COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 4

Loop the loop

LAST LECTURE...

ON MONDAY

- Basic IF statements
- Conditionals - running our code based on some sort of condition being met
- More complex IF statements
- Catching scanf errors with IF statements
- While loops
    - Conditional

IN THIS LECTURE

TODAY...

- Refresh
- While loops
- A loop inside a loop
- Custom data types:
  - Structs
  - Enums

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/26T1/LIVE/WEEK02/

REFRESHER

IFS AND LOOPS
OH MY!

- Tea or coffee?
- Keep drinking tea until you ask for coffee

WHILE

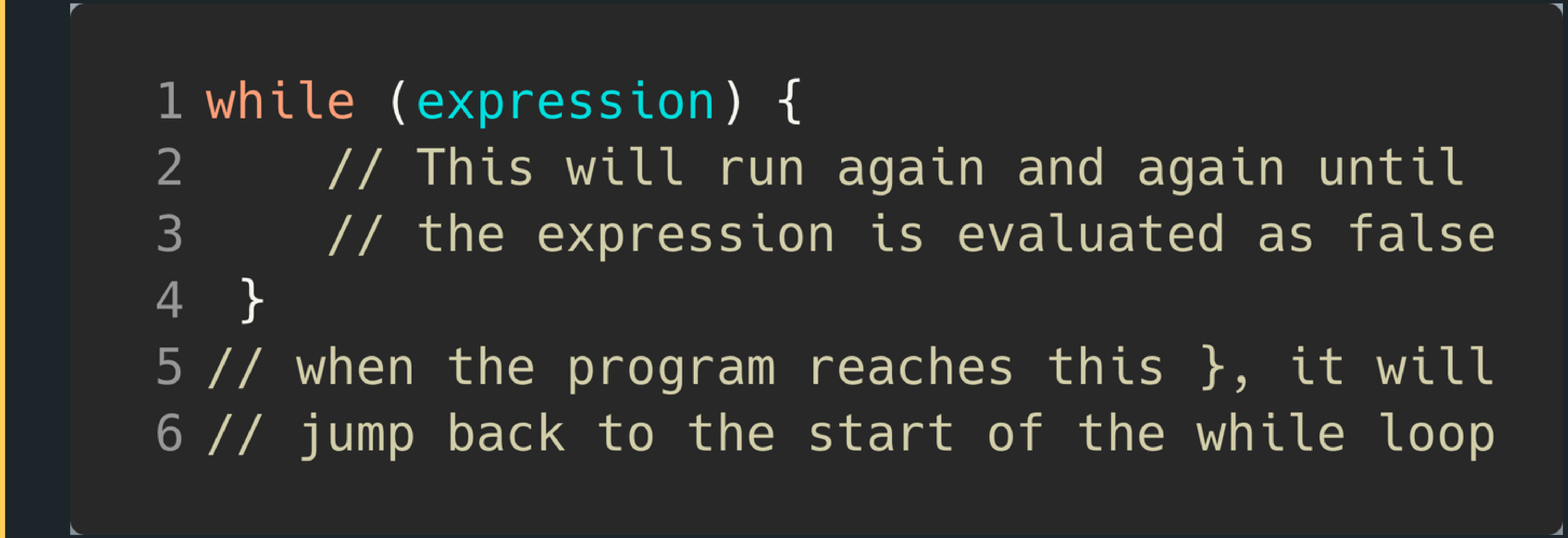
REPETITIVE TASKS SHOULDN’T REQUIRE REPETITIVE CODING

- C normally executes in order, line by line (starting with the main function after any # commands have been executed)
  - if statements allow us to “turn on or off” parts of our code
  - But up until now, we dont have a way to repeat code
- Copy-pasting the same code again and again is not a feasible solution
- Let’s see an example where it is inefficient to copy and paste code...

WHILE

WHILE SOMETHING IS TRUE, DO SOMETHING

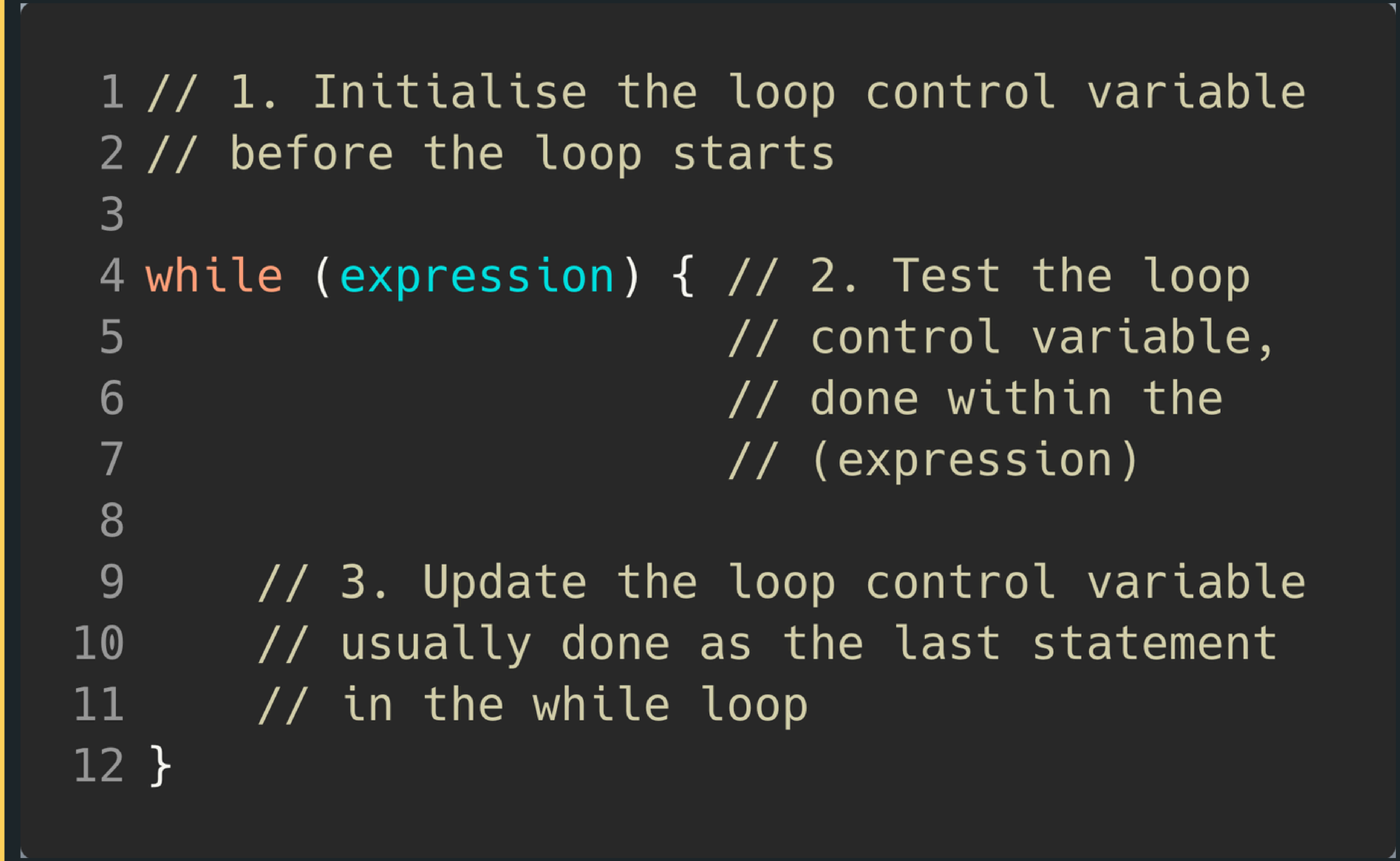
- **`while()`** loops - can commonly be controlled in three ways:
    - Count loops
    - Sentinel loops
    - Conditional loops



```
while (expression) {
    // This will run again and again until
    // the expression is evaluated as false
 }
// when the program reaches this }, it will
// jump back to the start of the while loop
```

WHILE

CONTROL THE WHILE LOOP

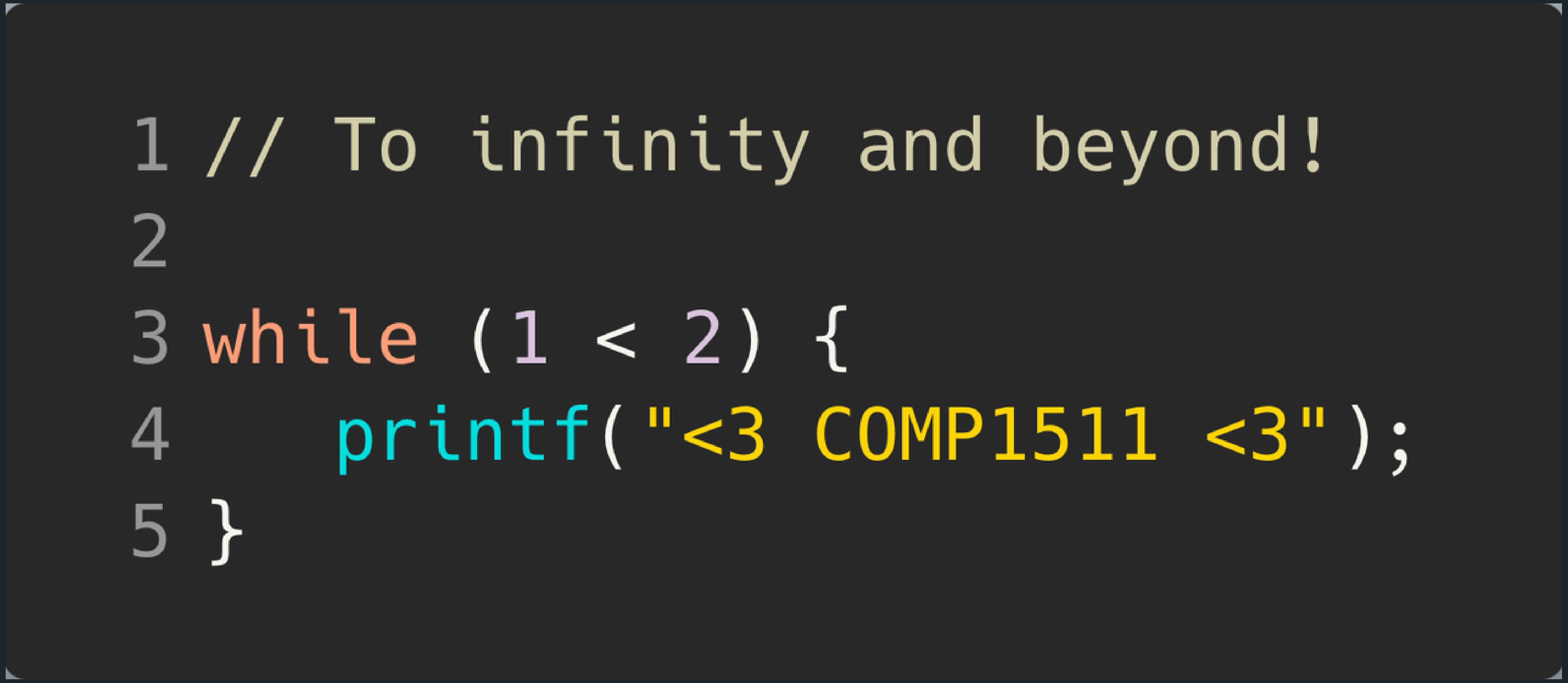


```
// 1. Initialise the loop control variable
// before the loop starts

while (expression) { // 2. Test the loop
                     // control variable,
                     // done within the
                     // (expression)

    // 3. Update the loop control variable
    // usually done as the last statement
    // in the while loop
}
```

TO INFINITY AND BEYOND

TERMINATING YOUR LOOP

- It's actually very easy to make a program that goes forever
- Consider the following while loop:

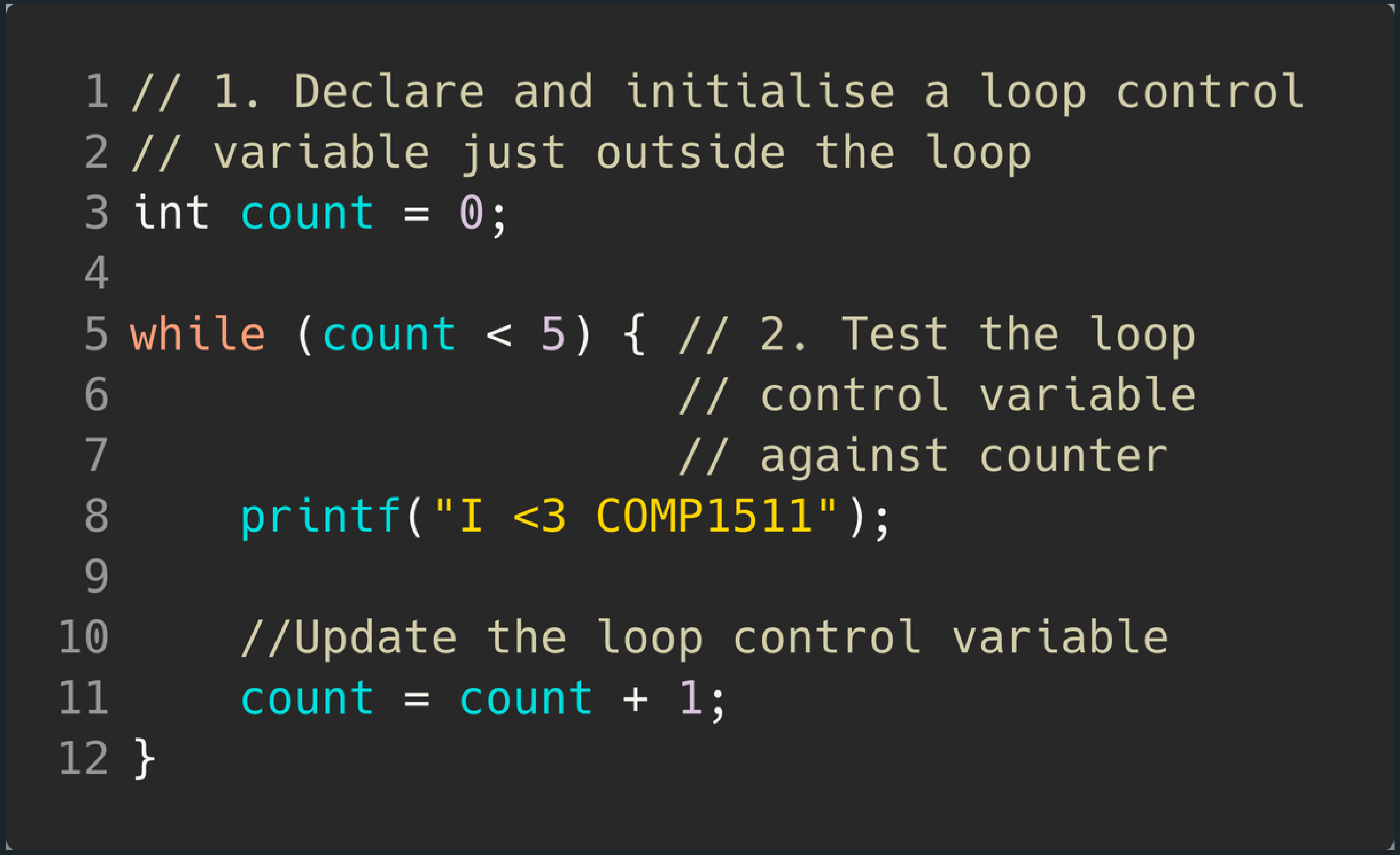


```c
// To infinity and beyond!

while (1 < 2) {
    printf("<3 COMP1511 <3");
}
```

CONTROL THE WHILE LOOP

COUNT LOOPS

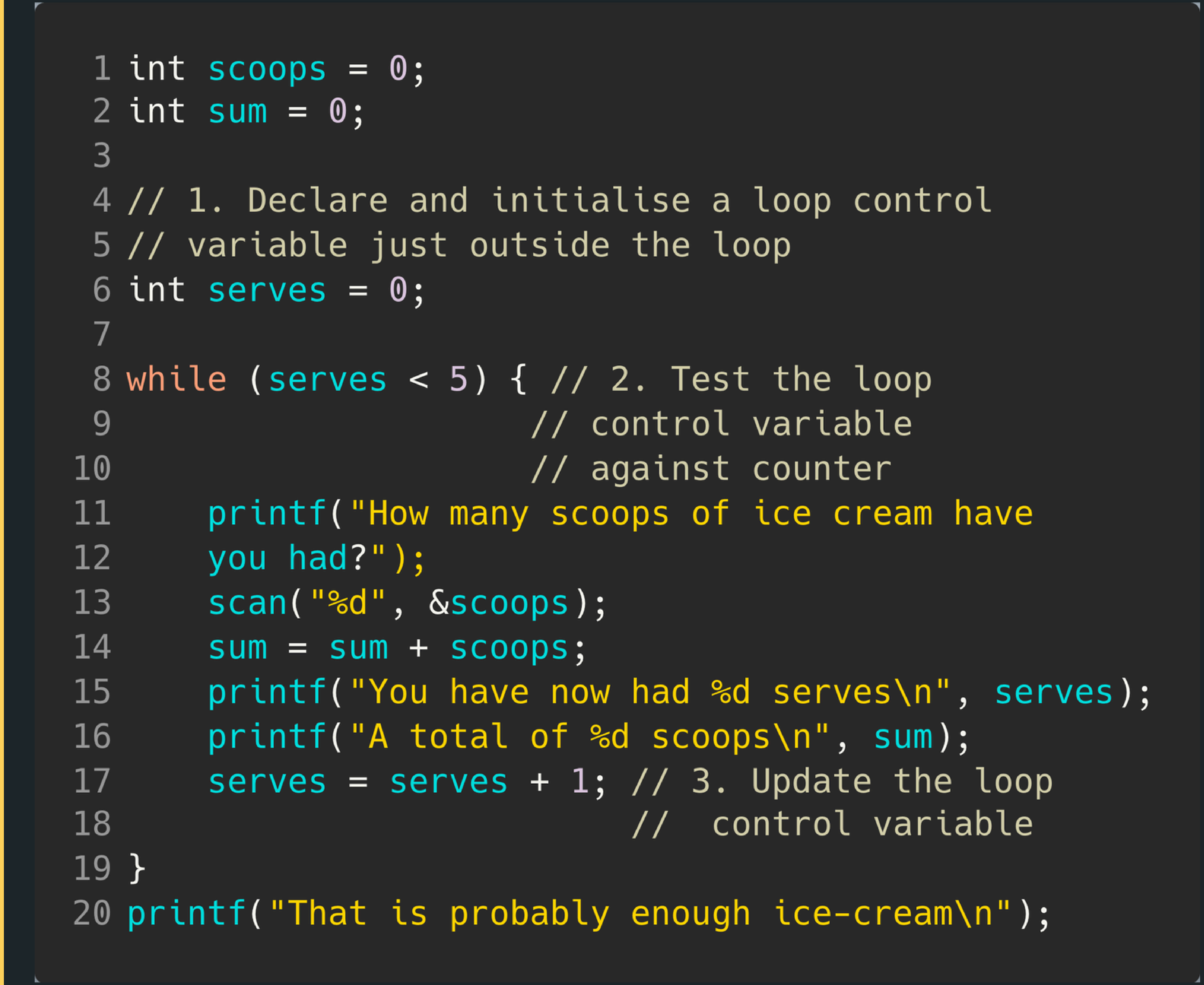
- Use a variable to control how many times a loop runs - a ”loop counter”
- It’s an **`int`** that’s declared outside the loop
- It’s “termination condition” can be checked in the while expression
- It will be updated inside the loop



```c
// 1. Declare and initialise a loop control
// variable just outside the loop
int count = 0;

while (count < 5) { // 2. Test the loop
                    // control variable
                    // against counter
    printf("I <3 COMP1511");

    //Update the loop control variable
    count = count + 1;
}
```

CONTROL THE WHILE LOOP

COUNT LOOPS



```
int scoops = 0;
int sum = 0;

// 1. Declare and initialise a loop control
// variable just outside the loop
int serves = 0;

while (serves < 5) { // 2. Test the loop
                    // control variable
                    // against counter
    printf("How many scoops of ice cream have
    you had?");
    scan("%d", &scoops);
    sum = sum + scoops;
    printf("You have now had %d serves\n", serves);
    printf("A total of %d scoops\n", sum);
    serves = serves + 1; // 3. Update the loop
                         //  control variable
}
printf("That is probably enough ice-cream\n");
```

SENTINEL VALUES

WHAT IS A SENTINEL?

- When we use a loop counter, we assume that we know how many times we need to repeat something
- Consider a situation where you don't know the number of repetitions required, but you need to repeat whilst there is valid data
- A sentinel value is a 'flag value', it tells the loop when it can stop…
- For example, keep scanning in numbers until an odd number is encountered
  - We do not know how many numbers we will have to scan before this happens
  - We know that we can stop when we see an odd number

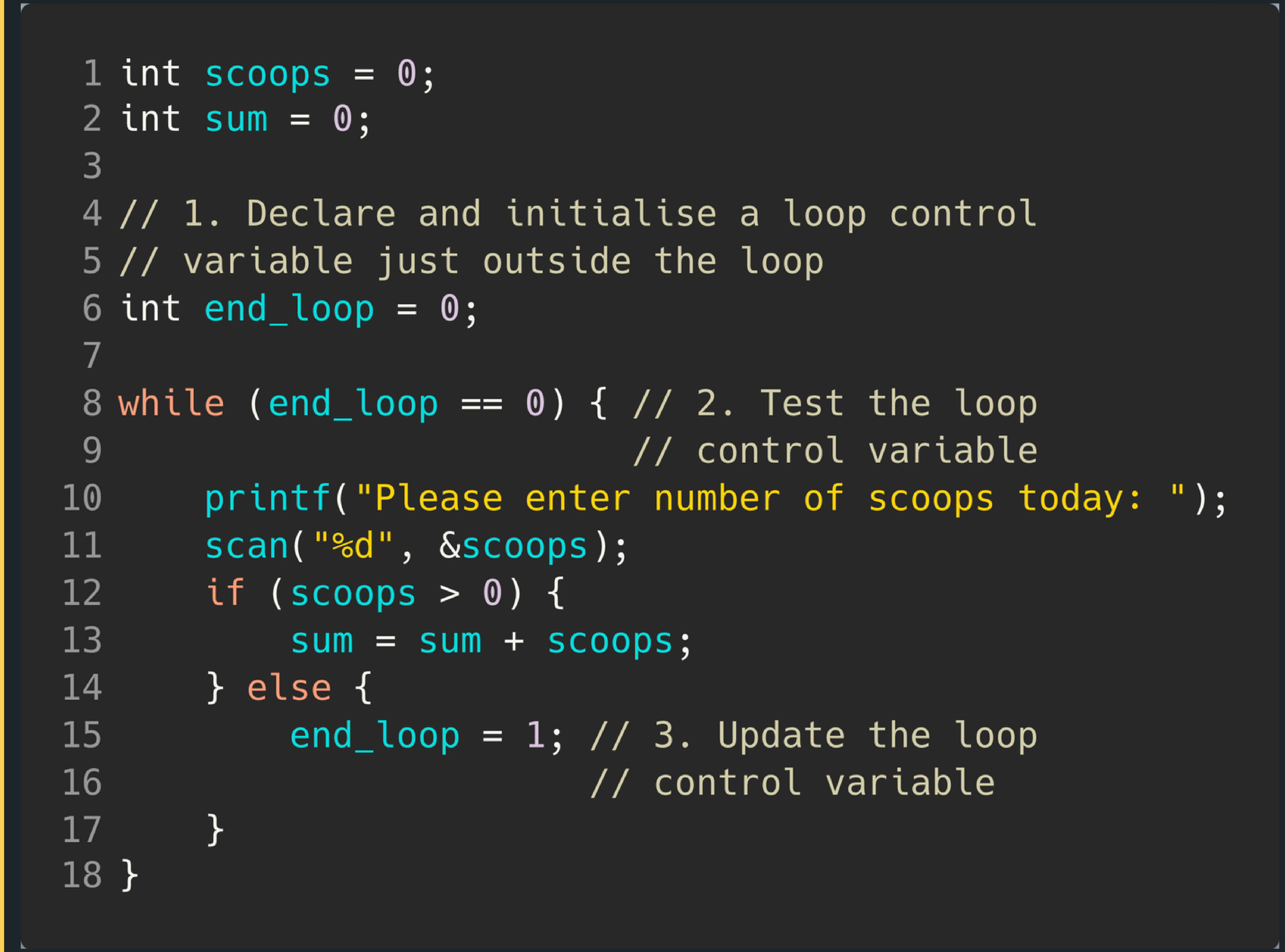
CONTROL THE WHILE LOOP

SENTINEL LOOPS

- Sentinel Loops: can also use a variable to decide to exit a loop at any time
- We call this variable a ”sentinel”
- It’s like an on/off switch for the loop
- It is declared and set outside the loop
- It’s “termination condition” can be checked in the while expression
- It will be updated inside the loop (often attached to a decision statement)

CONTROL THE WHILE LOOP

SENTINEL LOOPS



```c
int scoops = 0;
int sum = 0;

// 1. Declare and initialise a loop control
// variable just outside the loop
int end_loop = 0;

while (end_loop == 0) { // 2. Test the loop
                        // control variable
    printf("Please enter number of scoops today: ");
    scan("%d", &scoops);
    if (scoops > 0) {
        sum = sum + scoops;
    } else {
        end_loop = 1; // 3. Update the loop
                      // control variable
    }
}
```

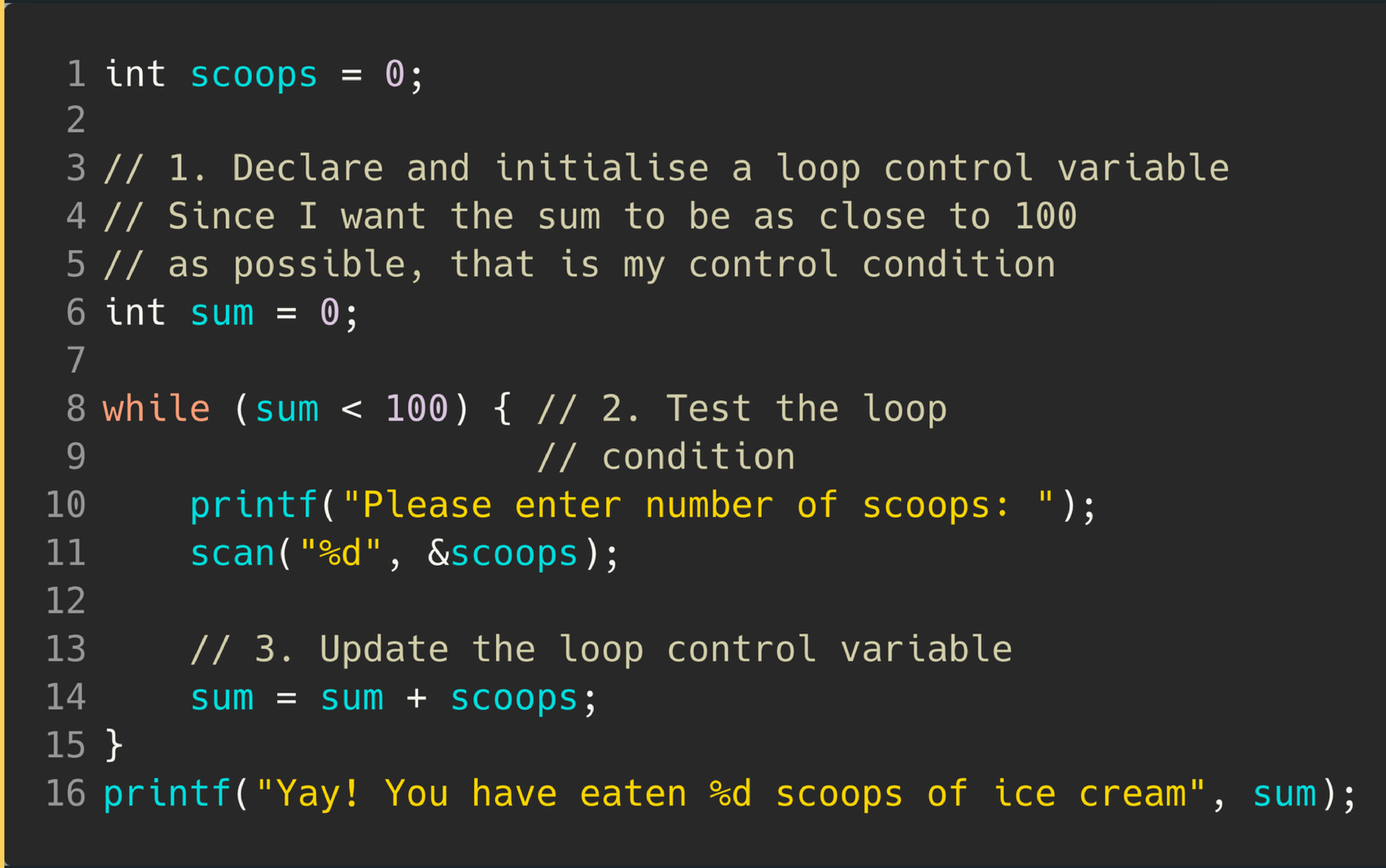
CONTROL THE WHILE LOOP

CONDITIONAL LOOPS

- Conditional Loops: can also use a condition to decide to exit a loop at any time
- This is called conditional looping
- Also do not know how many times we may need to repeat.
- We will terminate as a result of some type of calculation

CONTROL THE WHILE LOOP

COUNT LOOPS



```c
int scoops = 0;

// 1. Declare and initialise a loop control variable
// Since I want the sum to be as close to 100
// as possible, that is my control condition
int sum = 0;

while (sum < 100) { // 2. Test the loop
                    // condition
    printf("Please enter number of scoops: ");
    scan("%d", &scoops);

    // 3. Update the loop control variable
    sum = sum + scoops;
}
printf("Yay! You have eaten %d scoops of ice cream", sum);
```

ACTION TIME

CODE DEMO

- While loop with a counter:

`while_count.c`

- While loop with a sentinel:

`while_sentinel.c`

- While loop with a condition:

`while_condition.c`

WHILE INSIDE A WHILE

PUTTING A LOOP INSIDE A LOOP

- If we put a loop inside a loop . . .
- Each time a loop runs
- It runs the other loop
- The inside loop ends up running a LOT of times

PROBLEM TIME

PRINT OUT A GRID OF NUMBERS

- Print out a grid of numbers:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

- Break down the problem…
- Get it down to a component that you can do…

PROBLEM TIME

PRINT OUT A GRID OF NUMBERS

- Think of the grid not as numbers, but as 5 rows of bricks. To build it:
  - You must lay 5 bricks side-by-side to finish a row.
  - You must finish a row before you can start the next one below it.
- So we would start at Row 1.
- Lay brick 1, then 2, then 3, then 4, then 5.
- Now that you've hit 5, you have to physically move your body back to the start and up one level.
- Repeat: Go to Row 2 and do the exact same thing. Repeat until you have finished 5 rows.

PROBLEM TIME

PRINT OUT A PYRAMID OF NUMBERS

- What if we now print out a half pyramid of numbers:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

- Break down the problem…
- Get it down to a component that you can do…

ACTION TIME

CODE DEMO

- While loop print a grid:
  `grid.c`
- While loop print a pyramid:
  `pyramid.c`

ORGANISING DIFFERENT TYPES INTO ONE RELATED WHOLE

USER DEFINED DATA TYPE `struct`

- Structures…. Or **`struct`** (as they are known in C!)
- Structs (short for structures) are a way to create custom variables
- Structs are variables that are made up of other variables

STRUCTURES

WHAT? WHY? EXAMPLES?

- What happens if you wanted to group some variables together to make a single structure?
- Why do we need structures?
    - Helps us to organise related but different components into one structure
    - Useful in defining real life problems
- What are some examples in real life where some things go together to make a single component?

HOW DO WE CREATE A STRUCT?

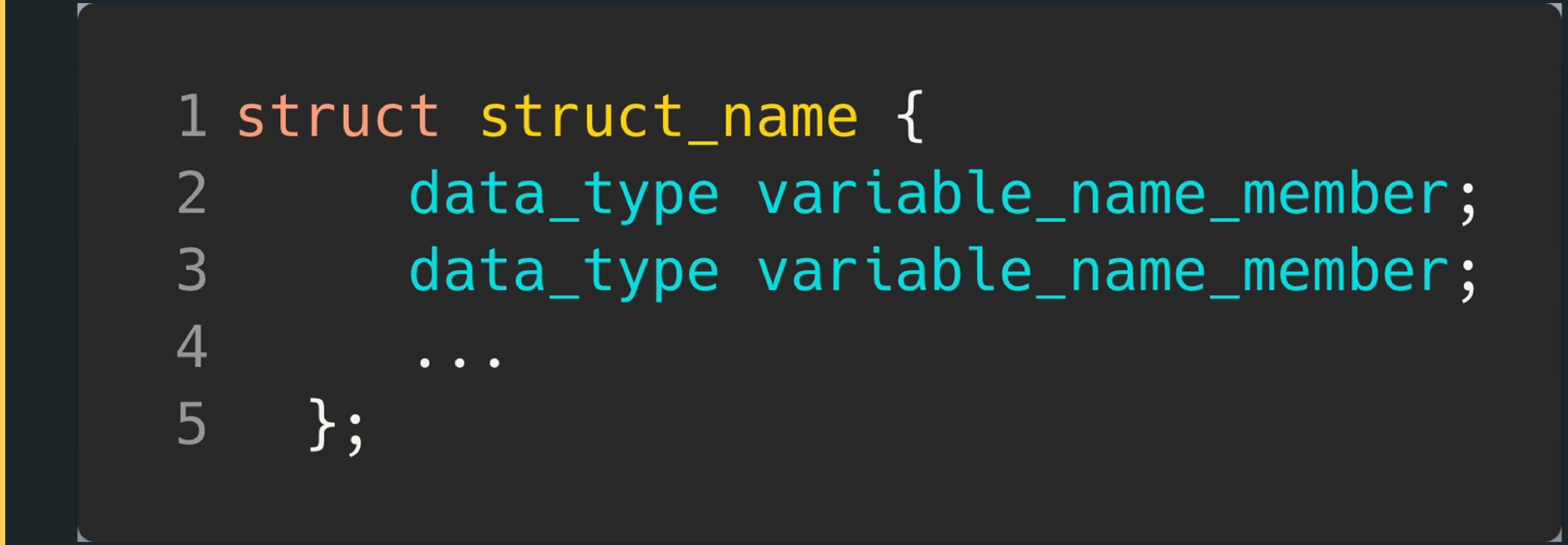
To create a struct, there are three steps:

1. Define the struct (outside the main)
2. Declare the struct (inside your main)
3. Initialise the struct (inside your main)

1. DEFINING A STRUCT

WHAT AM I GROUPING TOGETHER INTO ONE WHOLE? LET'S USE AN EXAMPLE OF A COORDINATE POINT

Because structures are a variable that we have created, made up of components that we decided belong together, we need to define what the struct (or structure is). To define a struct, we define it before our main function and use some special syntax.

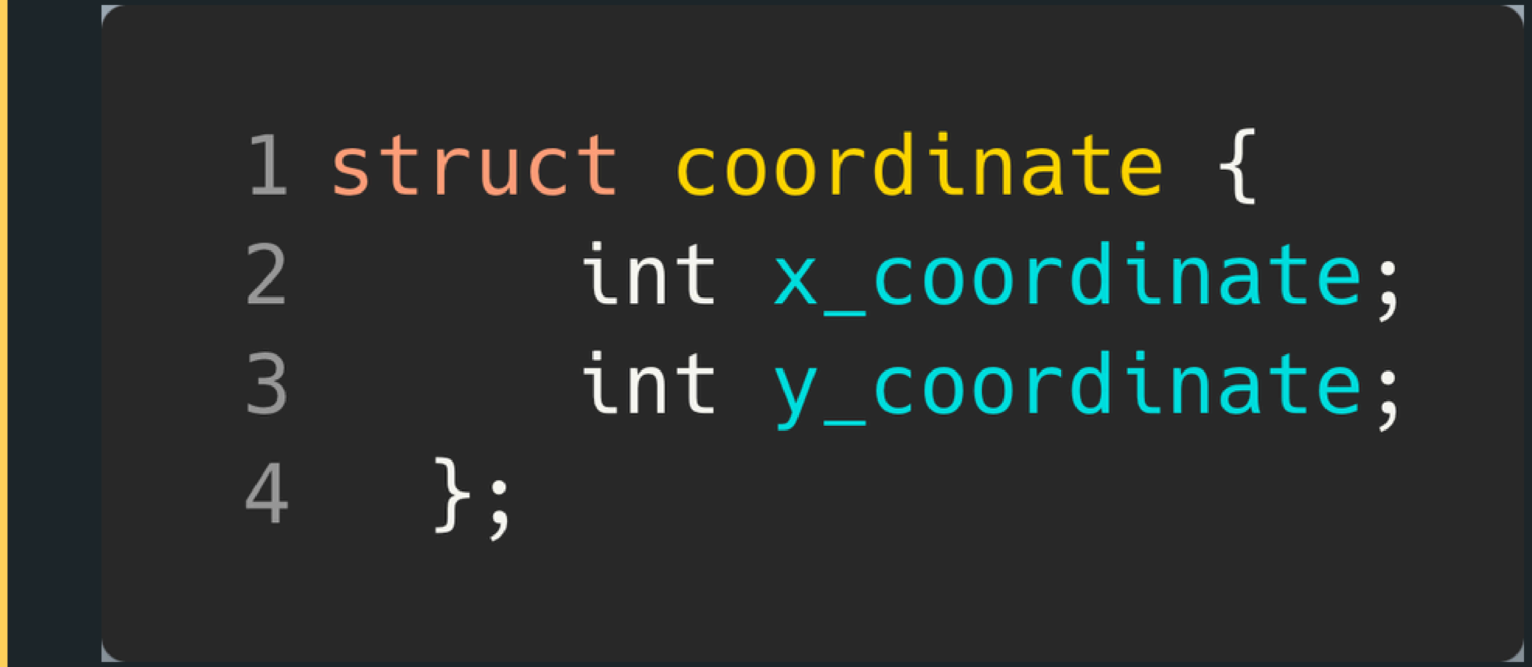


```
struct struct_name {
        data_type variable_name_member;
        data_type variable_name_member;
        ...
  };
```

1. DEFINING A STRUCT

WHAT AM I GROUPING TOGETHER INTO ONE WHOLE? LET'S USE AN EXAMPLE OF A COORDINATE POINT

For example, using the coordinate point example, to make a structure called coordinate, that has two members - the x_coordinate and the y_coordinate:

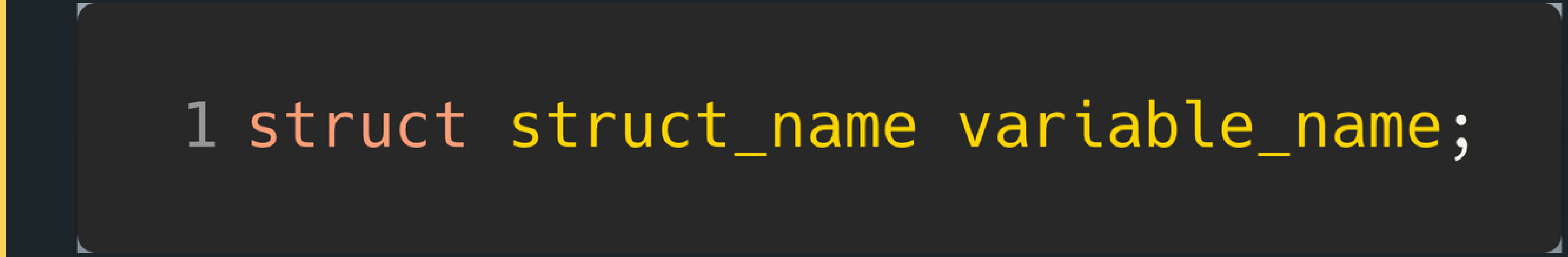


```
struct coordinate {
     int x_coordinate;
     int y_coordinate;
  };
```

2. DECLARING A STRUCT

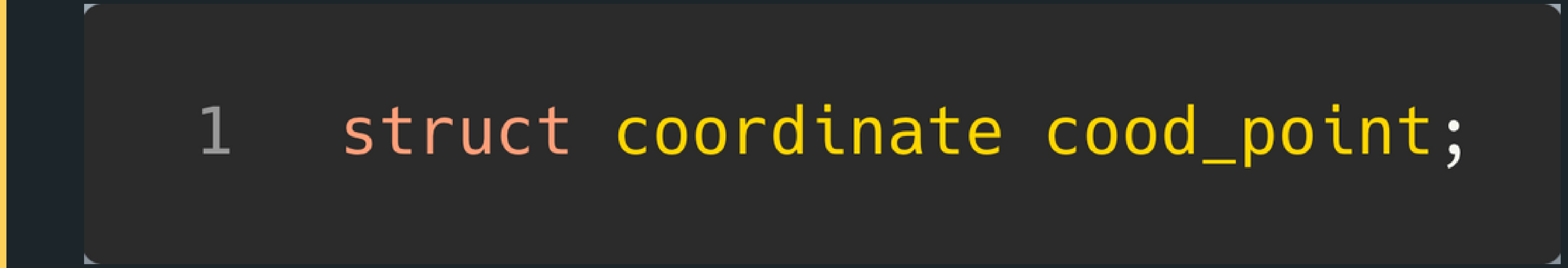
INSIDE YOUR MAIN

To declare a struct, inside the main function (or wherever you are using the structure - more on this later)...



```
struct struct_name variable_name;
```

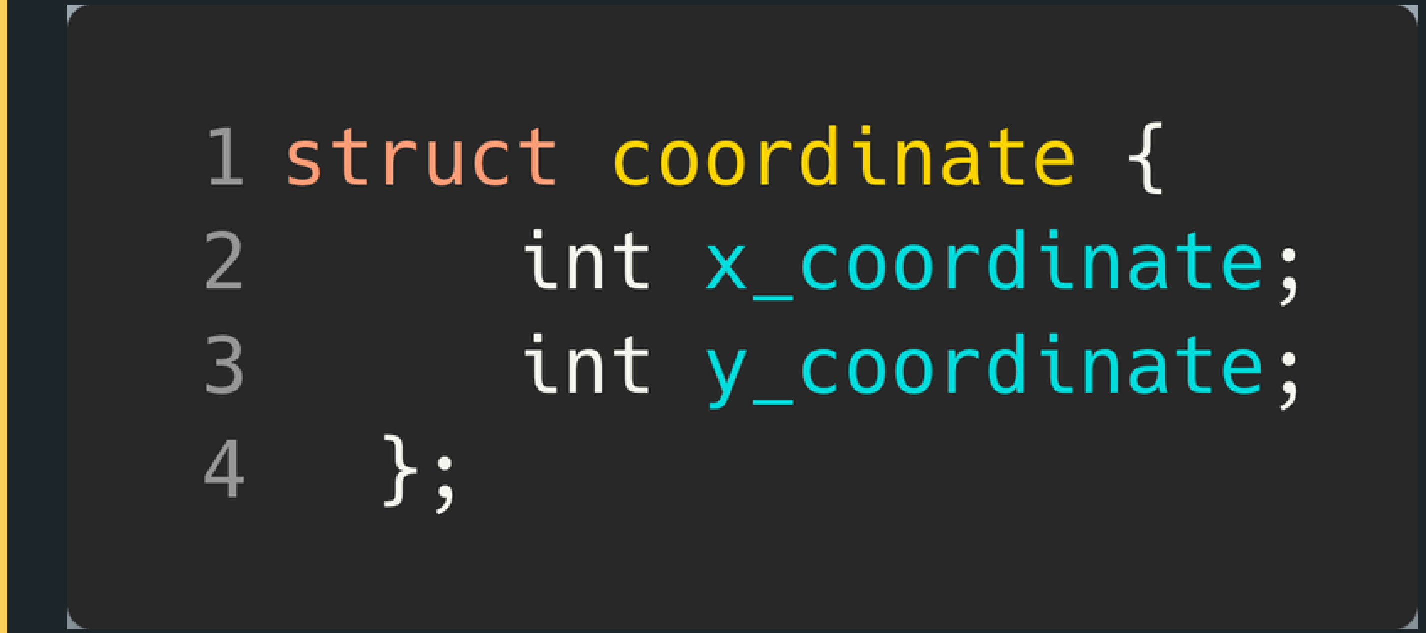
For example, using the coordinate point example, to declare a variable, cood_point, of type struct coordinate



```
  struct coordinate cood_point;
```

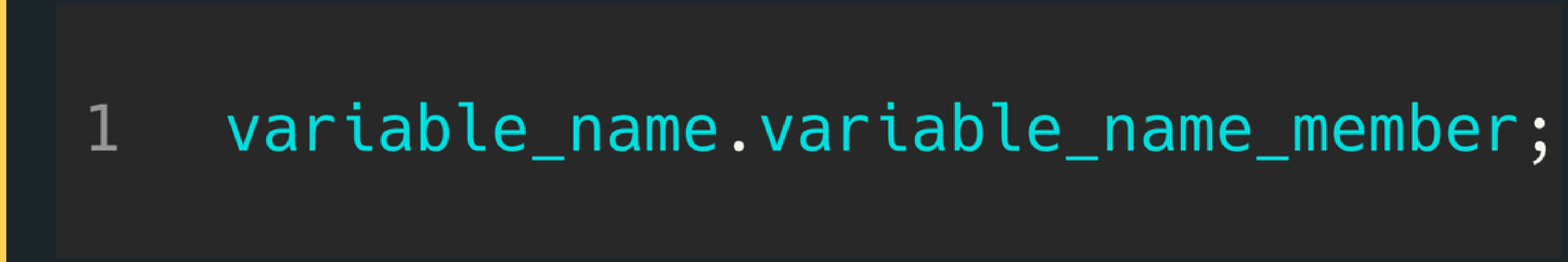
3.INITIALISE A STRUCT

INSIDE YOUR MAIN



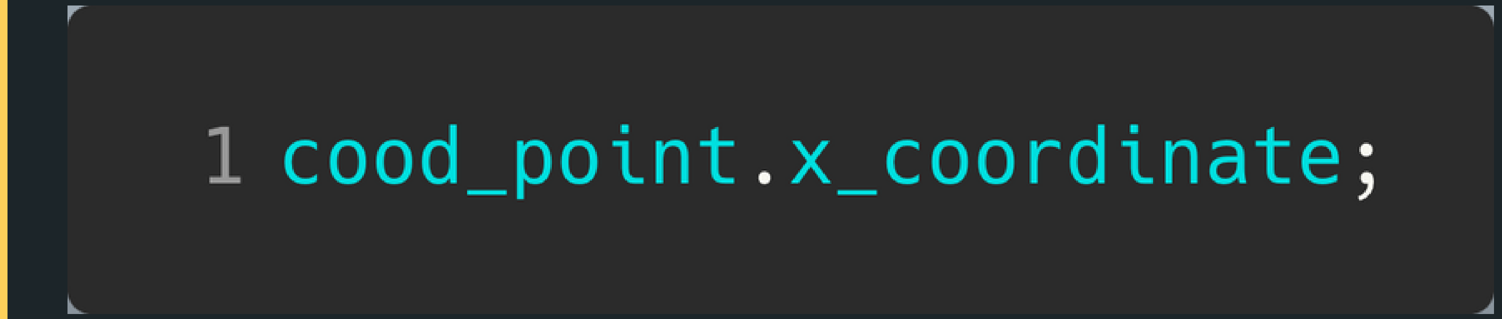
```
struct coordinate {
     int x_coordinate;
     int y_coordinate;
  };
```

We access a member by using the dot operator .



```
variable_name.variable_name_member;
```

For example, using the coordinate point example, with variable name: cood_point, trying to access the x coordinate:



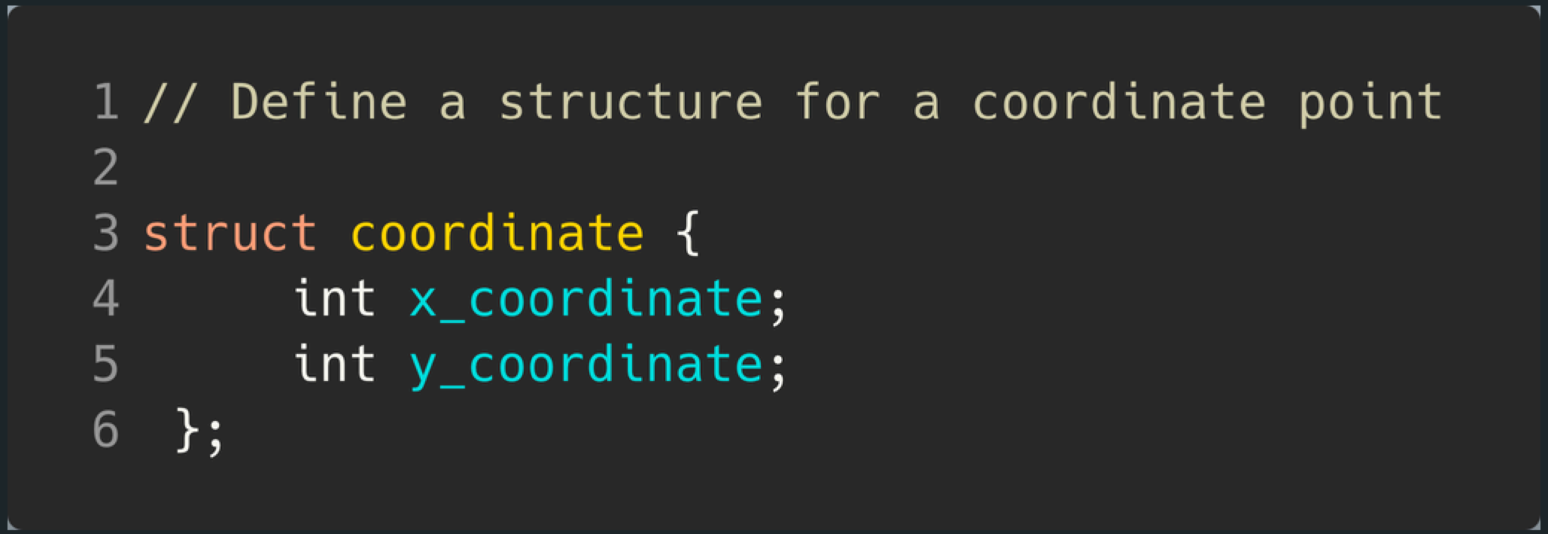
```
cood_point.x_coordinate;
```

LET'S SEE IT ALL TOGETHER FOR A COORDINATE POINT

1. DEFINE
2. DECLARE
3. INITIALISE

1. DEFINE

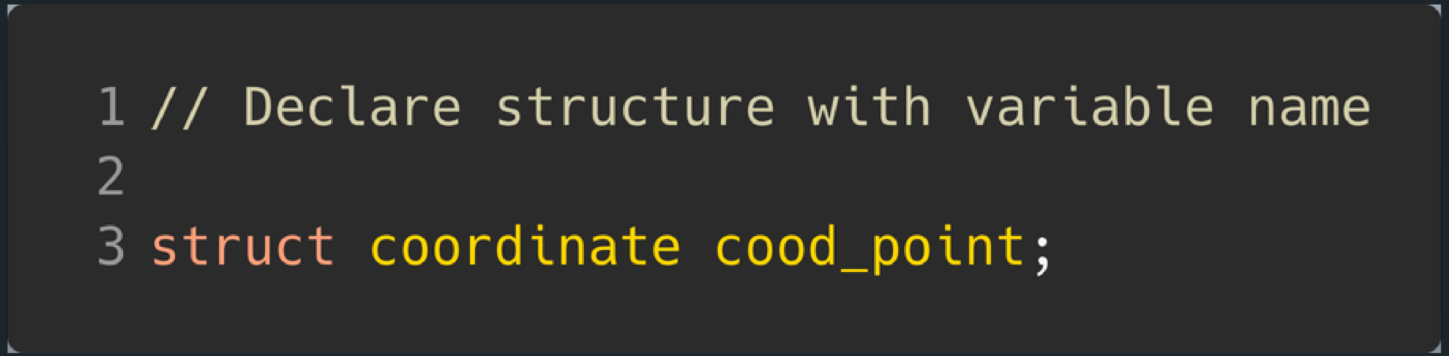
Inside the main function



```
// Define a structure for a coordinate point

struct coordinate {
     int x_coordinate;
     int y_coordinate;
 };
```

2. DECLARE

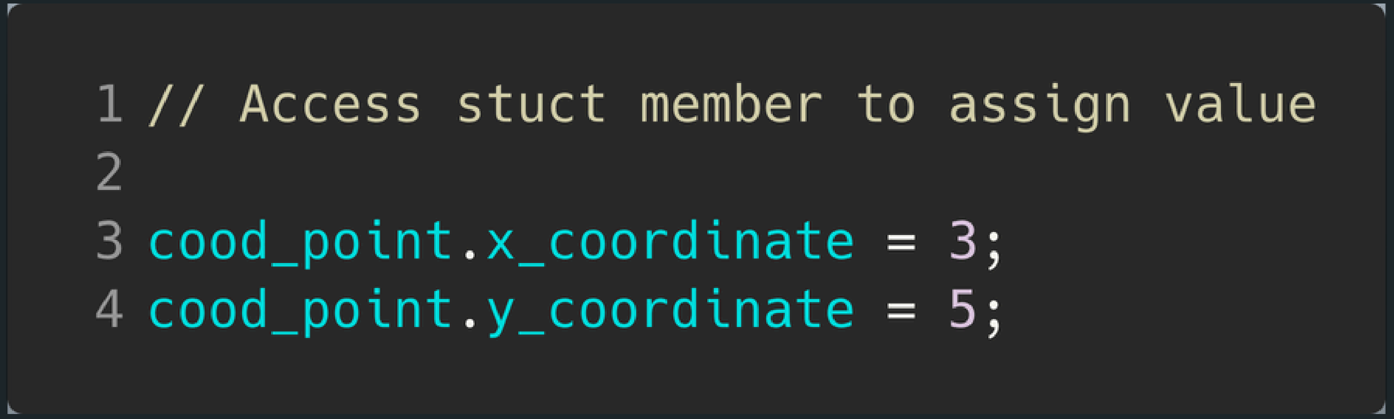
Inside the main function



```
// Declare structure with variable name

struct coordinate cood_point;
```

3. INITIALISE

Inside the main function

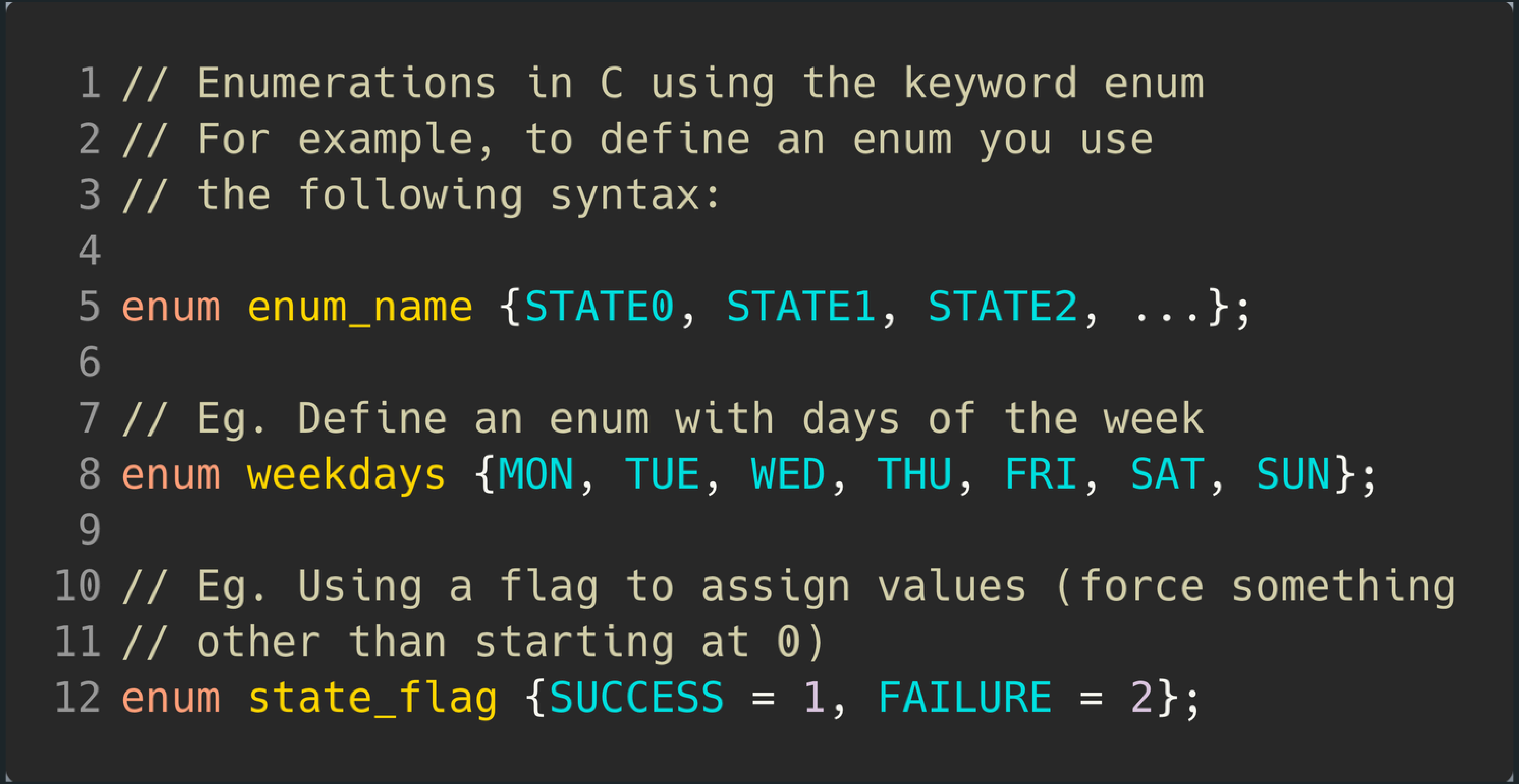


```
// Access stuct member to assign value

cood_point.x_coordinate = 3;
cood_point.y_coordinate = 5;
```

ENUMERATIONS

USER DEFINED DATA TYPE `enum`

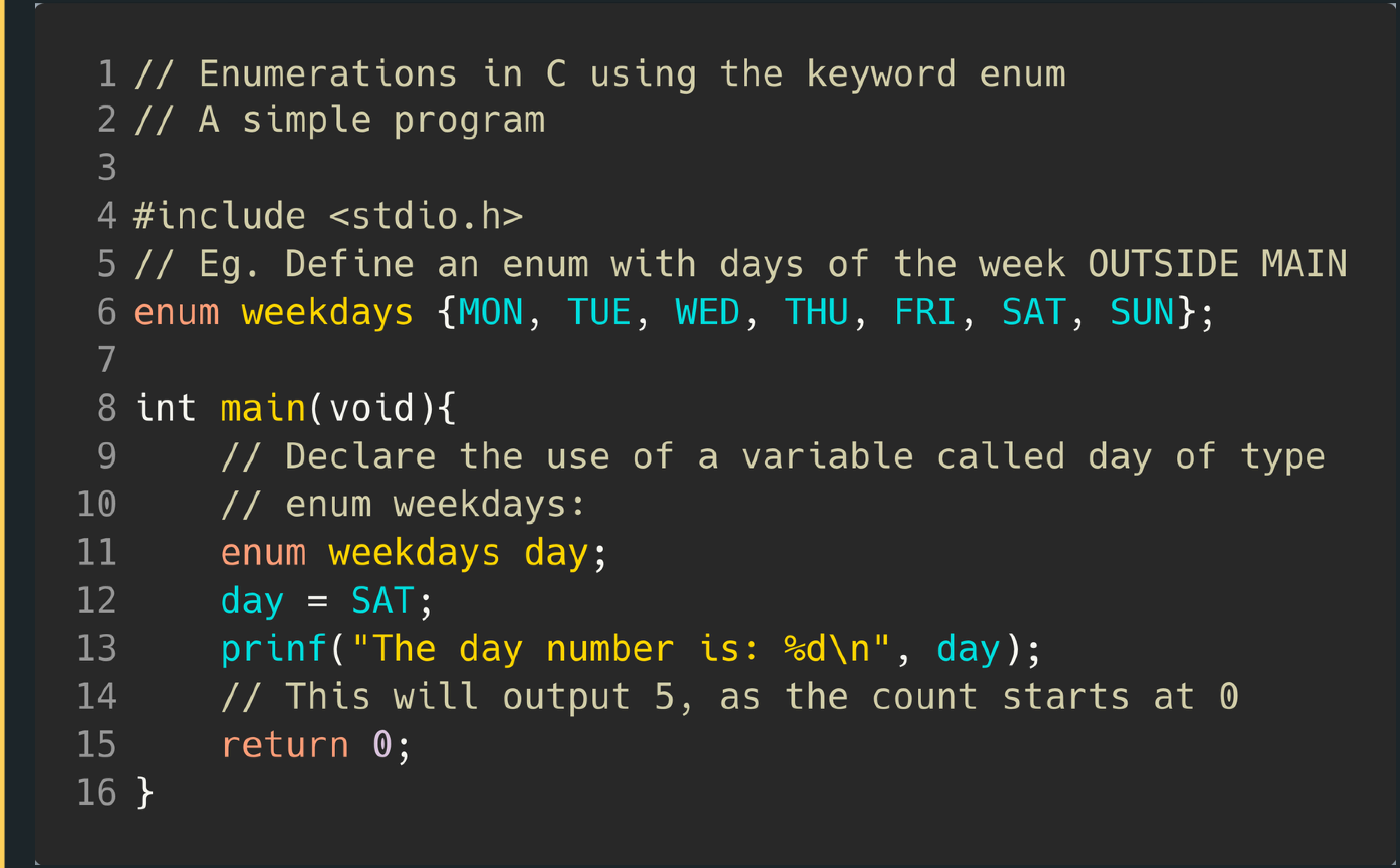
- Integer data types that you create with a limited range of values (enumerated constants)
- Used to assign names to integral constants
    - the names make the program easier to read and maintain



```
// Enumerations in C using the keyword enum
// For example, to define an enum you use
// the following syntax:

enum enum_name {STATE0, STATE1, STATE2, ...};

// Eg. Define an enum with days of the week
enum weekdays {MON, TUE, WED, THU, FRI, SAT, SUN};

// Eg. Using a flag to assign values (force something
// other than starting at 0)
enum state_flag {SUCCESS = 1, FAILURE = 2};
```

ENUMERATIONS

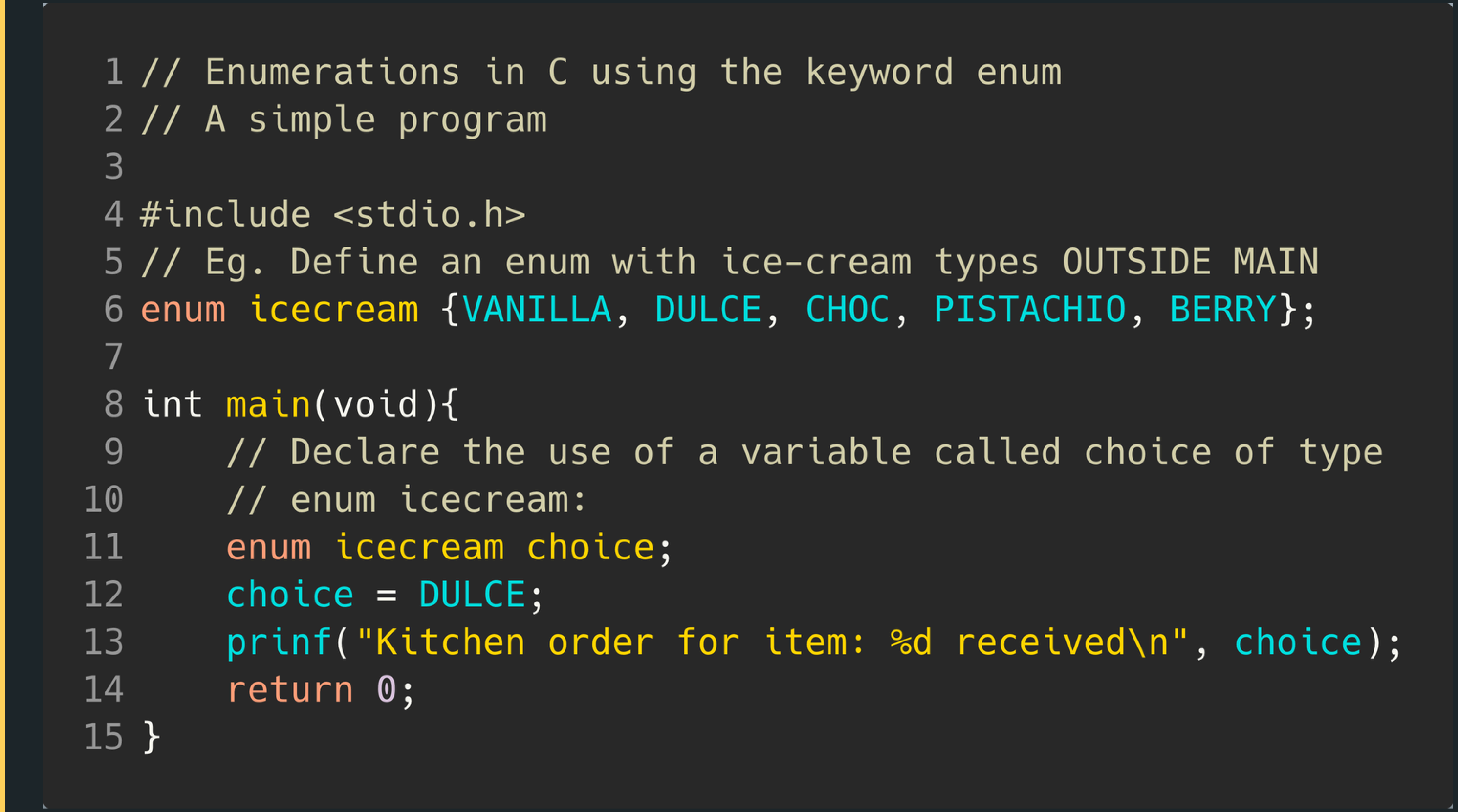
USER DEFINED DATA TYPE `enum`



```c
// Enumerations in C using the keyword enum
// A simple program

#include <stdio.h>
// Eg. Define an enum with days of the week OUTSIDE MAIN
enum weekdays {MON, TUE, WED, THU, FRI, SAT, SUN};

int main(void){
    // Declare the use of a variable called day of type
    // enum weekdays:
    enum weekdays day;
    day = SAT;
    prinf("The day number is: %d\n", day);
    // This will output 5, as the count starts at 0
    return 0;
}
```

ENUMERATIONS

FOR EXAMPLE USING MENU ITEMS, IMAGINE IF AN ICE CREAM SHOP HAD 57 FLAVOURS!



```c
// Enumerations in C using the keyword enum
// A simple program

#include <stdio.h>
// Eg. Define an enum with ice-cream types OUTSIDE MAIN
enum icecream {VANILLA, DULCE, CHOC, PISTACHIO, BERRY};

int main(void){
    // Declare the use of a variable called choice of type
    // enum icecream:
    enum icecream choice;
    choice = DULCE;
    prinf("Kitchen order for item: %d received\n", choice);
    return 0;
}
```

WHY ENUMS?

`enum vs #define`

- The advantages of using enums over #defines:
  - Enumerations follow scope rules:
    - You cannot have an enum state that is the same in two different types of enums
  - Enumerations are automatically assigned values, which makes the code easier to read
    - Think of the case where you have a large number of constants (error codes for example!?)
  - We use enums when we want a variable to have a specific set of values

WHAT DID WE LEARN TODAY?

LOOP THE LOOP WHILE (COUNTER)

while_counter.c

LOOP THE LOOP WHILE (SENTINEL)

while_sentinel.c

LOOP THE LOOP WHILE (CONDITION)

while_condition.c

LOOP INSIDE A LOOP (CAN'T GET ENOUGH OF A LOOP)

grid: grid.c

pyramid: pyramid.c

WHAT DID WE LEARN TODAY?

STRUCTURES

struct.c

ENUMERATIONS

enum.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au