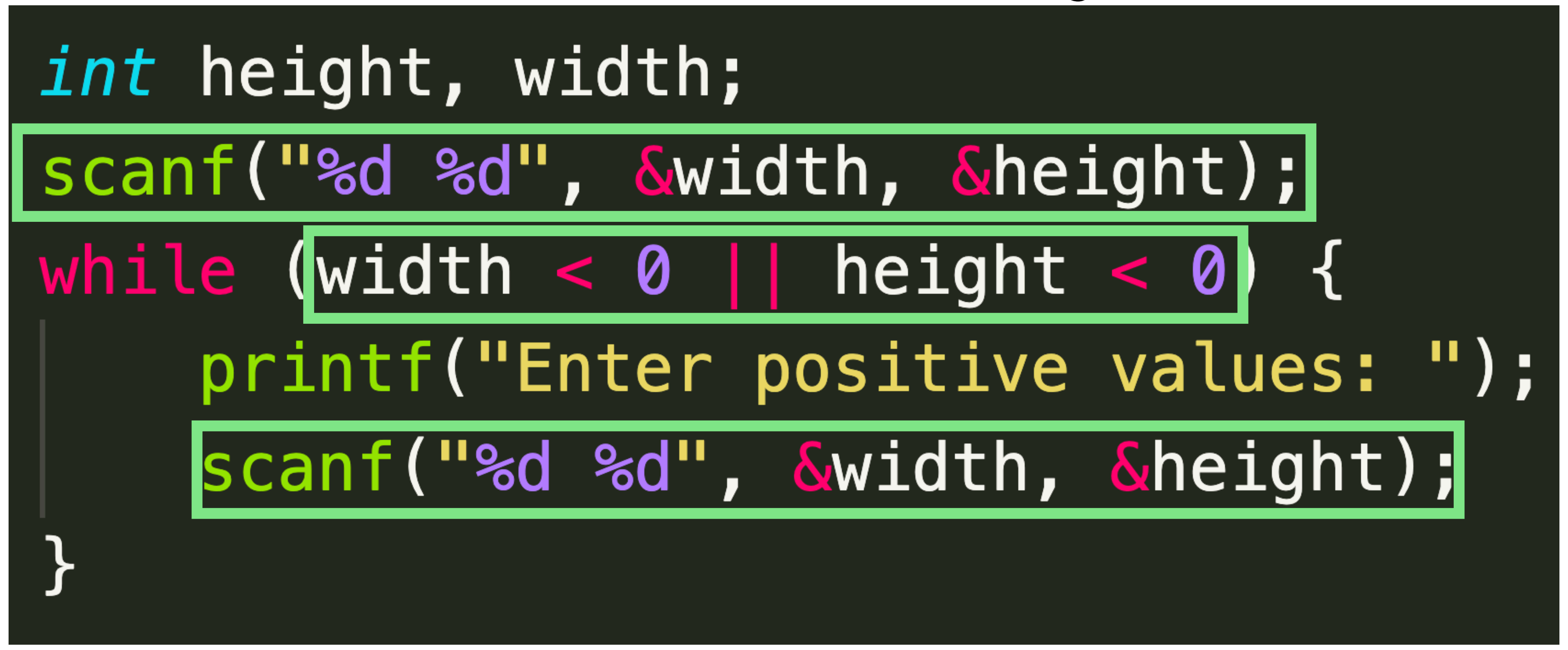


```
int i = 1;
while (i <= finish) {
    printf("%d\n", i);
    i++;
}
```

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?



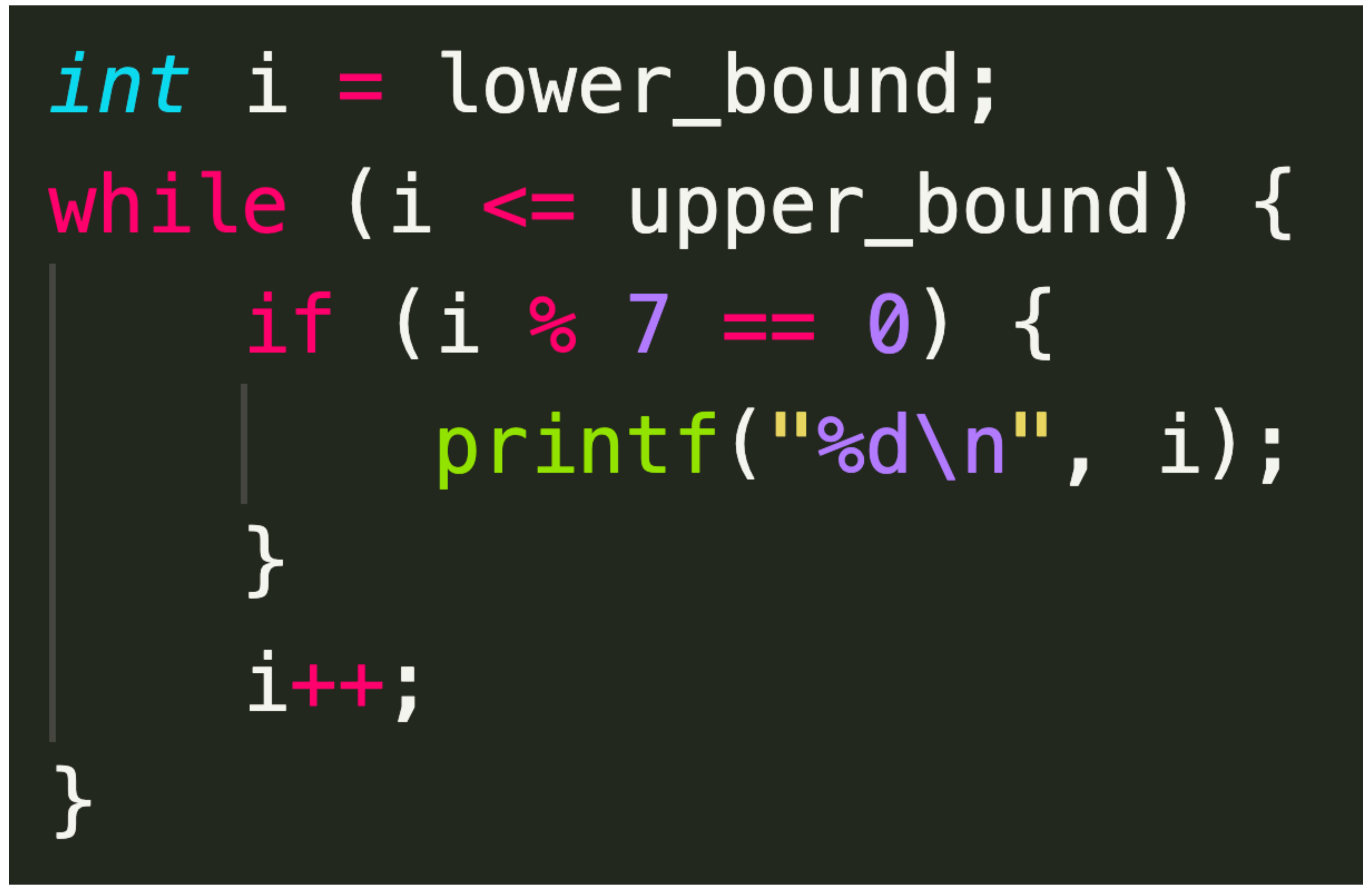
```
int height, width;
scanf("%d %d", &width, &height);
while (width < 0 || height < 0) {
    printf("Enter positive values: ");
    scanf("%d %d", &width, &height);
}
```

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

if statements and loops

Putting them together

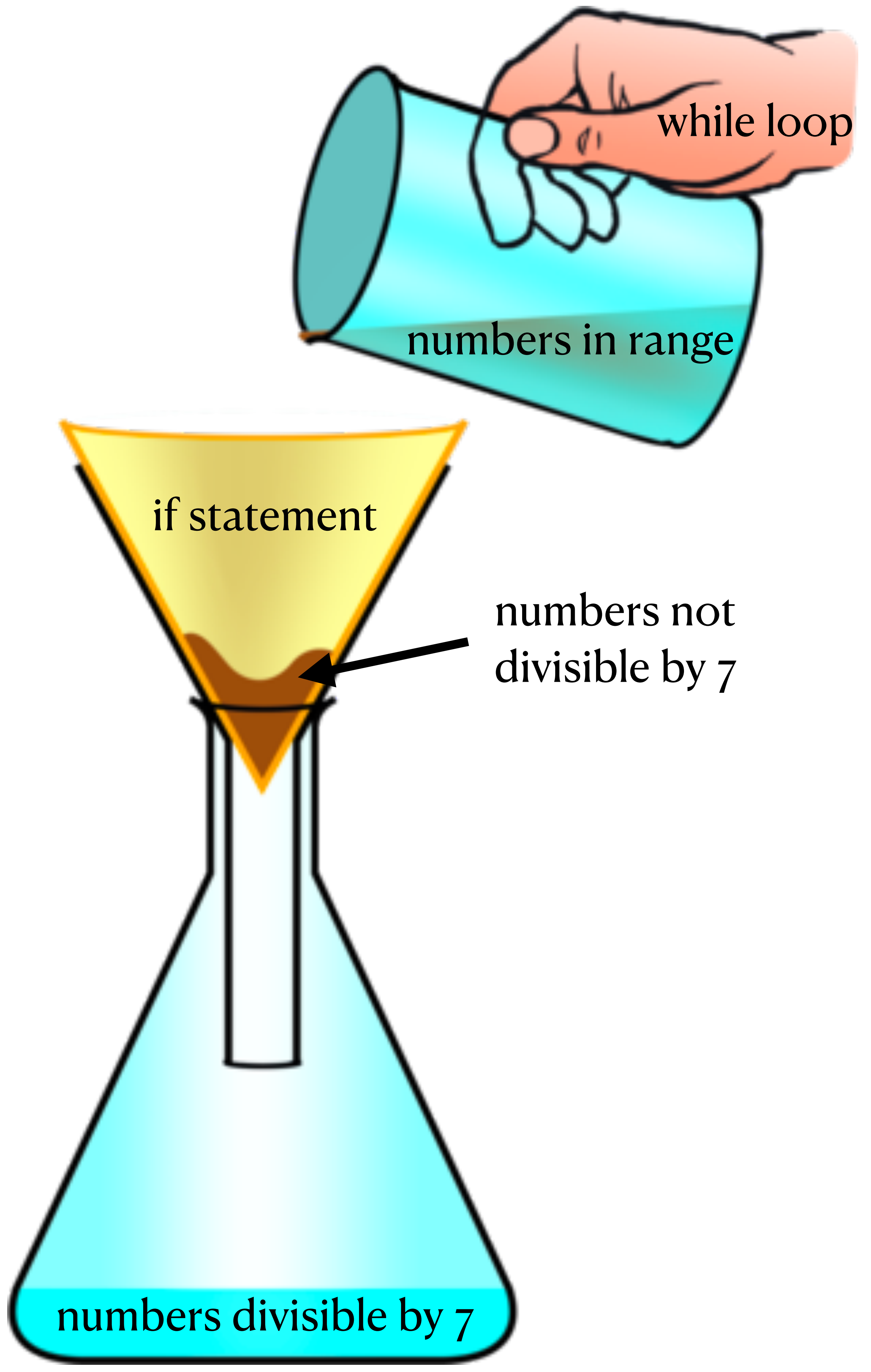
- Suppose we wanted to print only numbers divisible by 7 in a range of numbers. We'd write:



```
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.

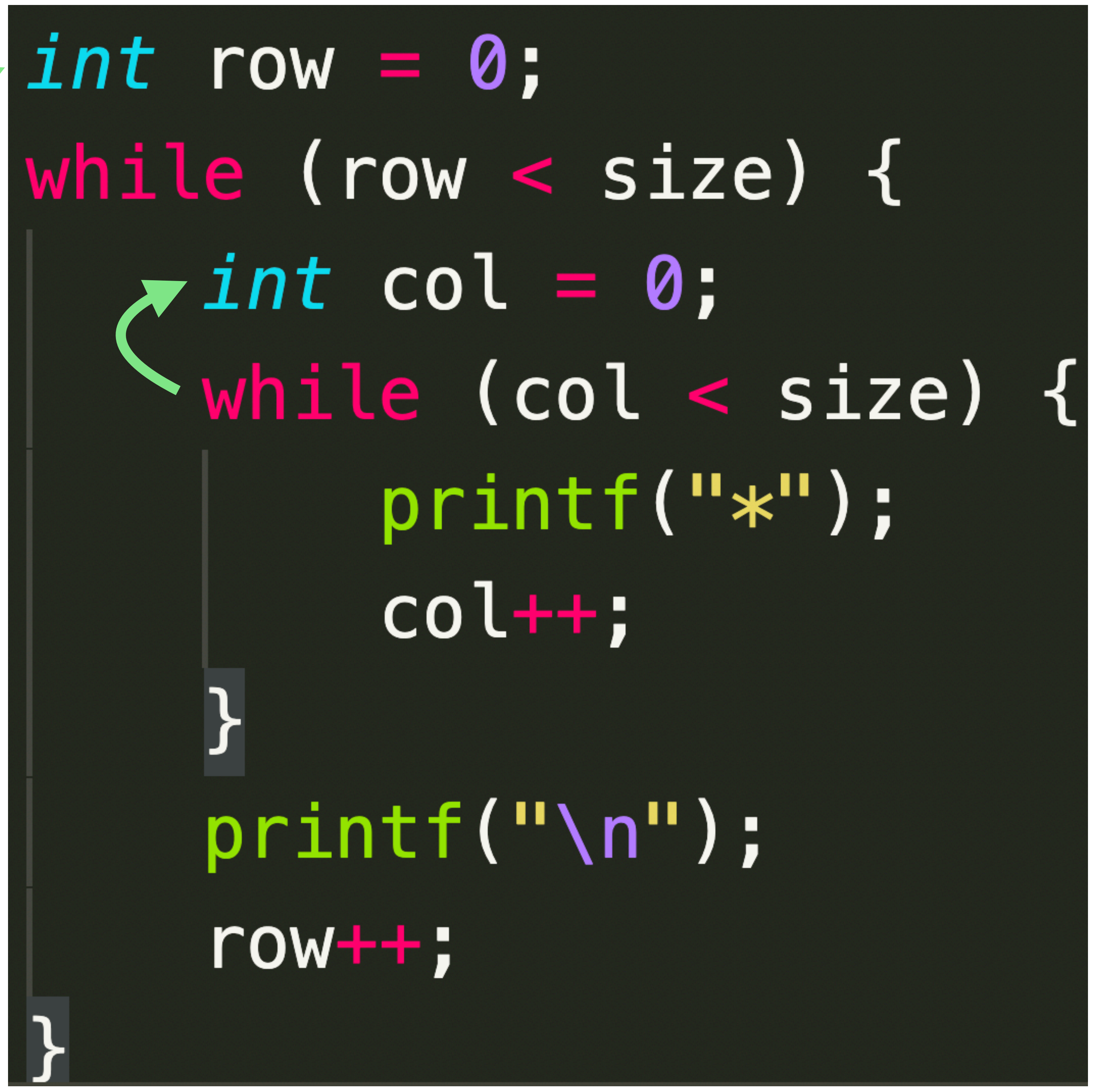
Nesting While Loops

- We sometimes need to put while loops inside while loops.
- For example, printing a ‘square’ of asterisks of given size.
- Which one of these will print properly?

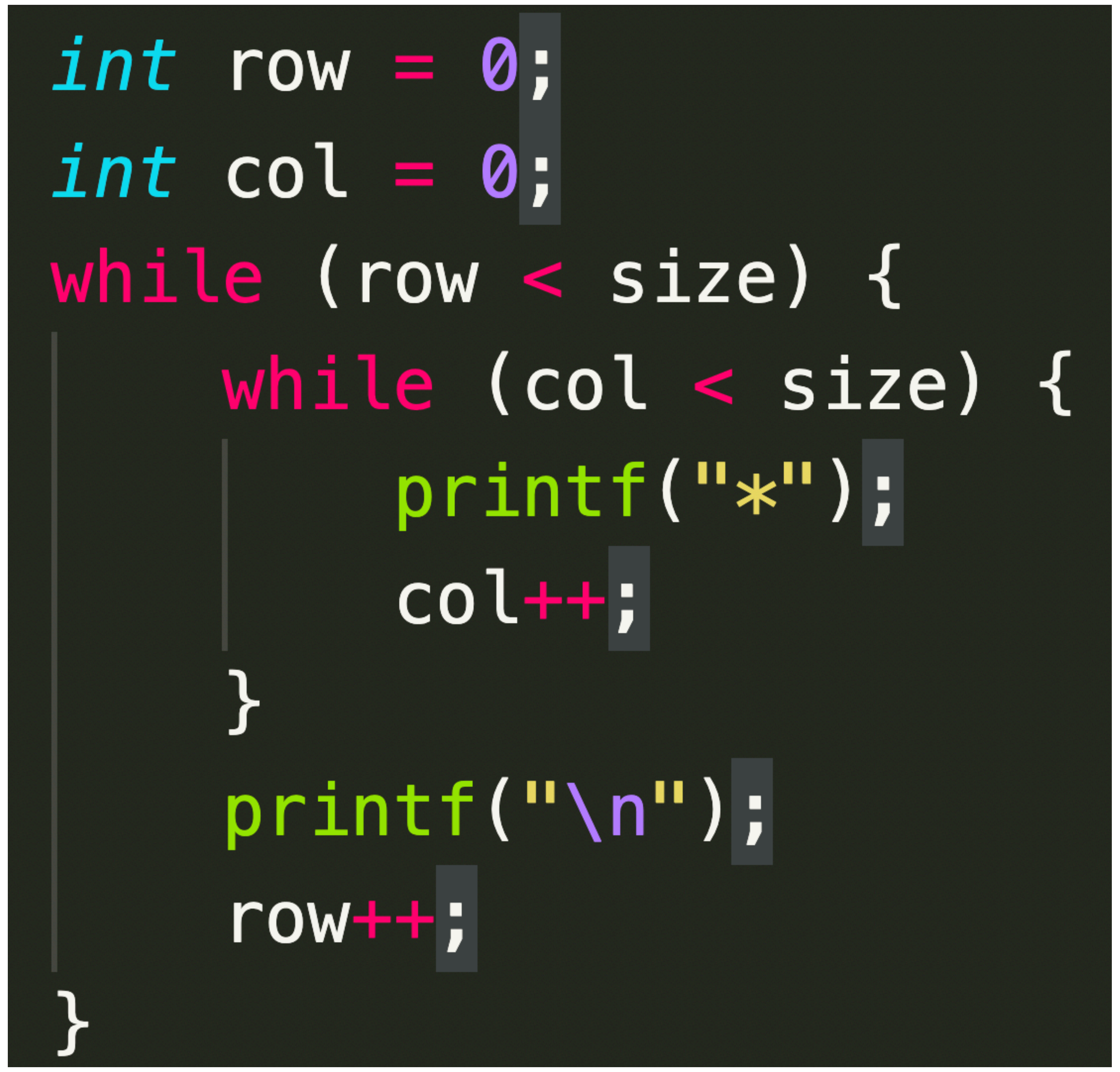
Sample output when size is 5:

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

Notice how the counter variables are declared directly above the while loop they correspond to



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



```c
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?