COMP1511 Week 5

2D Arrays and Pointers

Joanna Lin

What we'll cover this week

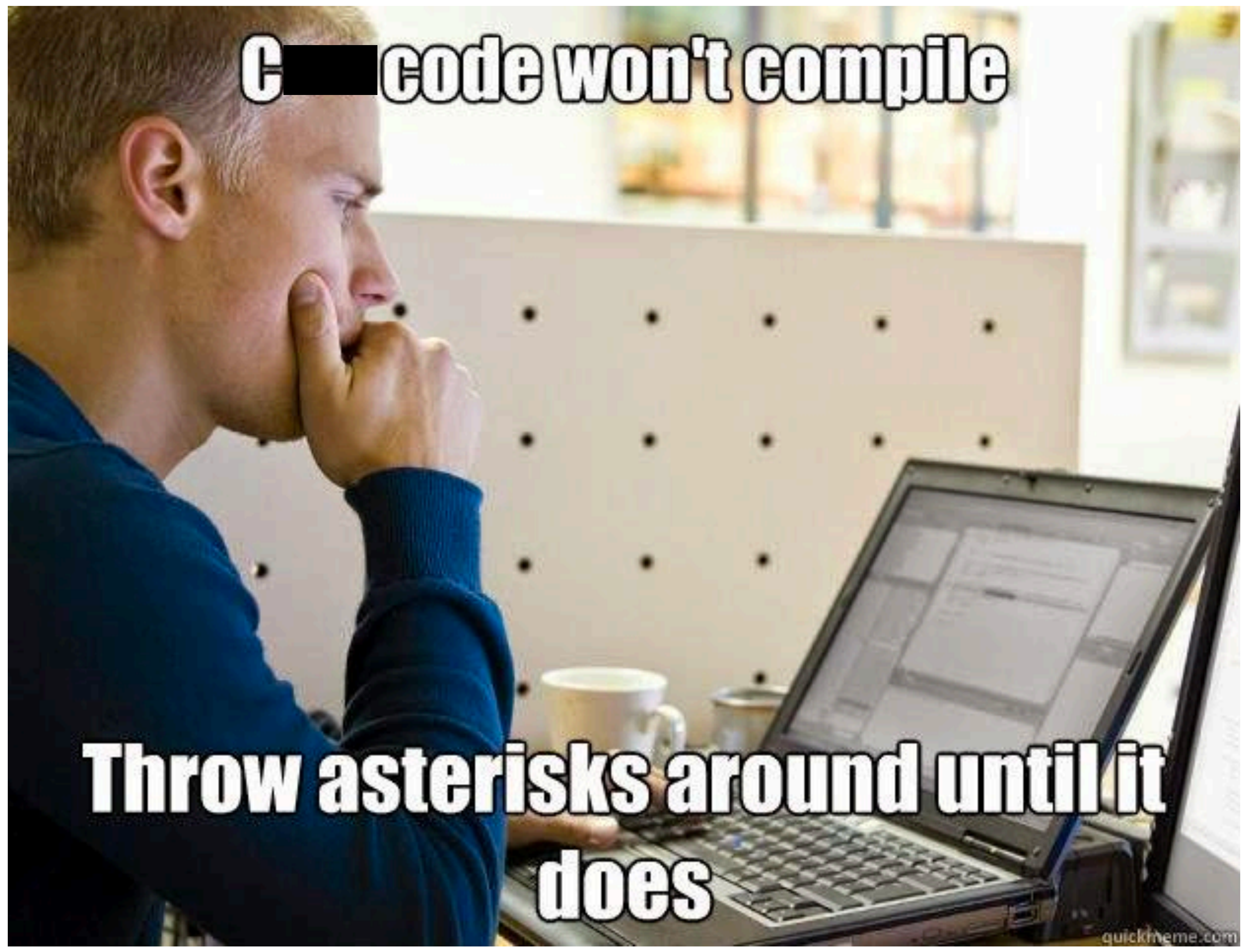
- Assignment progress updates
- Recap of functions

2D Arrays

- Indexing and usage with nested loops.

Pointers

- Storing addresses of variables
- Changing variables directly through functions.


C[redacted]code won't compile
Throw asterisks around until it does
quickmeme.com

2D Arrays

2D Arrays

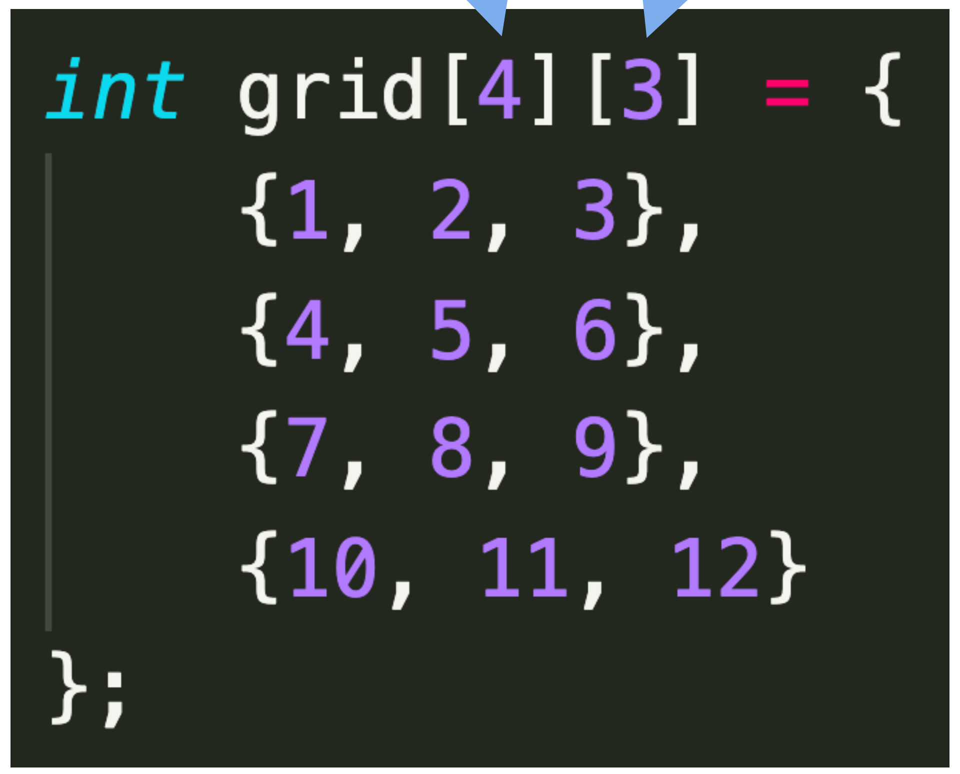
What are they?

- An array of arrays.
- We call them 2D arrays because they we can view them as a grid.

Sample Declaration and Initialisation

number of
1D arrays
(num rows)

number of elements
in each 1D array
(num columns)



```
int grid[4][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

Visualisation

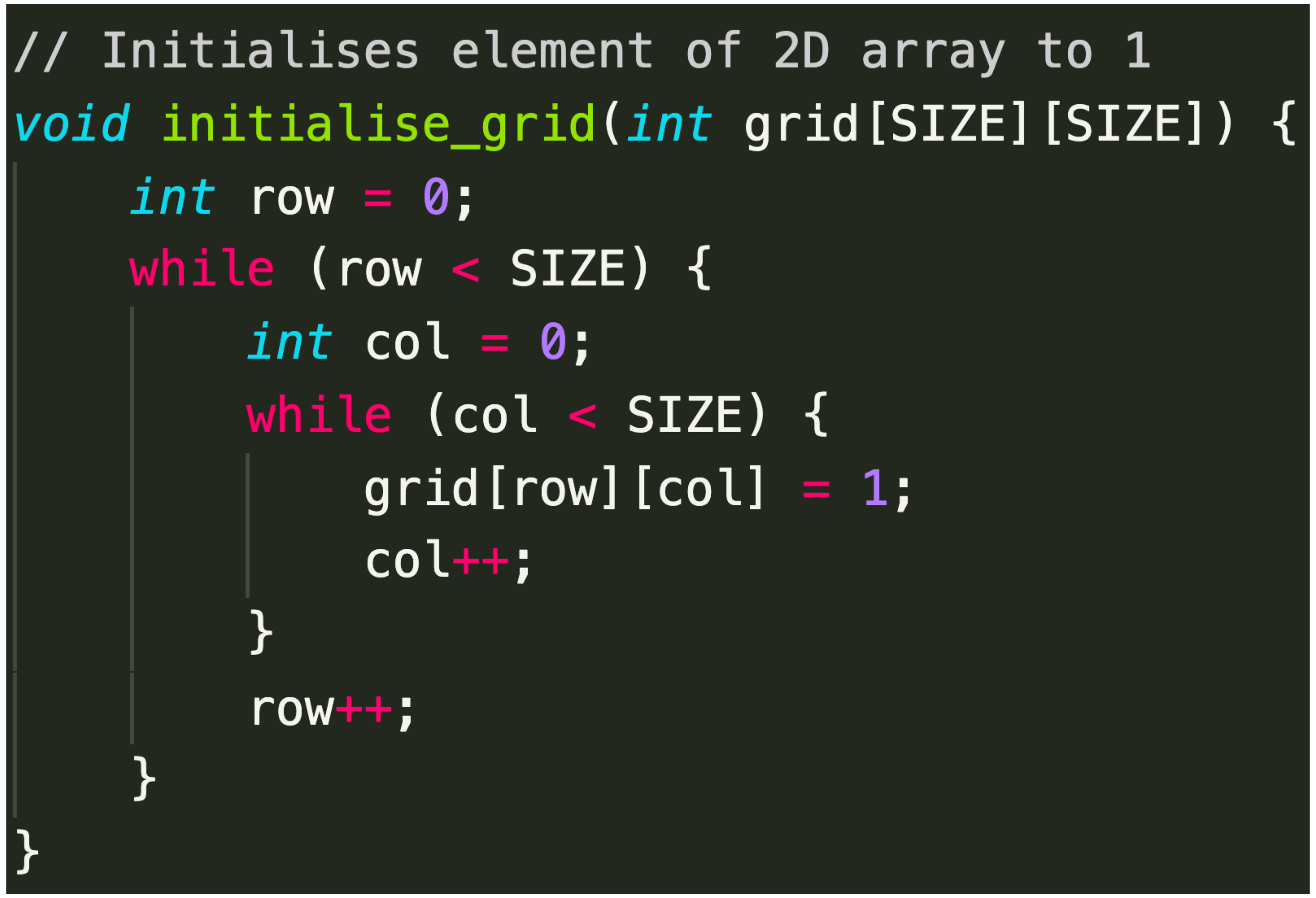
Index	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

`grid[2][1]`
Note: row number is indexed first before column number

2D Array Usage

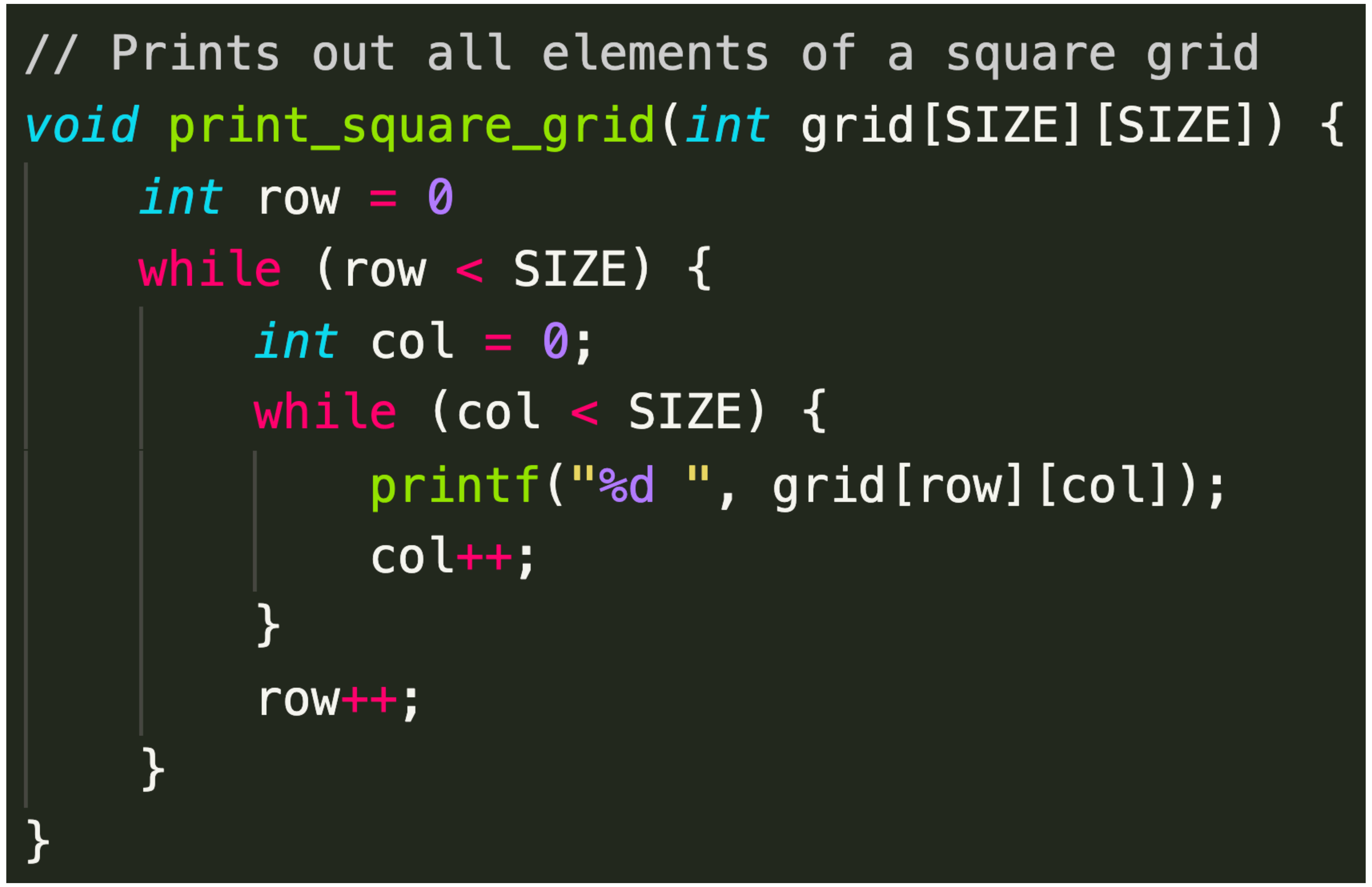
- Commonly alongside nested while loops.
- 2D arrays allow us to store state, which isn't possible with nested while loops alone.

Indexing and Changing Elements



```c
// Initialises element of 2D array to 1
void initialise_grid(int grid[SIZE][SIZE]) {
    int row = 0;
    while (row < SIZE) {
        int col = 0;
        while (col < SIZE) {
            grid[row][col] = 1;
            col++;
        }
        row++;
    }
}
```

Printing Elements



```c
// Prints out all elements of a square grid
void print_square_grid(int grid[SIZE][SIZE]) {
    int row = 0
    while (row < SIZE) {
        int col = 0;
        while (col < SIZE) {
            printf("%d ", grid[row][col]);
            col++;
        }
        row++;
    }
}
```

Assume **`SIZE`** is a **`#define`**'d constant

Pointers

Pointers

What are they?

- Variables that store the memory location (address) of other variables, granting us access to them.
- **Analogy:** Party invitations contain the <u>address</u> of the venue. Guests can follow this address to reach a common destination. We can think of an invitation as a pointer, and the destination as the variable it is pointing to.

Pointer Syntax
Note the 2 meanings of `*`

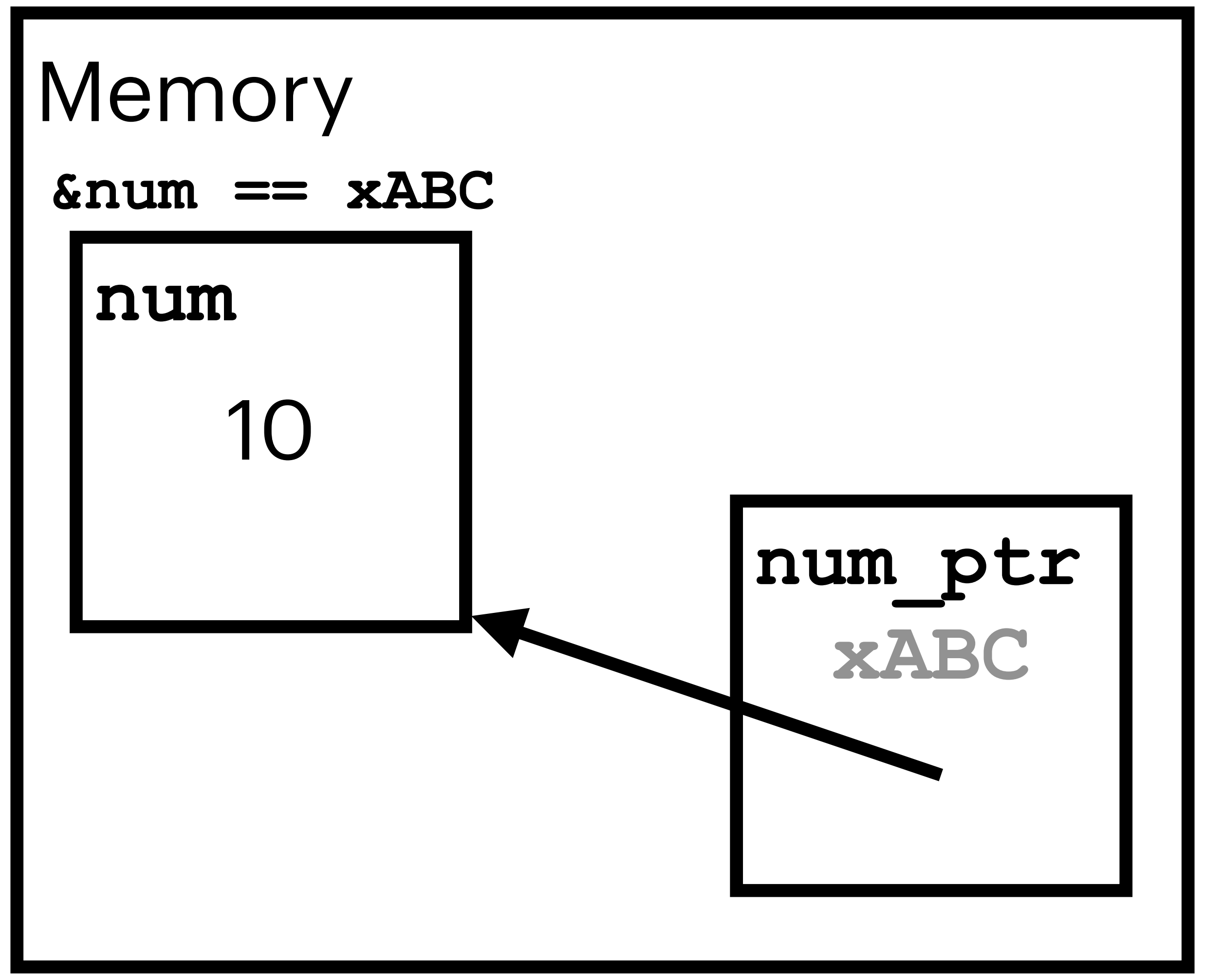
```c
int num = 10;
int *num_ptr = &num;

printf("num_ptr points to %p\n", num_ptr);
printf("at num_ptr, the value %d is stored\n", *num_ptr);
```

use `int *` to initialise a pointer to an integer. This means it must store the address of an integer

`num_ptr` by itself allows us to read the variable value

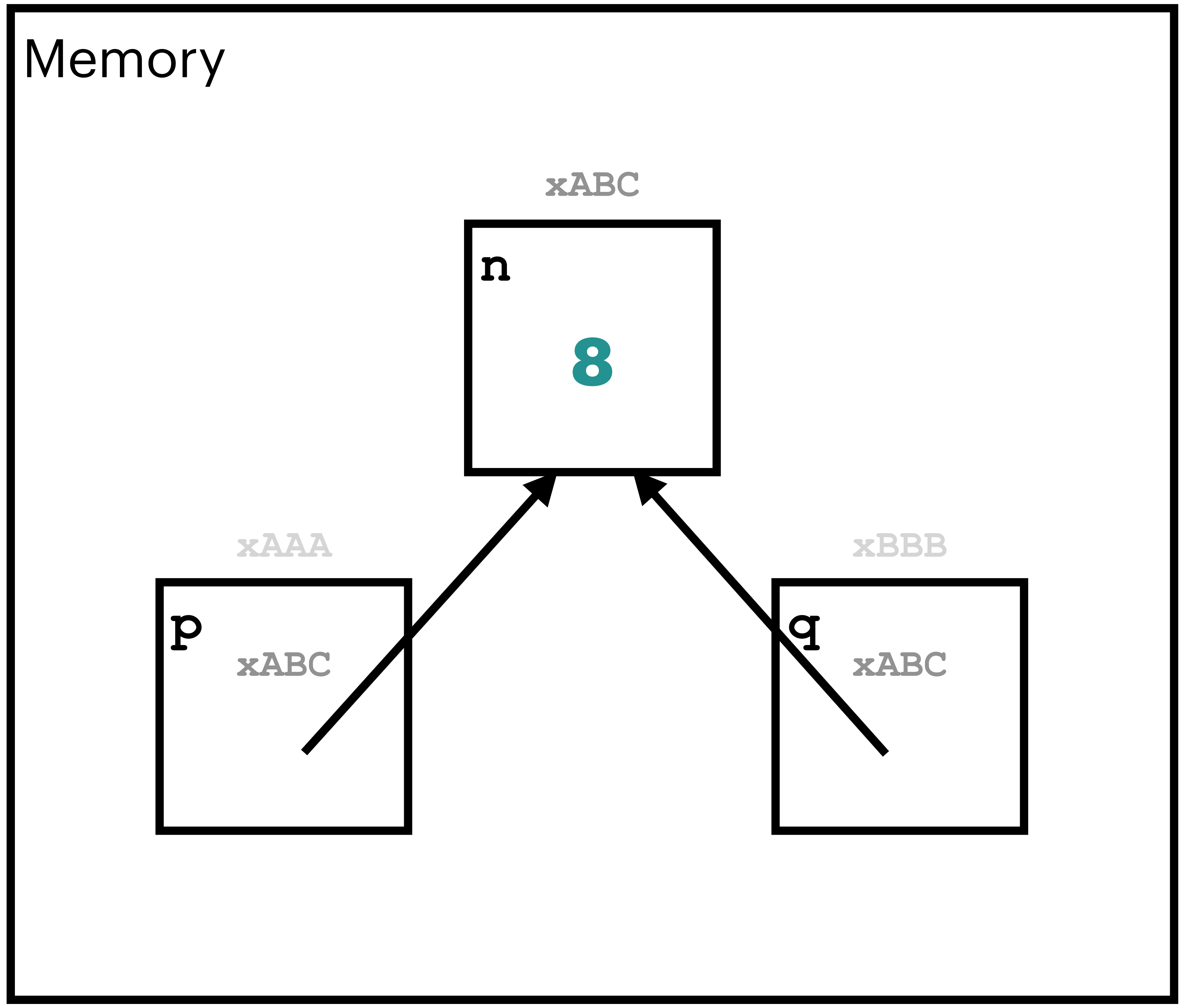
putting a `*` in front of a pointer means 'go to the address stored at the pointer', known as **`dereferencing`** the pointer. This allows us to read or modify that variable



Pointer Syntax

What's happening in this code snippet?

```
✅ int n = 42;
✅ int *p;
✅ int *q;
✅ p = &n;
✅ *p = 5;
❌ *q = 17;   (q doesn't store an address yet)
✅ q = p;
✅ *q = 8;
```

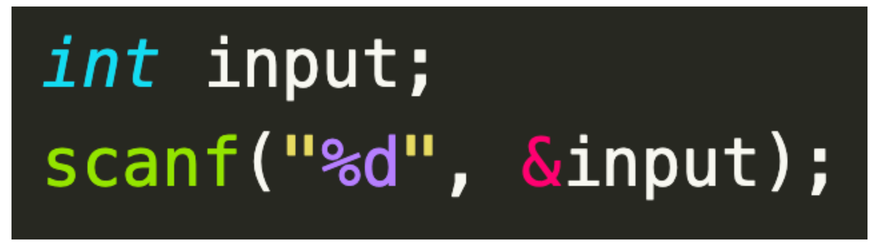


Pointers and Functions

What do they help us achieve?

- In C, all arguments are passed in by value, but we may want to change value of variables through functions.
- Instead of passing in the value of the variable, we can pass in its address.

`scanf` example

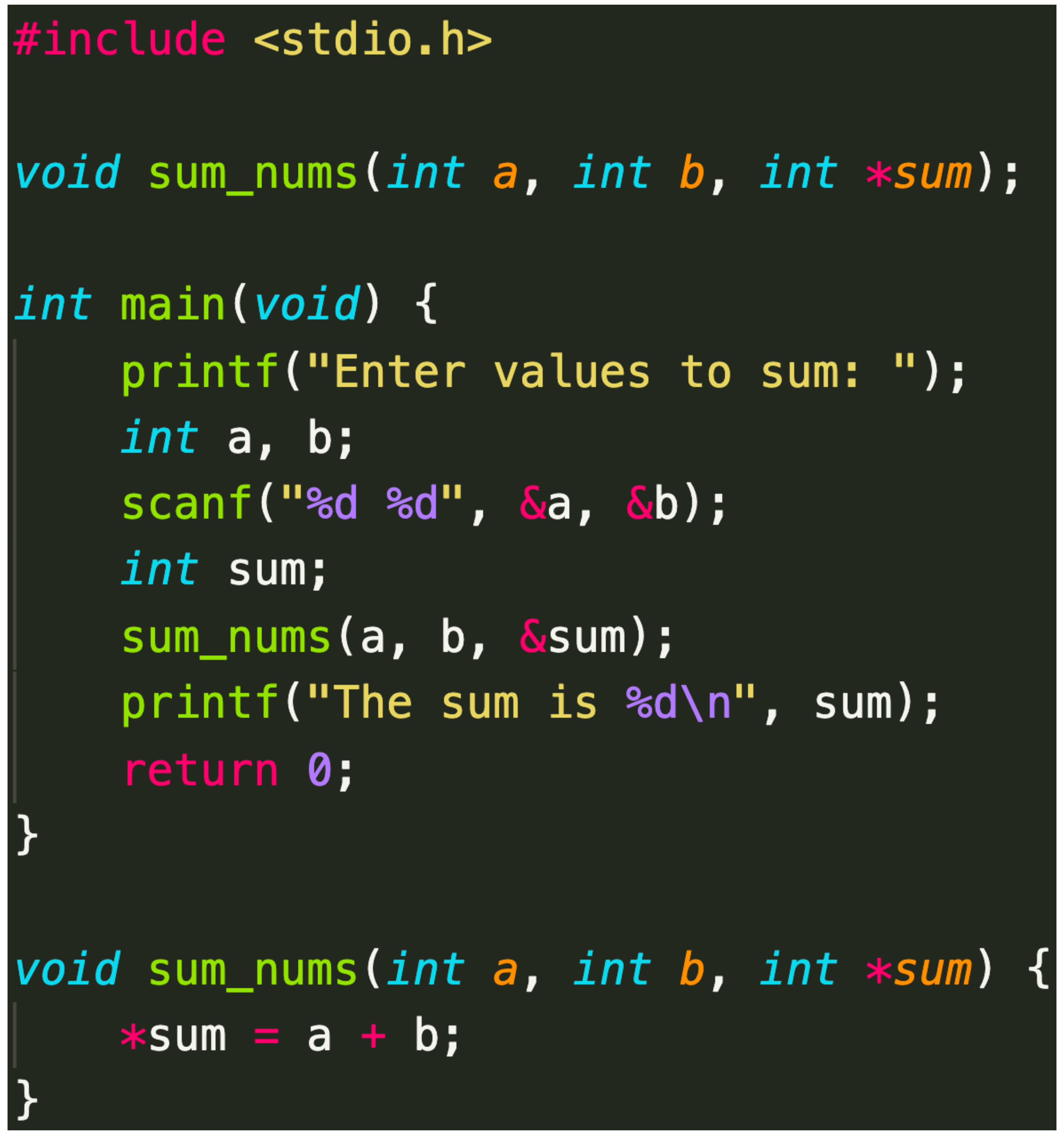


```c
int input;
scanf("%d", &input);
```

- For example, passing the address of a variable into **`scanf`** allows it to put the user's input value into our variable.

Writing Functions with Pointers

Changing values of variables with functions

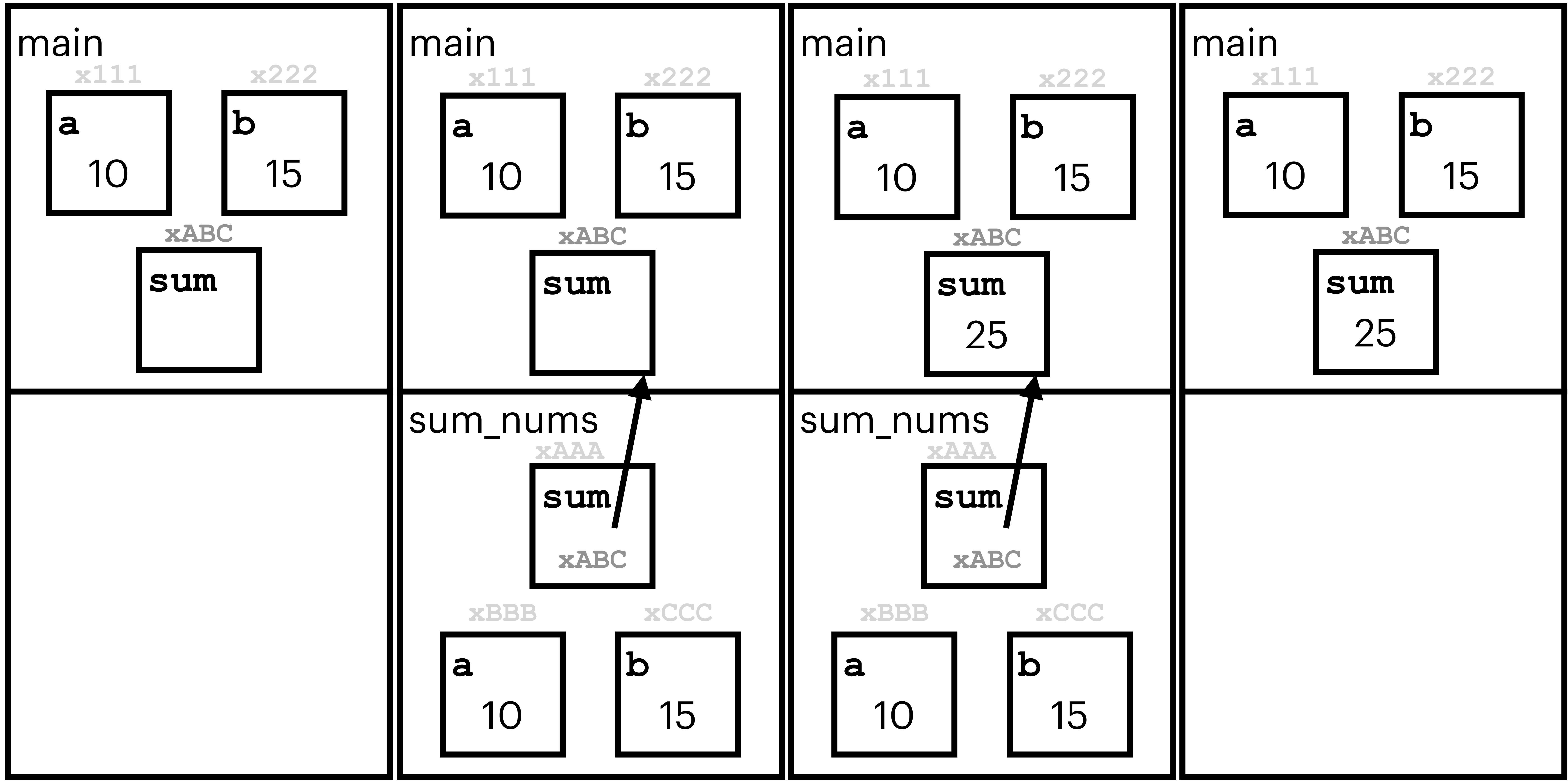


```c
#include <stdio.h>

void sum_nums(int a, int b, int *sum);

int main(void) {
    printf("Enter values to sum: ");
    int a, b;
    scanf("%d %d", &a, &b);
    int sum;
    sum_nums(a, b, &sum);
    printf("The sum is %d\n", sum);
    return 0;
}

void sum_nums(int a, int b, int *sum) {
    *sum = a + b;
}
```



User enters 10 and 15.

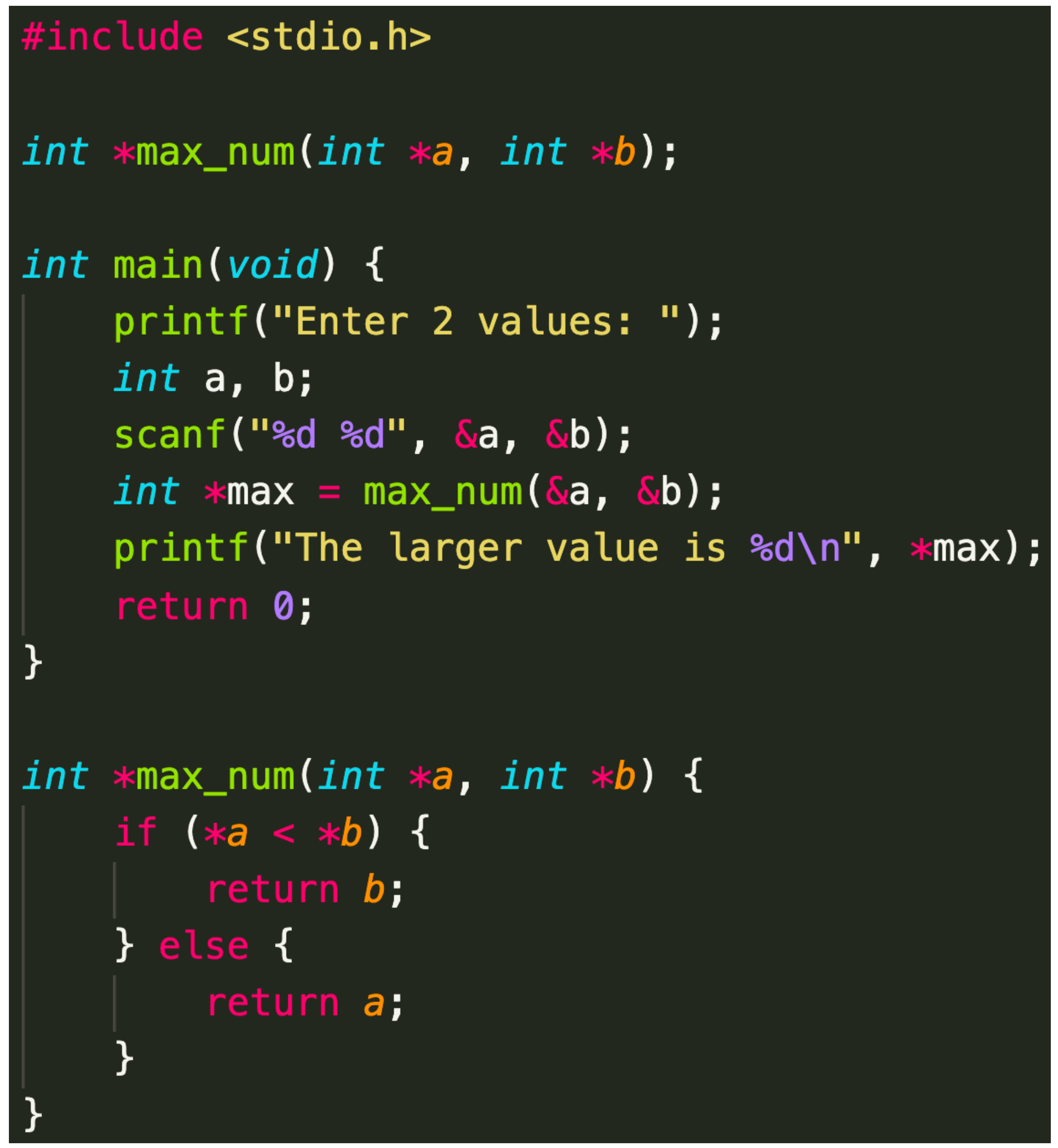
sum_nums is called. **sum**(**int** pointer) in **sum_nums** points to **sum** (**int**) in **main**.

a and **b** are summed together in **sum_nums**. **sum** is dereferenced and the result is stored in **sum** in **main**

Exit **sum_nums**

Functions with Pointers

Returning pointers

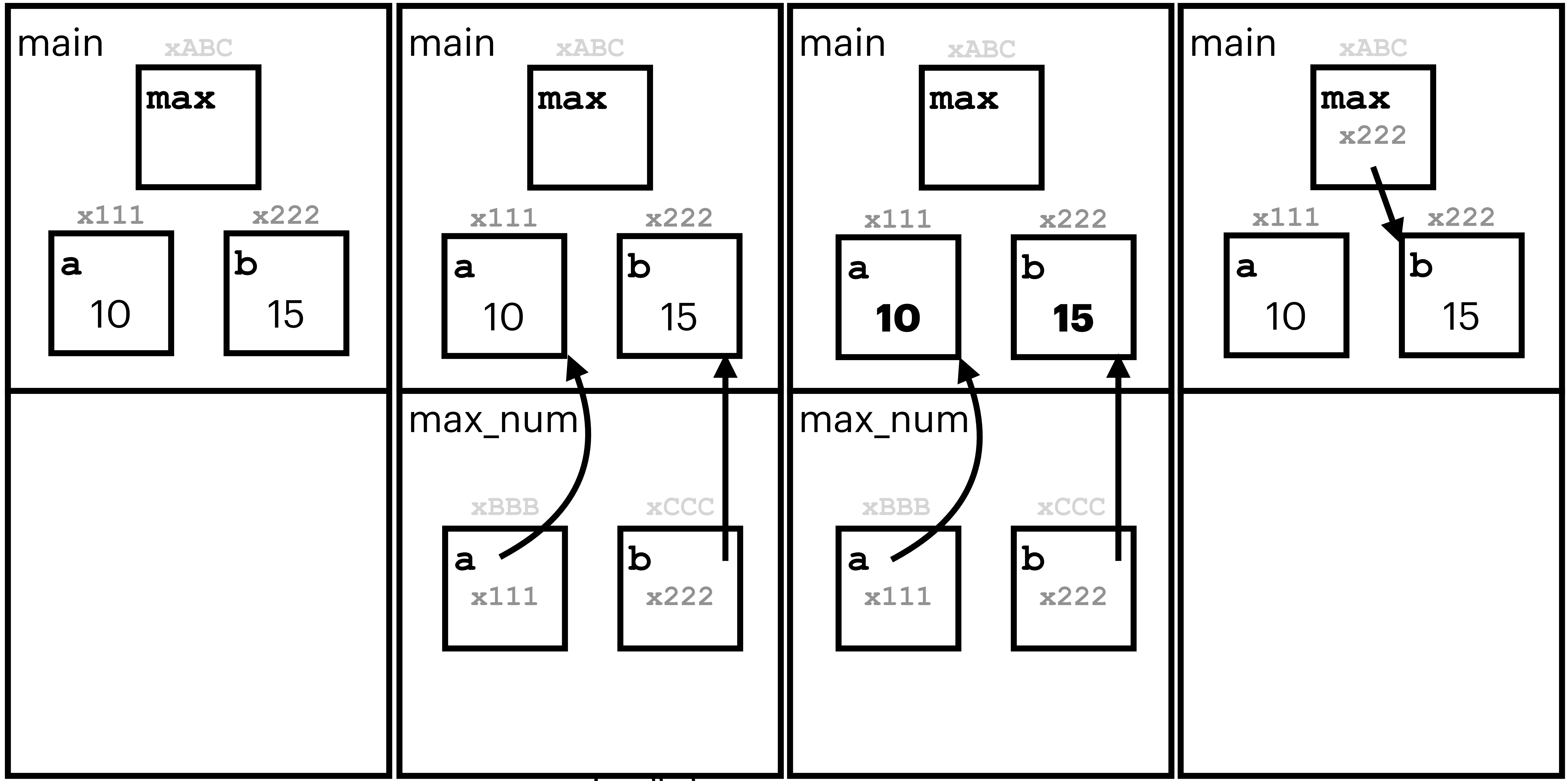


```c
#include <stdio.h>

int *max_num(int *a, int *b);

int main(void) {
    printf("Enter 2 values: ");
    int a, b;
    scanf("%d %d", &a, &b);
    int *max = max_num(&a, &b);
    printf("The larger value is %d\n", *max);
    return 0;
}

int *max_num(int *a, int *b) {
    if (*a < *b) {
        return b;
    } else {
        return a;
    }
}
```



User enters 10 and 15.

max_num is called. **a** and **b**(**int** pointers) in max num points to **a** and **b** (**int**s) n main.

Checking which number is larger...

Returns the pointer **b**. result is stored in **max**