COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 7

An array of arrays, 2D

LAST WEEK...

IN WEEK 3...

- Talked about the importance of style - work neatly as you go!
- Discovered functions (separate chunks of code for reuse, help to segment the problem)
- Got introduced to arrays - homogenous collections - stores the same type of variable in a collection

THIS LECTURE...

TODAY...

- Recap basic arrays
- Recap basic strings
- String library functions
- Array of arrays
- Array of structs
- A taste of pointers... (maybe if we have time!)

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T3/LIVE/WEEK04/

ASSIGNMENT 1

RELEASED TODAY

- Assignment 1 will be released after this lecture
- CS Frogger - welcome to my misspent childhood playing frogger
- Aims of the assignment
  - Apply arrays and two-dimensional arrays in solving problems
  - Apply good style to your code
  - Apply the use of functions in code
  - Practice skills in debugging code, and skills in patience as you search for one missing semi-colon

ASSIGNMENT 1

LIVESTREAM

- The Assignment has 4 stages, each stage ramps up with difficulty (just like the lab exercises)
- Suggest going through the stages chronologically - do not skip stages
- Live Stream to go through the assignment in more detail:
  - Friday 12:30pm
  - Link for the livestream:

RECAP OF ARRAYS

Remember that arrays:

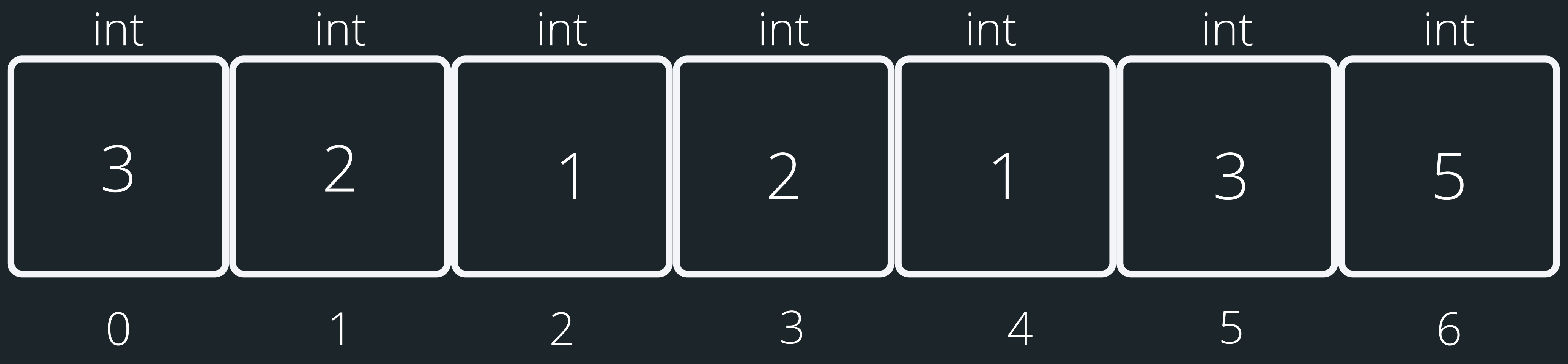
- are a collection all of the same type
- are declared by using a type, name and a size of the array
- you can easily access individual elements of an array by using an index
- Indexing starts at 0 and moves through until (size - 1) of the array
- go hand in hand with while loops that make it easy to work through an array

RECAP OF ARRAYS

- So let's say we have this declared and initialised:

```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};
```

- This is what it looks like visually:



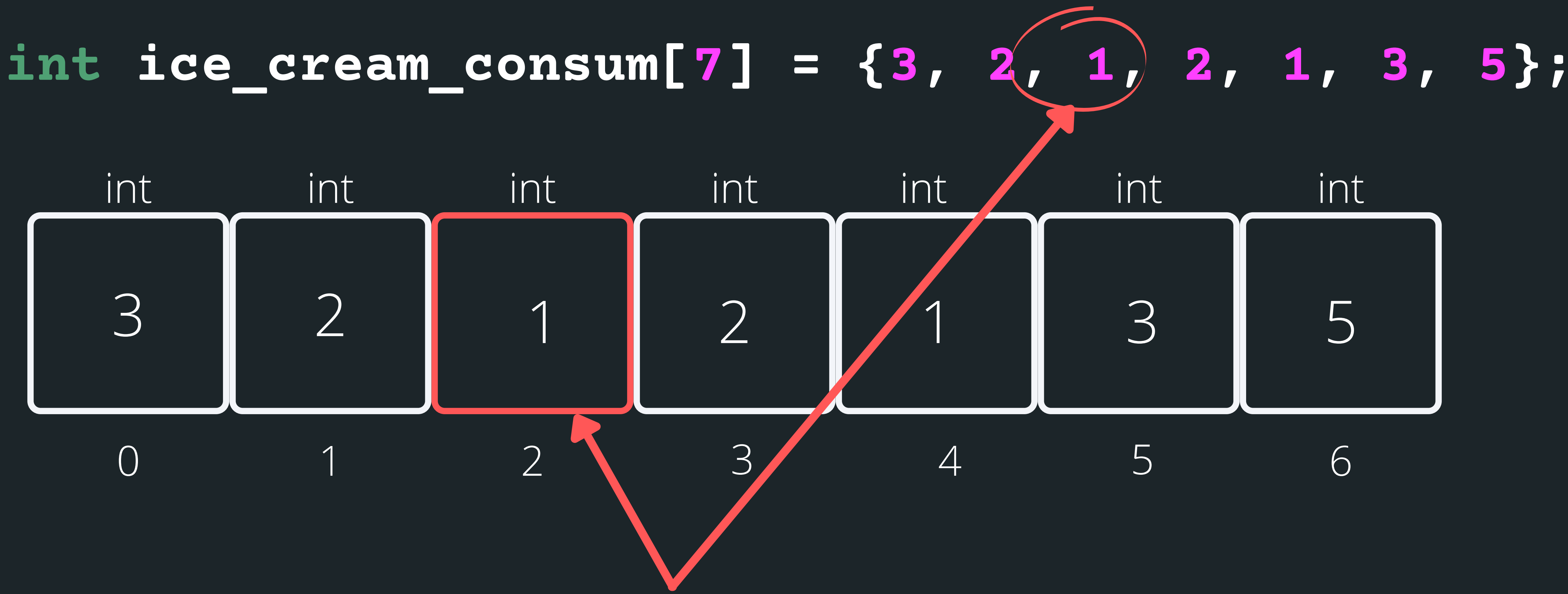
this array holds 7 integers

Note that indexing starts at 0

RECAP OF ARRAYS

- You can access any element of the array by referencing its index
- Note, that indexes start from 0
- Trying to access an index that does not exist, will result in an error

```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};
```



If I wanted the third element of the array

The index would be 2, so to access it:
ice_cream_consum[2]

RECAP OF ARRAYS

AN EXAMPLE PROBLEM

Problem: A user is asked to enter 10 numbers. We will then go through these numbers and find the lowest number and output what the lowest number is to the user.

`lowest_number.c`

RECAP OF STRINGS

WHAT ARE THEY?

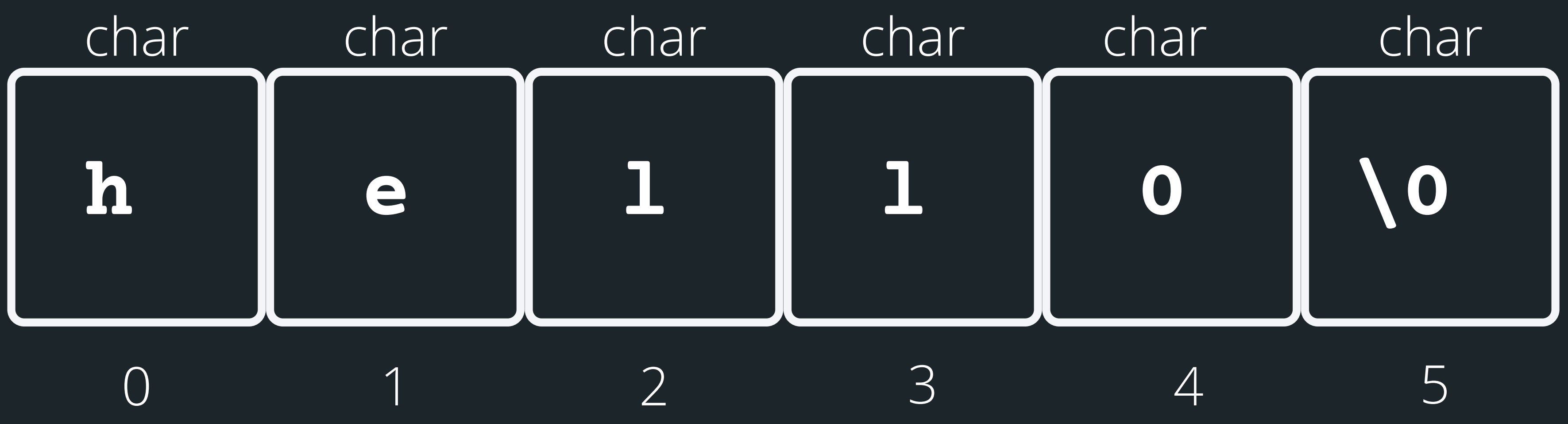
- Strings are a collection of characters that are joined together
  - an array of characters!
- There is one very special thing about strings in C - it is an array of characters that finishes with a
  - This symbol is called a null terminating character
- It is always located at the end of an array, therefore an array has to always be able to accomodate this character
- It is not displayed as part of the string
- It is a placeholder to indicate that this array of characters is a string
- It is very useful to know when our string has come to an end, when we loop through the array of characters

HOW DO WE DECLARE A STRING?

WHAT DOES IT LOOK LIKE VISUALLY?

- Because strings are an array of characters, the array type is char.
- To declare and initialise a string, you can use two methods:

```c
//the more convenient way
char word[] = "hello";
//this is the same as'\0':
char word[] = {'h','e','l','l','o','\0'};
```



HELPFUL LIBRARY FUNCTIONS FOR STRINGS

FGETS()

There is a useful function for reading strings:

`fgets(array[], length, stream)`

The function needs three inputs:

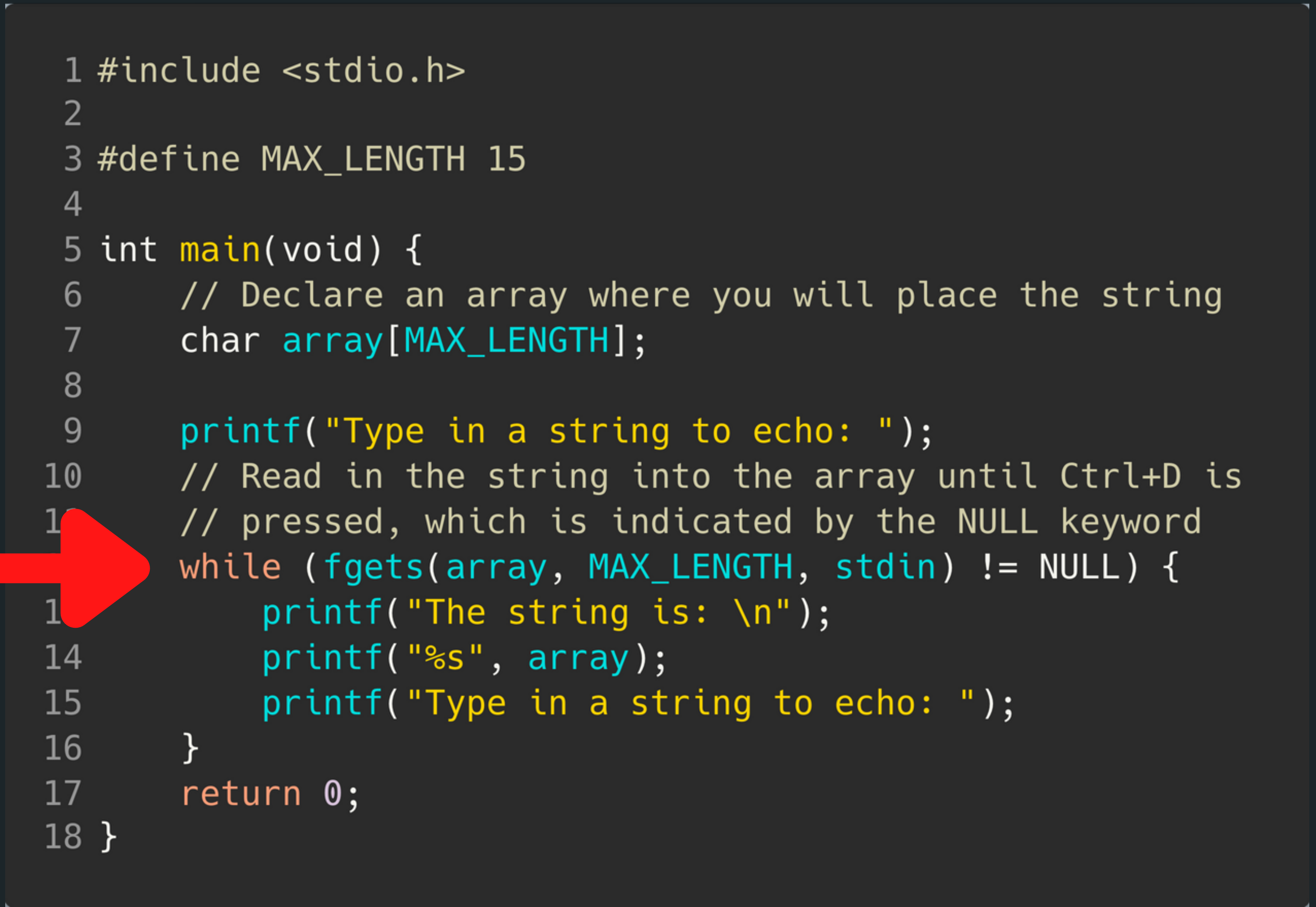
- array[] - the array that the string will be stored into
- length - the number of characters that will be read in
- stream - this is where this string is coming from - you don't have to worry about this one, in your case, it will always be stdin (the input will always be from terminal)

```
// Declare an array where you will place the
string that you read from somewhere
char array[MAX_LENGTH];
// Read in the string into array of length
MAX_LENGTH from terminal input
fgets(array, MAX_LENGTH, sdin)
```

HOW DO I KEEP READING STUFF IN OVER AND OVER AGAIN?

Using the **NULL** keyword, you can continuously get string input from terminal until Ctrl+D is pressed

- fgets() stops reading when either length-1 characters are read, newline character is read or an end of file is reached, whichever comes first



```c
#include <stdio.h>

#define MAX_LENGTH 15

int main(void) {
    // Declare an array where you will place the string
    char array[MAX_LENGTH];

    printf("Type in a string to echo: ");
    // Read in the string into the array until Ctrl+D is
    // pressed, which is indicated by the NULL keyword
    while (fgets(array, MAX_LENGTH, stdin) != NULL) {
        printf("The string is: \n");
        printf("%s", array);
        printf("Type in a string to echo: ");
    }
    return 0;
}
```

SOME OTHER INTERESTING STRING FUNCTIONS

<STRING.H> STANDARD LIBRARY

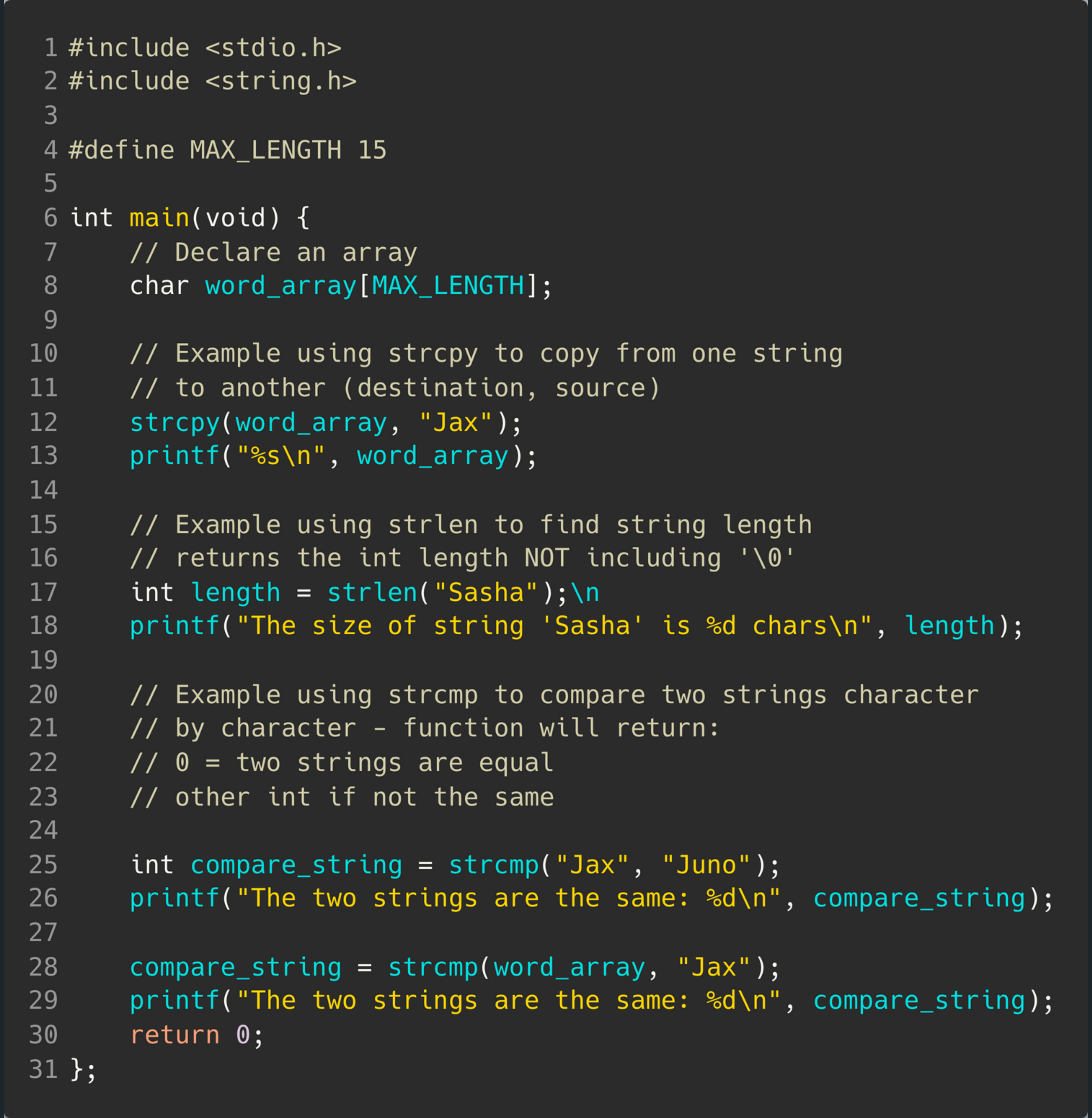
CHECK OUT THE REST OF THE FUNCTIONS: HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/STRING_H.HTM

Some other useful functions for strings:

- **`strlen()`**gives us the length of the string (excluding the '\0'
- **`strcpy()`**copy the contents of one string to another
- **`strcat()`**attach one string to the end of another (concatenate)
- **`strcmp()`**compare two strings
- **`strchr()`** find the first or last occurance of a character

USING SOME OF THESE FUNCTIONS

STRINGS



```c
#include <stdio.h>
#include <string.h>

#define MAX_LENGTH 15

int main(void) {
    // Declare an array
    char word_array[MAX_LENGTH];

    // Example using strcpy to copy from one string
    // to another (destination, source)
    strcpy(word_array, "Jax");
    printf("%s\n", word_array);

    // Example using strlen to find string length
    // returns the int length NOT including '\0'
    int length = strlen("Sasha");\n
    printf("The size of string 'Sasha' is %d chars\n", length);

    // Example using strcmp to compare two strings character
    // by character - function will return:
    // 0 = two strings are equal
    // other int if not the same

    int compare_string = strcmp("Jax", "Juno");
    printf("The two strings are the same: %d\n", compare_string);

    compare_string = strcmp(word_array, "Jax");
    printf("The two strings are the same: %d\n", compare_string);
    return 0;
};
```

YOU CAN HAVE AN ARRAY OF ANYTHING

AN ARRAY OF STRUCTS

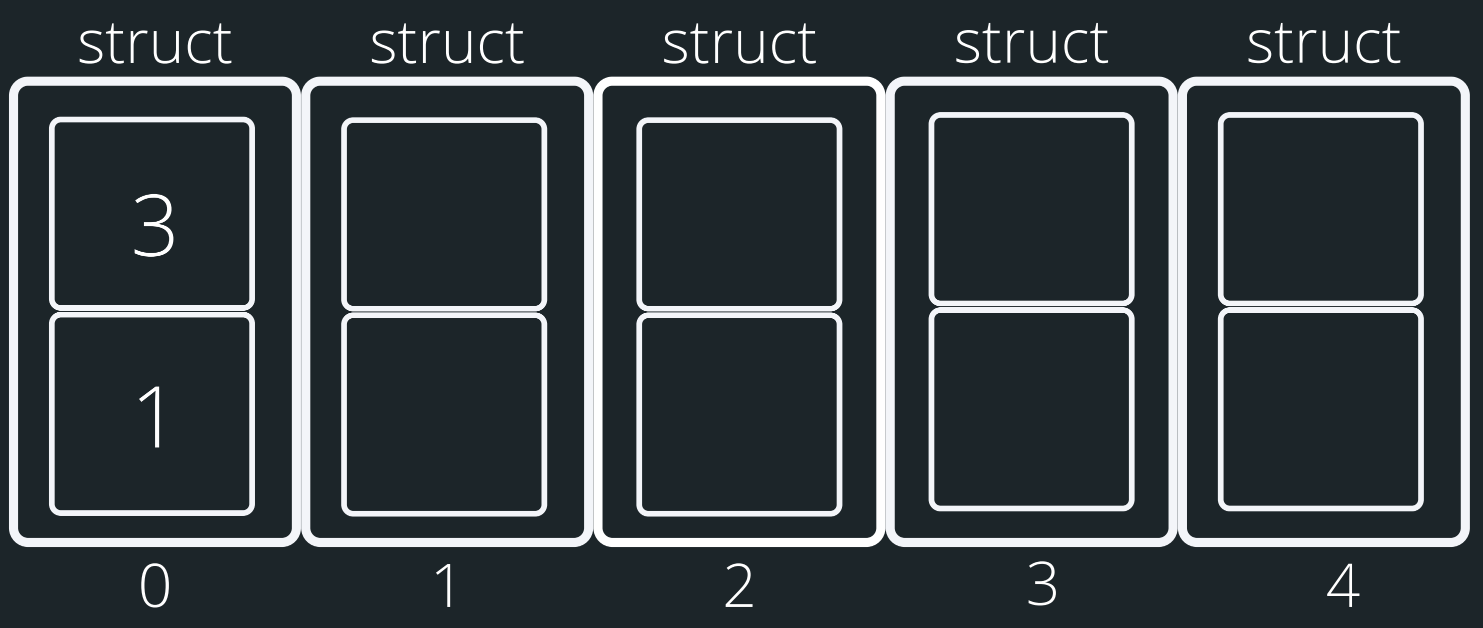
The struct for a coordinate point:

```
struct coordinate {
    int x;
    int y;
};
```

An array of structs declared:

```
struct coordinate map[5];
```

An array of structs visually:



```
map[0].x = 3;
map[0].y = 1;
```

ACCESSING AN ELEMENT INSIDE ARRAY OF ARRAYS

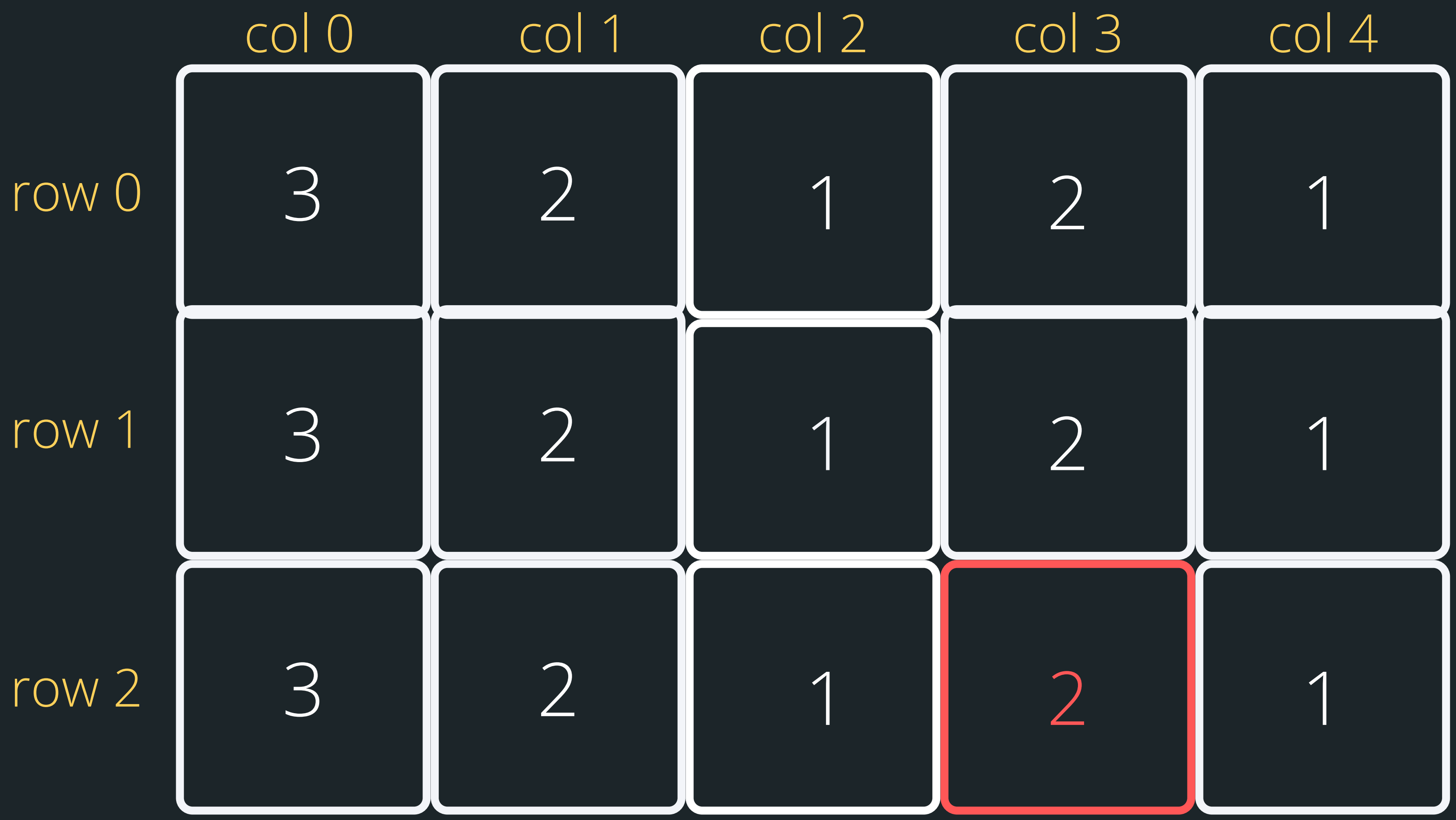
An array of arrays is basically a grid. To declare an array of arrays:

```
type array_name[num of rows][num of columns];
```

```
int array[3][5];
```

To access an element now you will need to:

```
array[2][3];
```

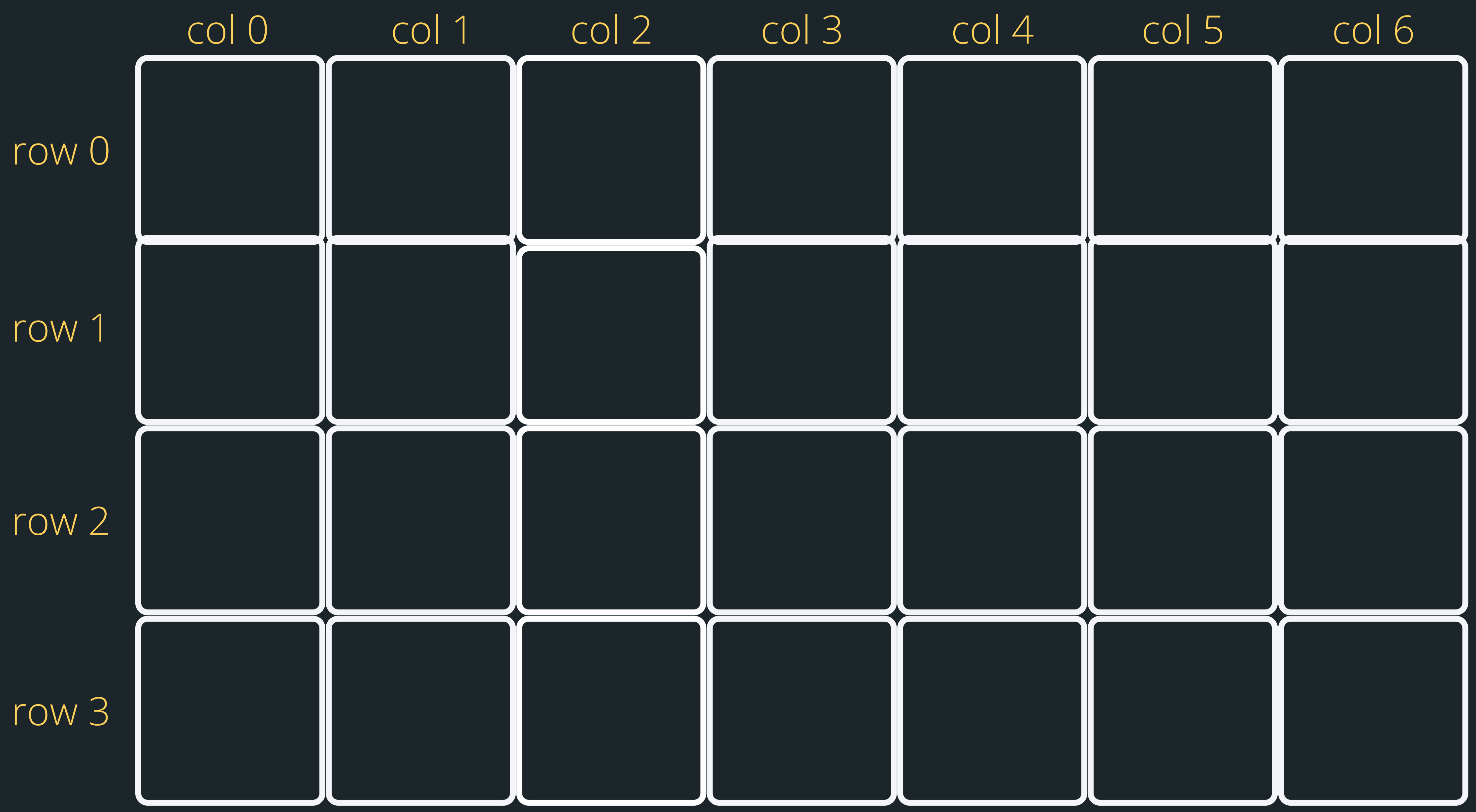


	col 0	col 1	col 2	col 3	col 4
row 0	3	2	1	2	1
row 1	3	2	1	2	1
row 2	3	2	1	2	1

ARRAY OF ARRAYS

Think of the problem last week where we tracked ice-cream consumption for a week. What if I want to do this for a month (a week at a time)?

```
int ice_cream[4][7];
```



REMEMBER A WHILE LOOP INSIDE A WHILE LOOP TO PRINT A GRID?

Do you remember when we printed out a grid of numbers in Week 2 (Friday night vibes)?

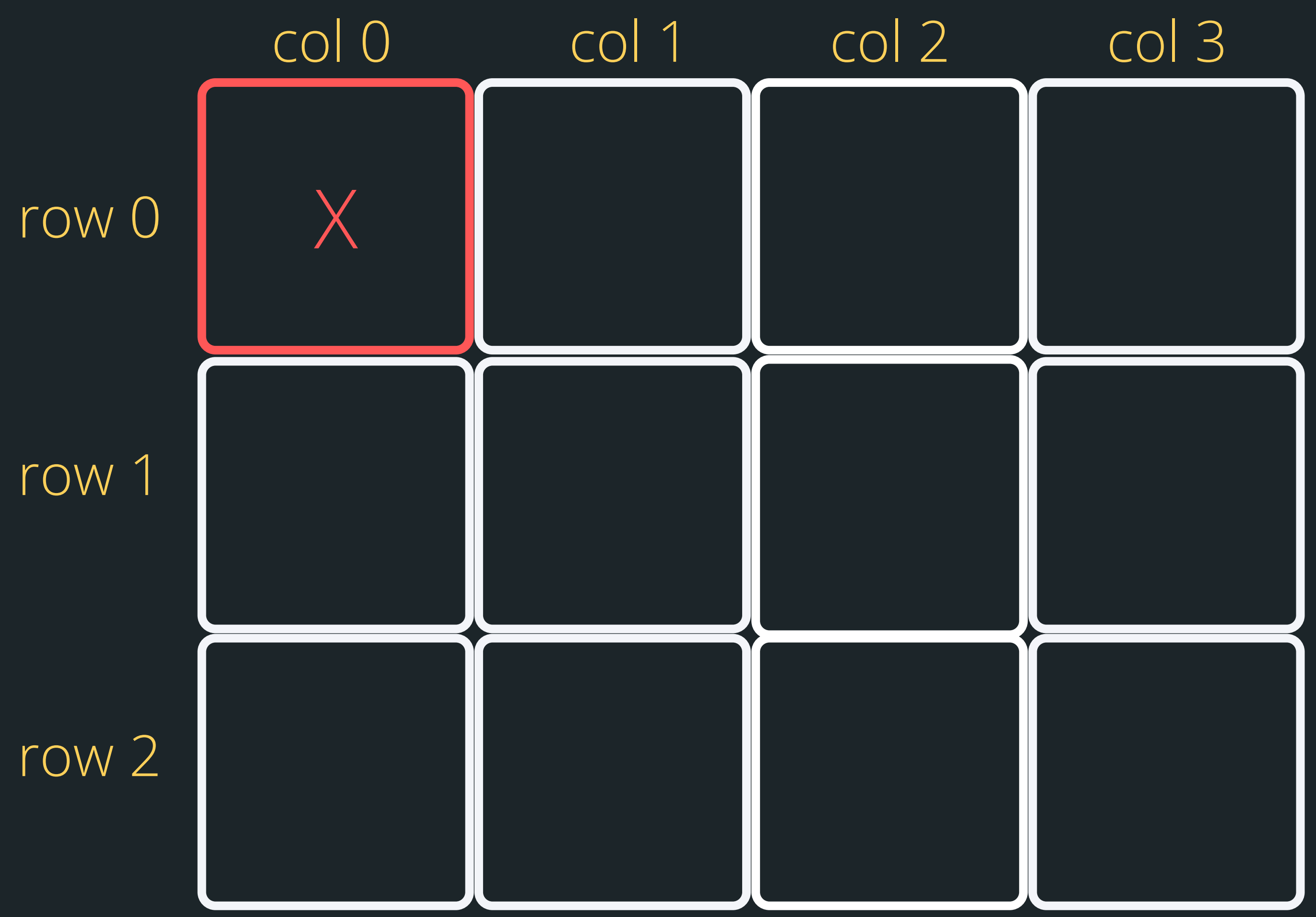
```c
int row = 0;
while (row <= SIZE){
    int col = 0;
    while (col <= SIZE){
        printf("%d", col);
        col++;
    }
printf("\n");
row++;
}
```

How can we transfer this knowledge to print out an array of arrays?

TRANSFER THIS TO AN ARRAY:

FIRST RUN AROUND THE SUN:
OUTSIDE WHILE
    ROW = 0
INSIDE WHILE
    COL = 0

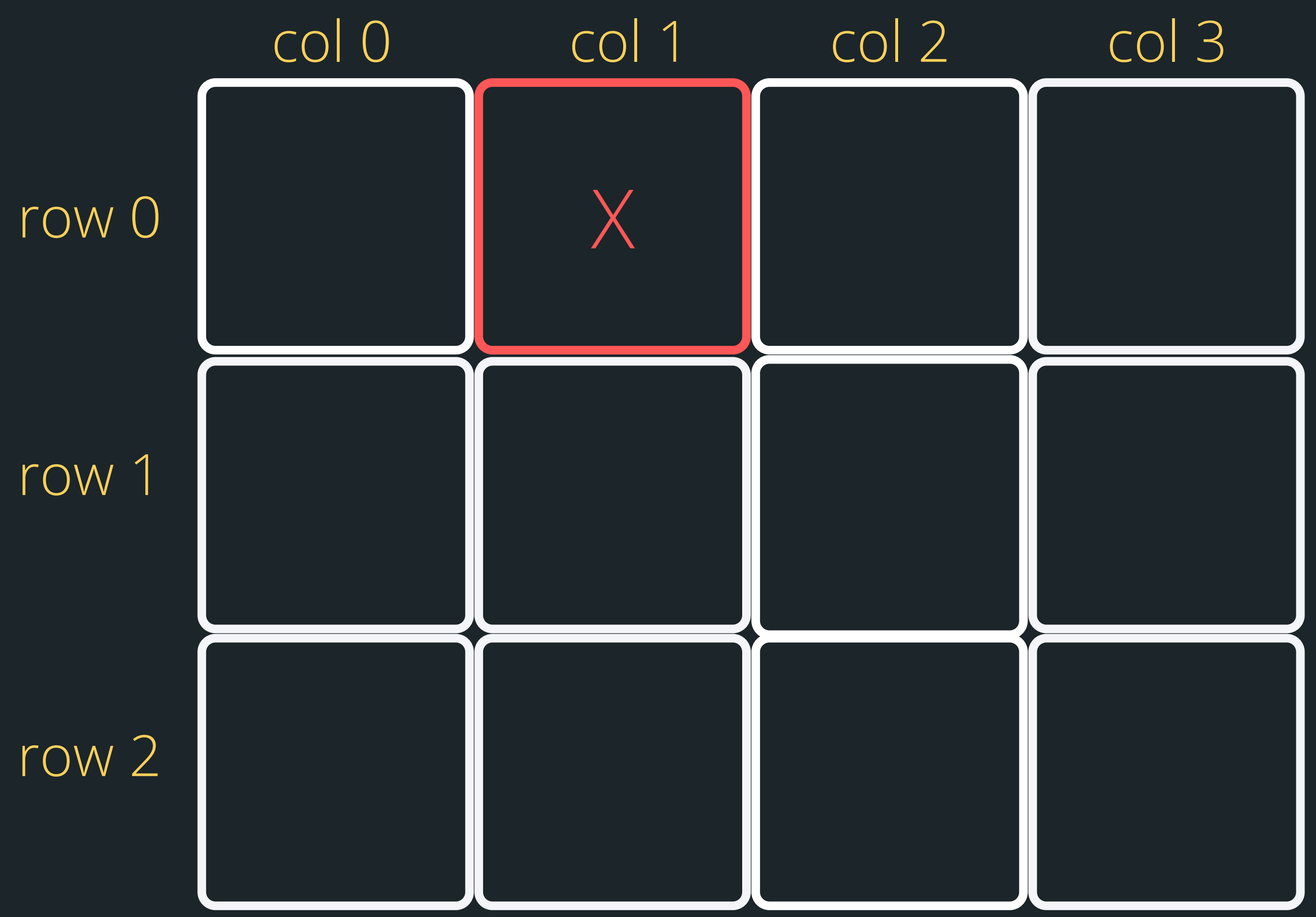
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

FIRST RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 0
INSIDE WHILE
COL = 1

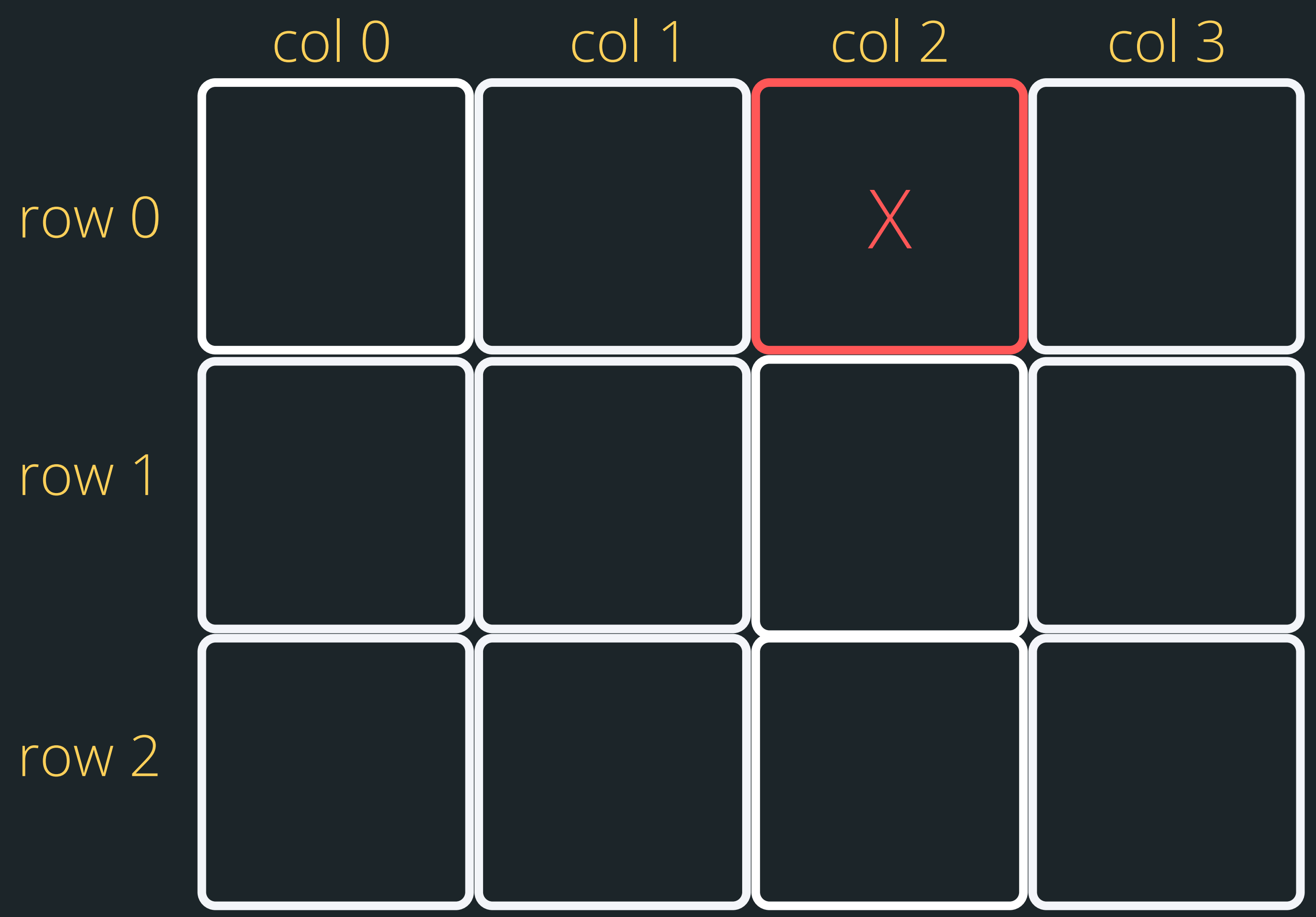
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

FIRST RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 0
INSIDE WHILE
COL = 2

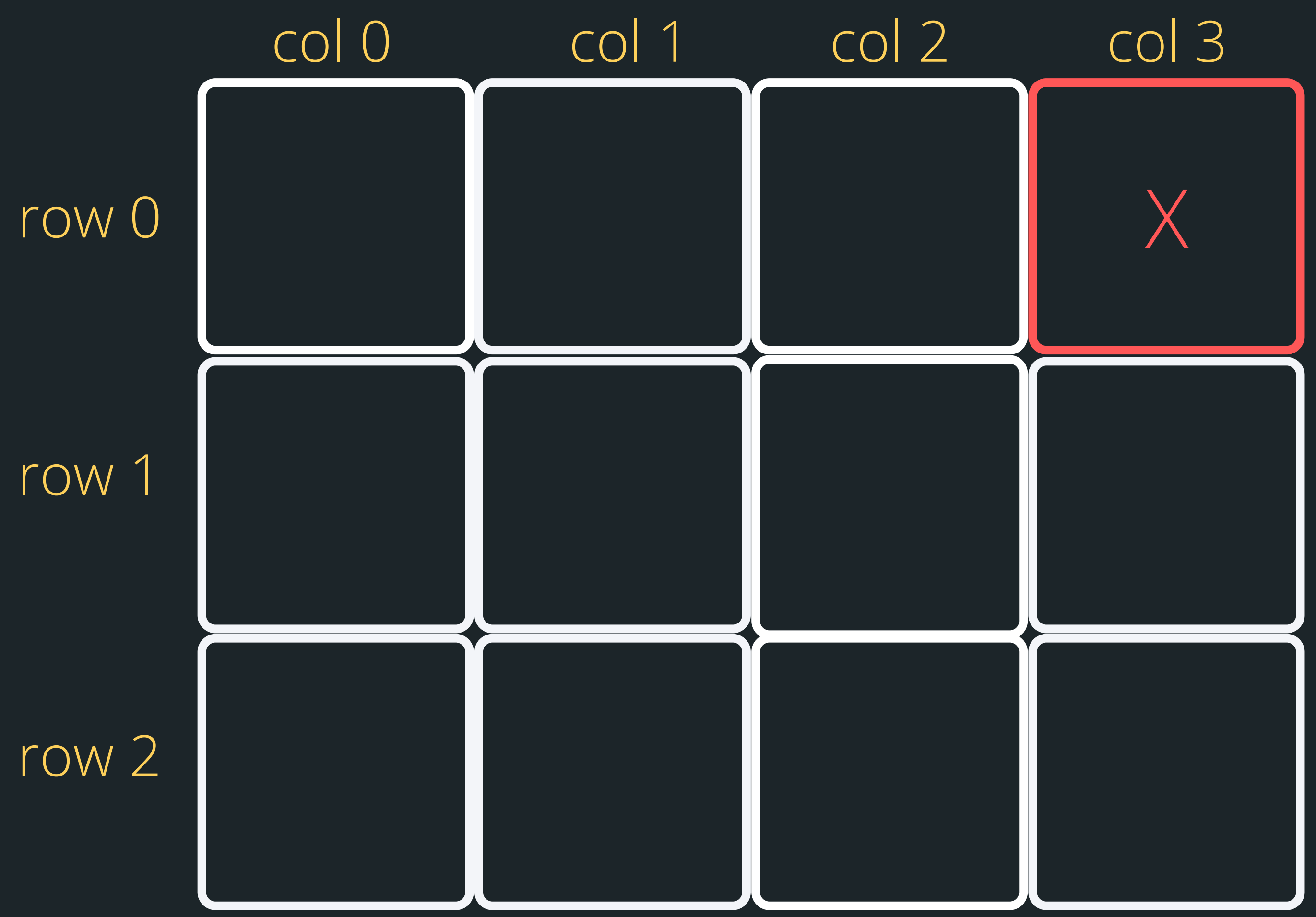
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

FIRST RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 0
INSIDE WHILE
COL = 3

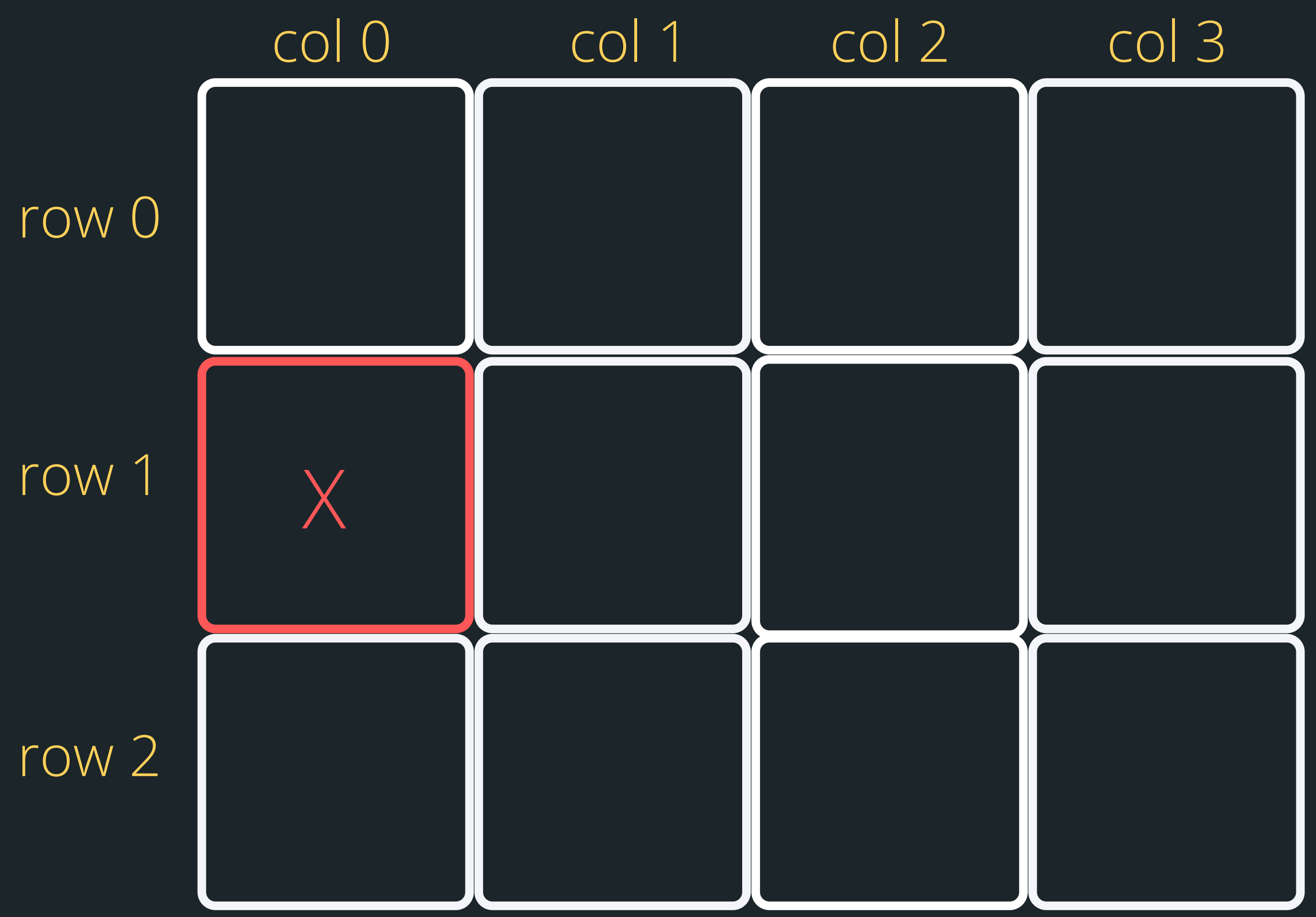
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

SECOND RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 1
INSIDE WHILE
COL = 0

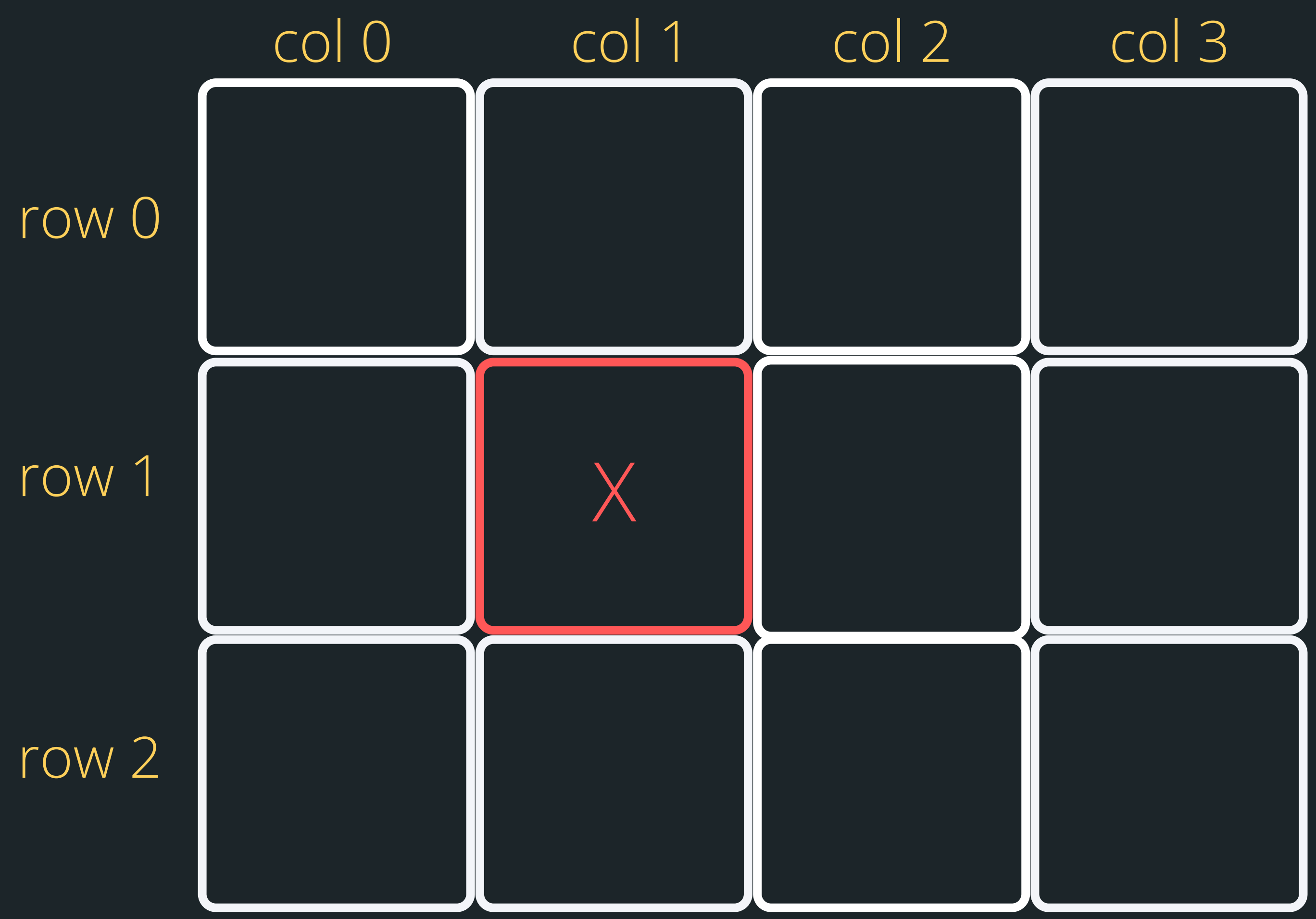
```
int array[3][4];
int row = 0;
while (row <= 3){
     int col = 0;
     while (col <= 4){
          printf("%d", array[row][col]);
          col++;
     }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

SECOND RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 1
INSIDE WHILE
COL = 1

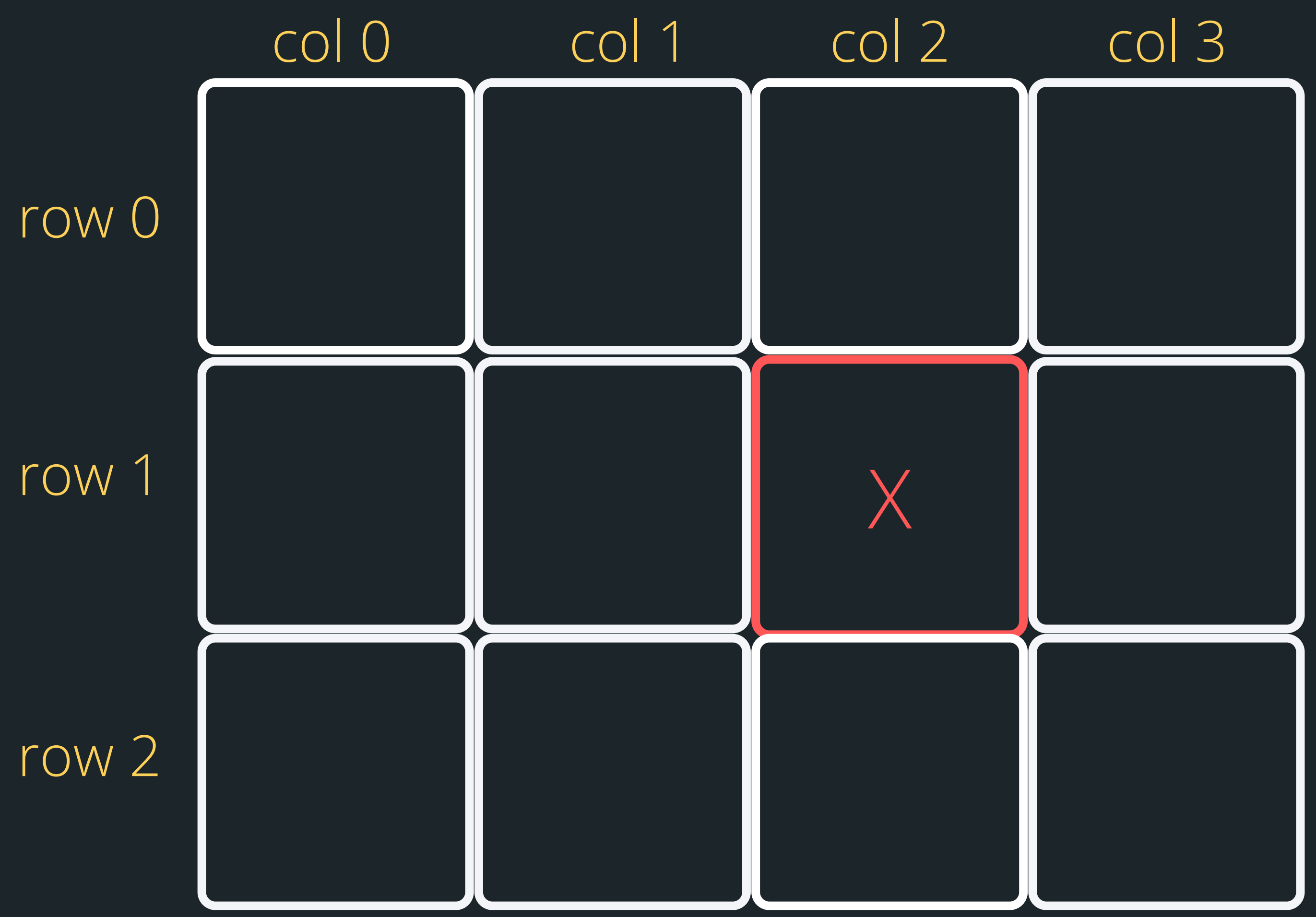
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

SECOND RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 1
INSIDE WHILE
COL = 2

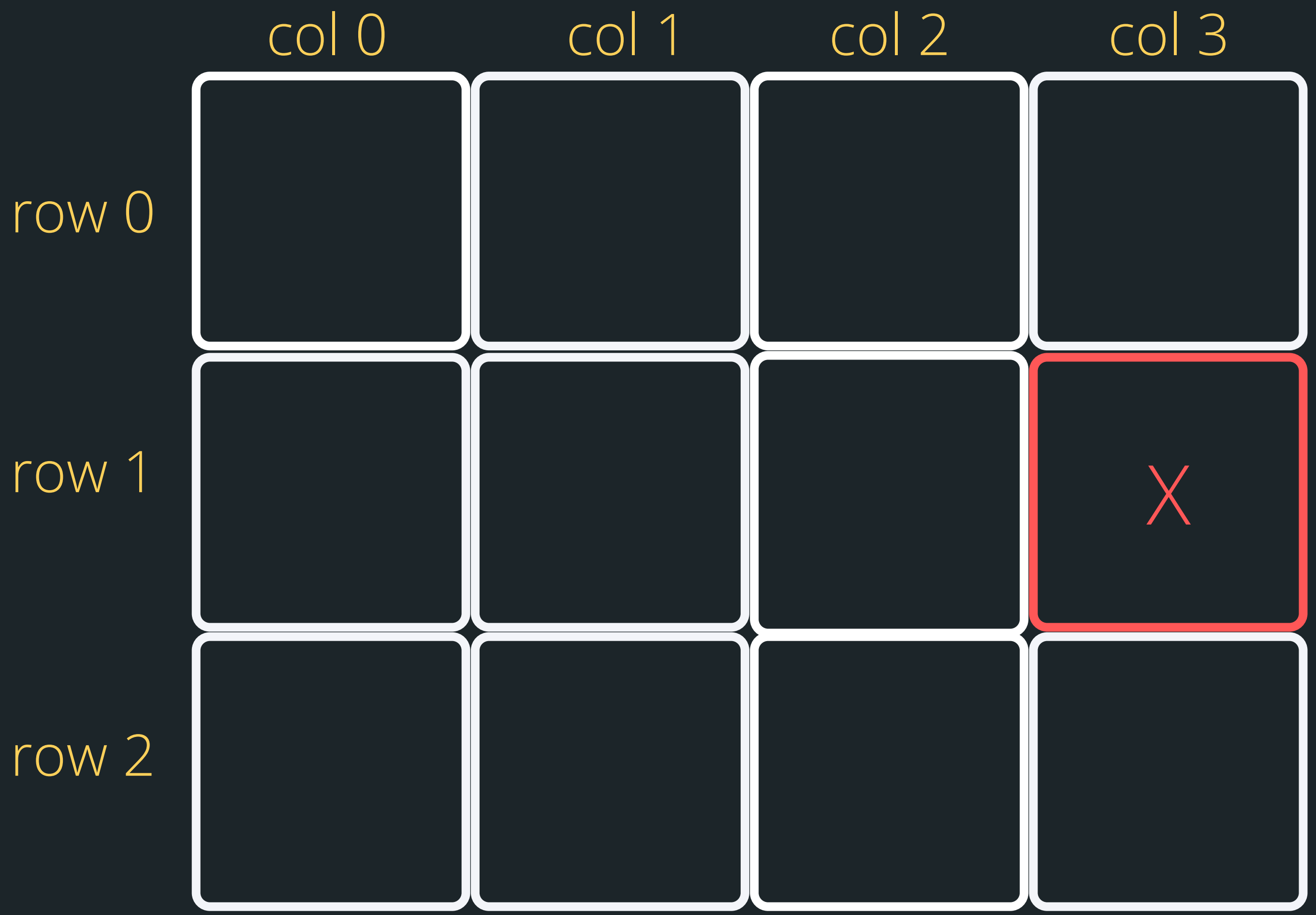
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

SECOND RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 1
INSIDE WHILE
COL = 3

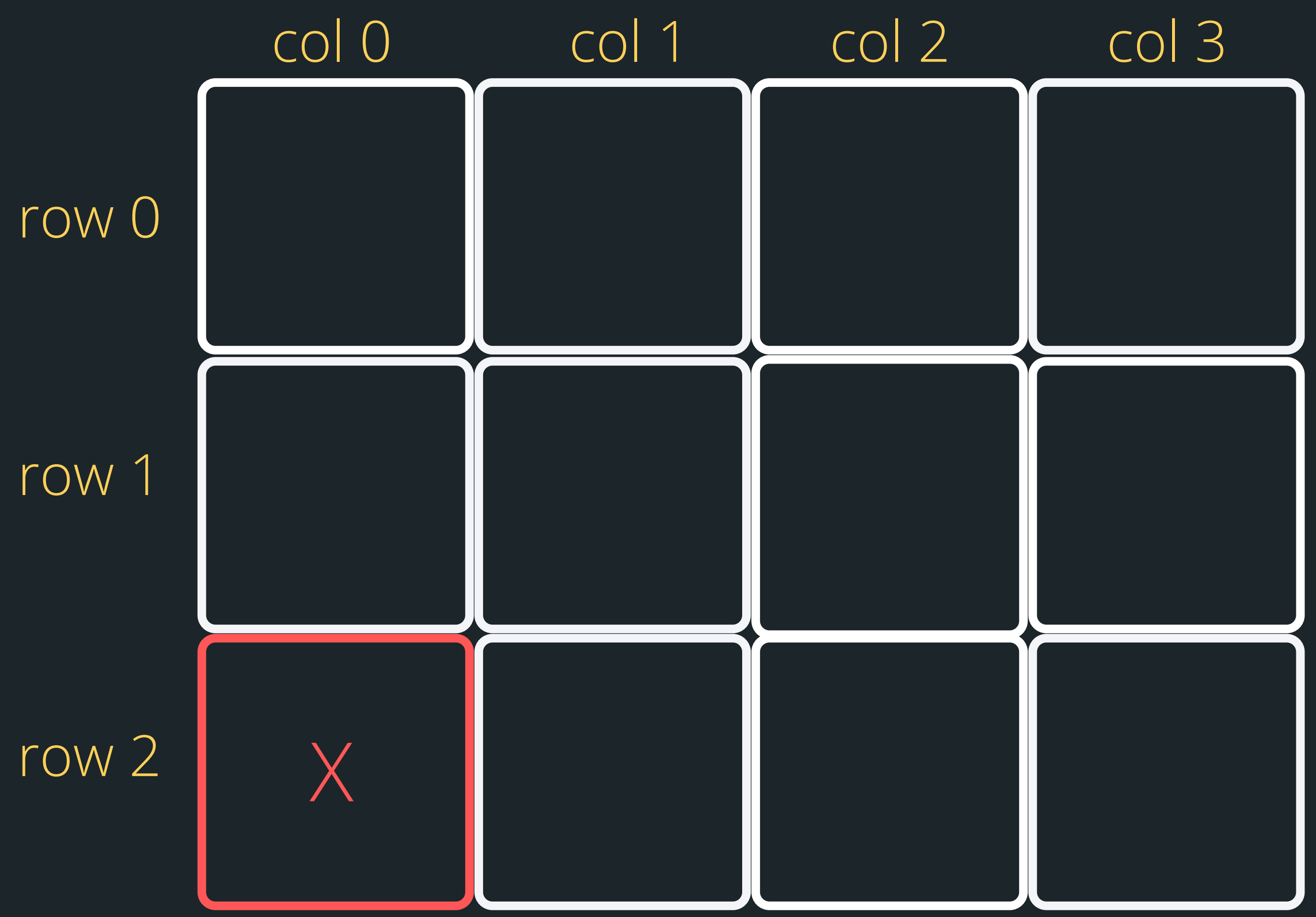
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

THIRD RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 2
INSIDE WHILE
COL = 0

```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

THIRD RUN AROUND THE SUN:

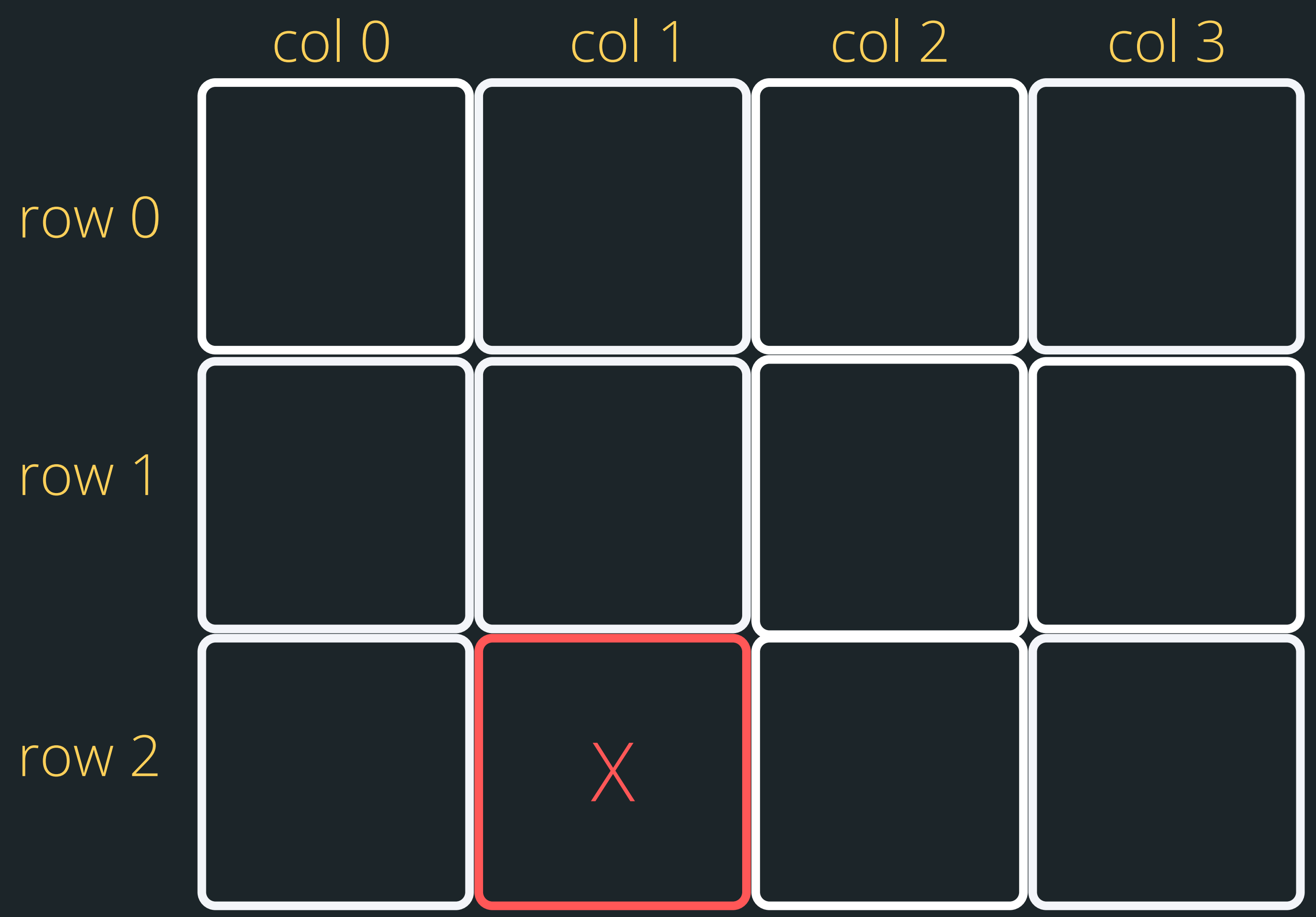
OUTSIDE WHILE

ROW = 2

INSIDE WHILE

COL = 1

```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

THIRD RUN AROUND THE SUN:

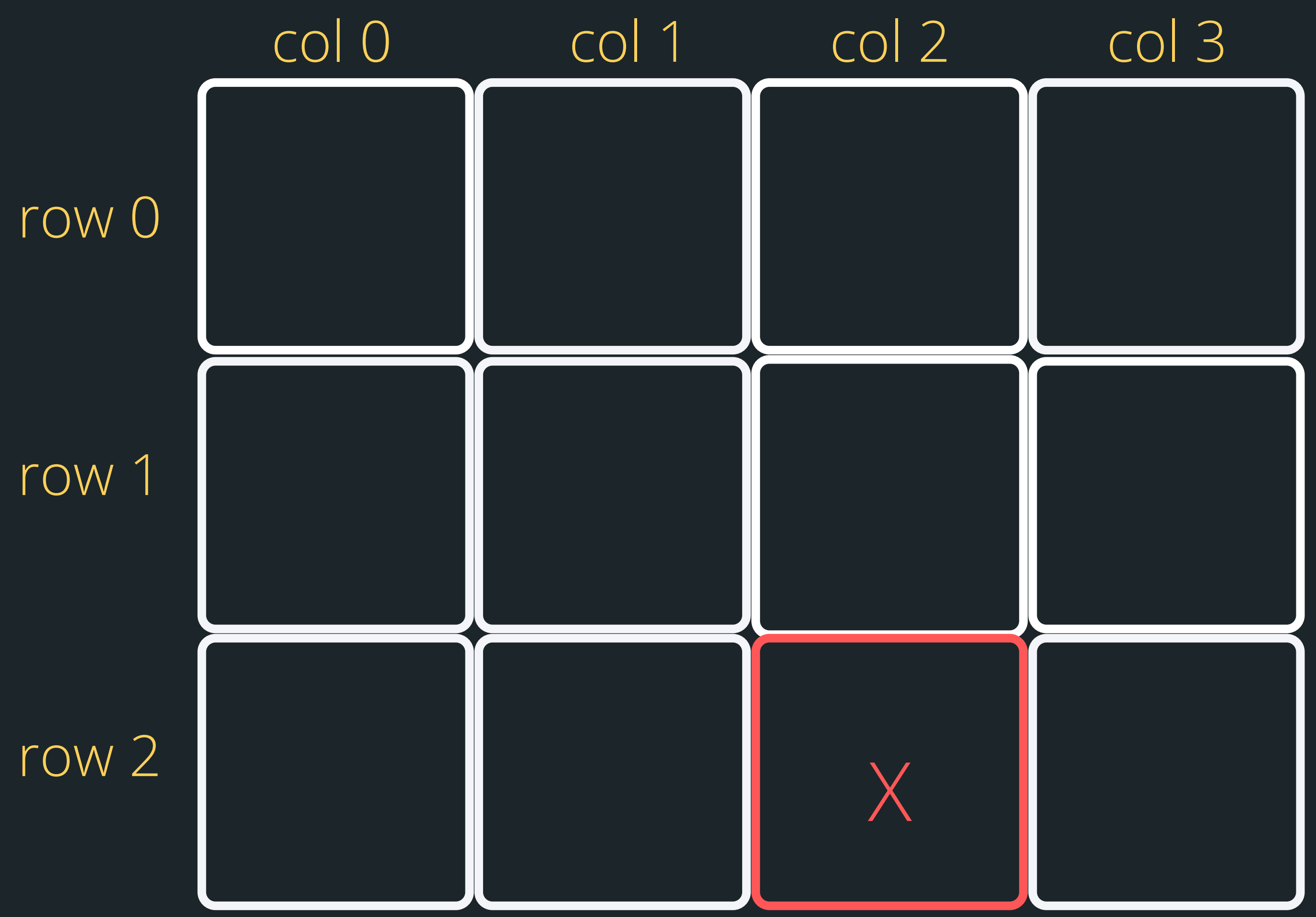
OUTSIDE WHILE

ROW = 2

INSIDE WHILE

COL = 2

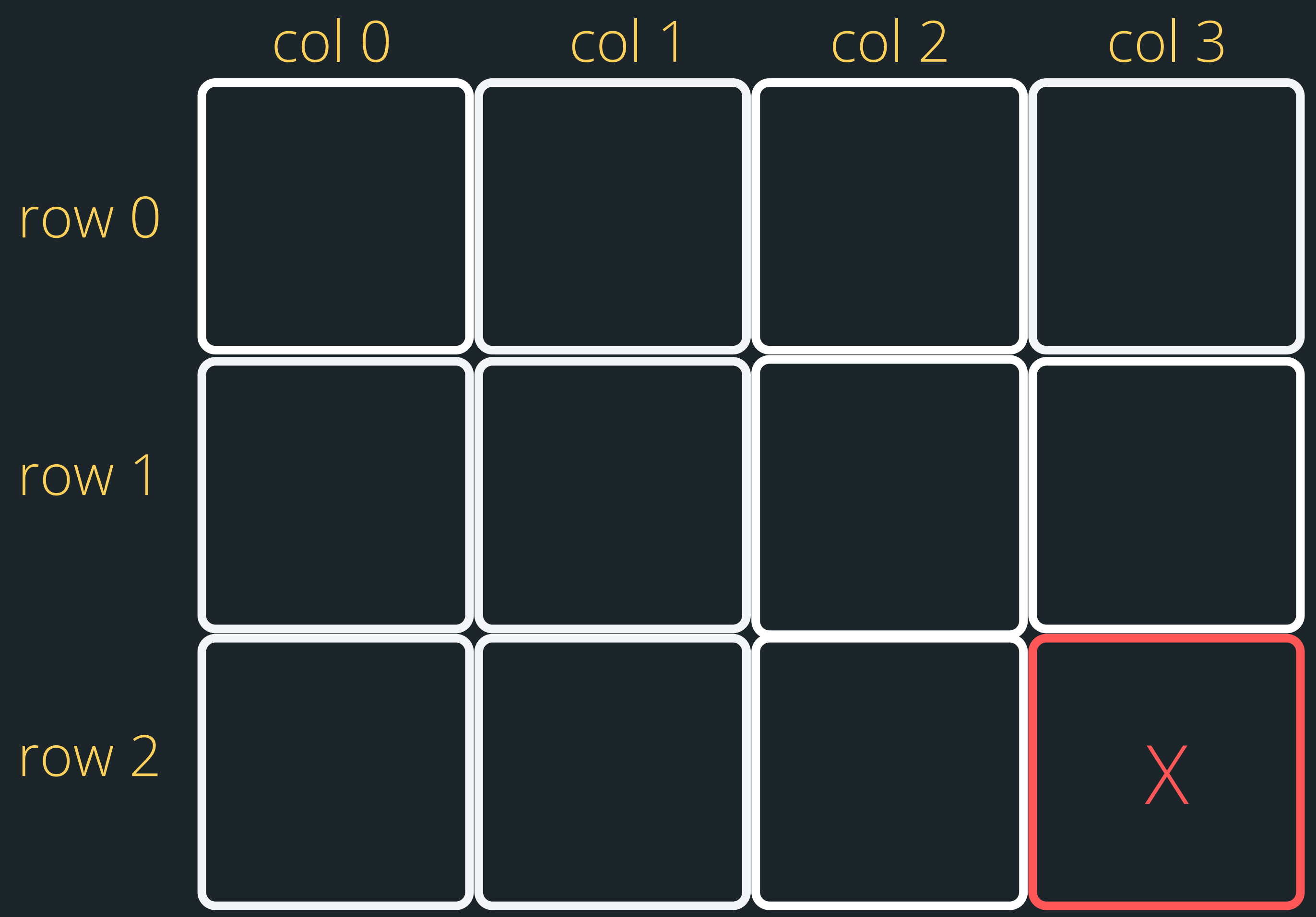
```
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



TRANSFER THIS TO AN ARRAY:

THIRD RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 2
INSIDE WHILE
COL = 3

```c
int array[3][4];
int row = 0;
while (row <= 3){
    int col = 0;
    while (col <= 4){
        printf("%d", array[row][col]);
        col++;
    }
printf("\n");
row++;
}
```



BREAK TIME

TIME TO STRETCH

There are five bags of gold that all look identical, and each has ten gold pieces in it. One of the five bags has fake gold in it. The real gold, fake gold, and all five bags are identical in every way, except the pieces of fake gold each weigh 1.1 grams, and the real gold pieces each weigh 1 gram. You have a perfectly accurate digital gram scale and can use it only once. How do you determine which bag has the fake gold?

LET'S WELCOME POINTERS INTO THE MIX

- A pointer is another variable that stores a memory address of a variable
- This is very powerful, as it means you can modify things at the source (this also has certain implications for functions which we will look at in a bit)
- To declare a pointer, you specify what type the pointer points to with an asterisk:

```
type_pointing_to *name_of_ variable;
```

  - For example, if your pointer points to an int:

    ```
    int *pointer;
    ```

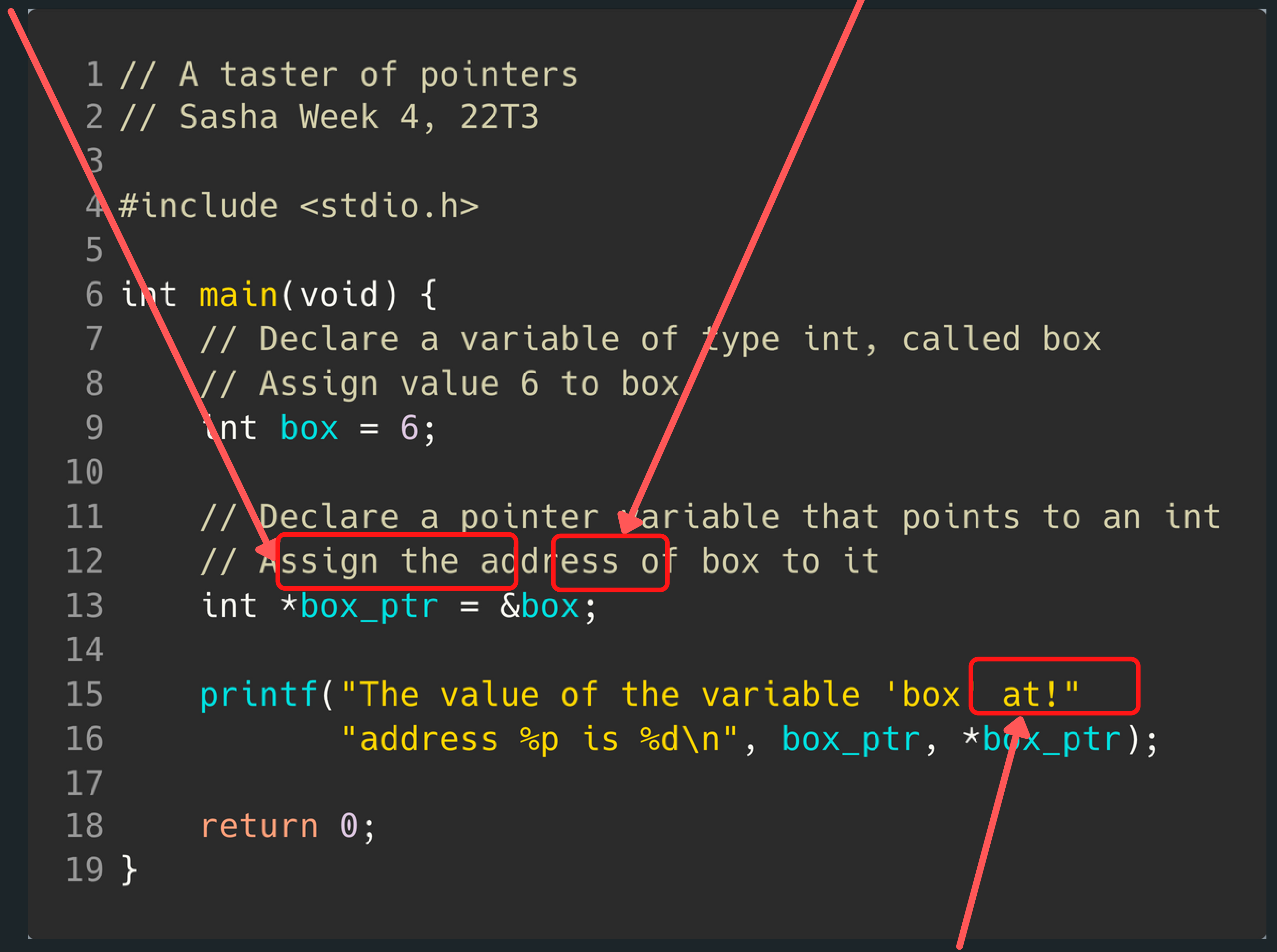
WHY DO WE NEED POINTERS?

- Pointers solve two common problems:
  - Remember how I said that when we pass some inputs into a function it actually makes a copy of that variable? Well, pointers kind of allow us to share information easier between sections of code without all that copying
  - Pointers also allow us to play with more complex data structures such as linked lists - coming in Week 7 and will really help with pointers :)

THERE ARE THREE PARTS TO A POINTER

*1. Declare a pointer with a * - this is where you will specify what type the pointer points to*

2. Initialise a pointer - assign the address to the variable with &

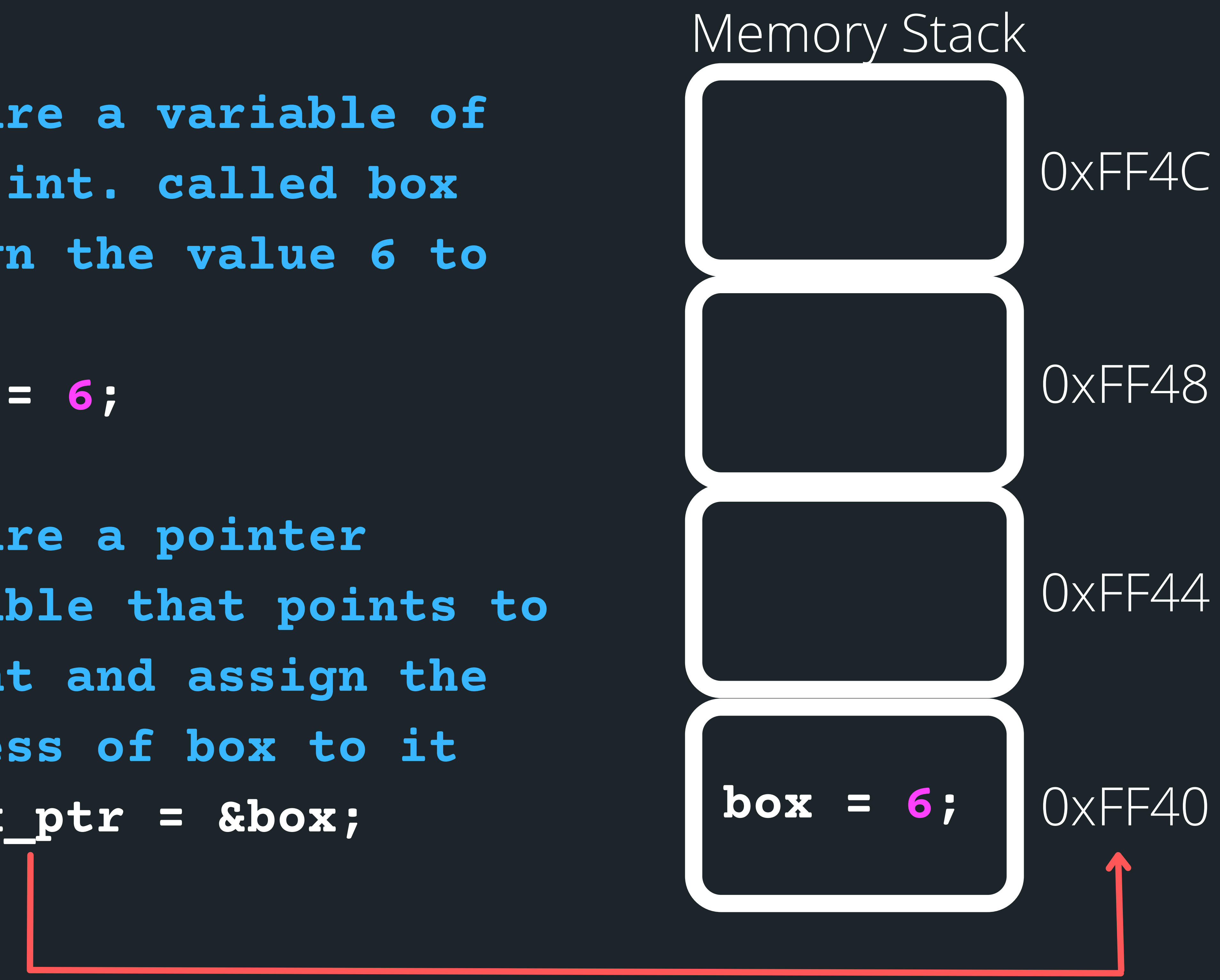


```c
// A taster of pointers
// Sasha Week 4, 22T3

#include <stdio.h>

int main(void) {
    // Declare a variable of type int, called box
    // Assign value 6 to box
    int box = 6;

    // Declare a pointer variable that points to an int
    // Assign the address of box to it
    int *box_ptr = &box;

    printf("The value of the variable 'box  at!"
           "address %p is %d\n", box_ptr, *box_ptr);

    return 0;
}
```

*3. Dereference a pointer -Using a * , go to the address that this pointer variable is assigned and find what is at that address*

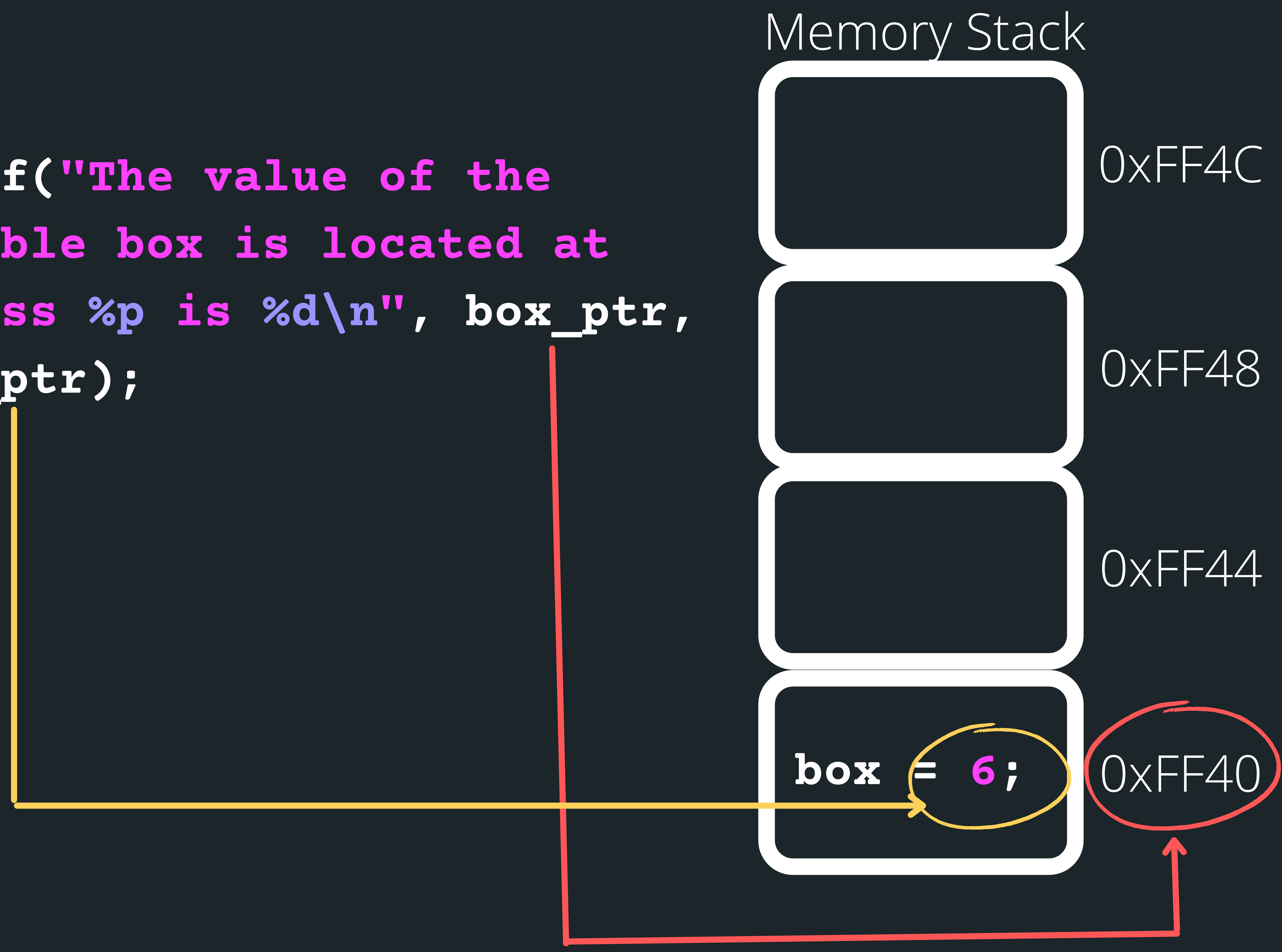
VISUALLY WHAT IS HAPPENING?

```
// Declare a variable of
// type int. called box
// Assign the value 6 to
// box
int box = 6;

// Declare a pointer
// variable that points to
// an int and assign the
// address of box to it
int *box_ptr = &box;
```



VISUALLY WHAT IS HAPPENING?

```
printf("The value of the
variable box is located at
address %p is %d\n", box_ptr,
*box_ptr);
```



YOU CAN HAVE A POINTER TO DIFFERENT VARIABLES

WHEN YOU DECLARE A POINTER, YOU WILL SPECIFY THE TYPE THAT IT POINTS TO FOLLOWED BY *

```
// Declare a variable of type int called box
// Assign the value 6 to box
int box = 6;

// Declare a pointer variable that points to
// an int and assign the address of box to it
int *box_ptr = &box;

// Declare a variable of type double called box
// Assign the value 3.2 to box
double box = 3.2;

// Declare a pointer variable that points to
// a double and assign the address of box to it
double *box_ptr = &box;

// Declare a variable of type char called box
// Assign the value 'c' to box
char box = 'c';

// Declare a pointer variable that points to
// a double and assign the address of box to it
char *box_ptr = &box;
```

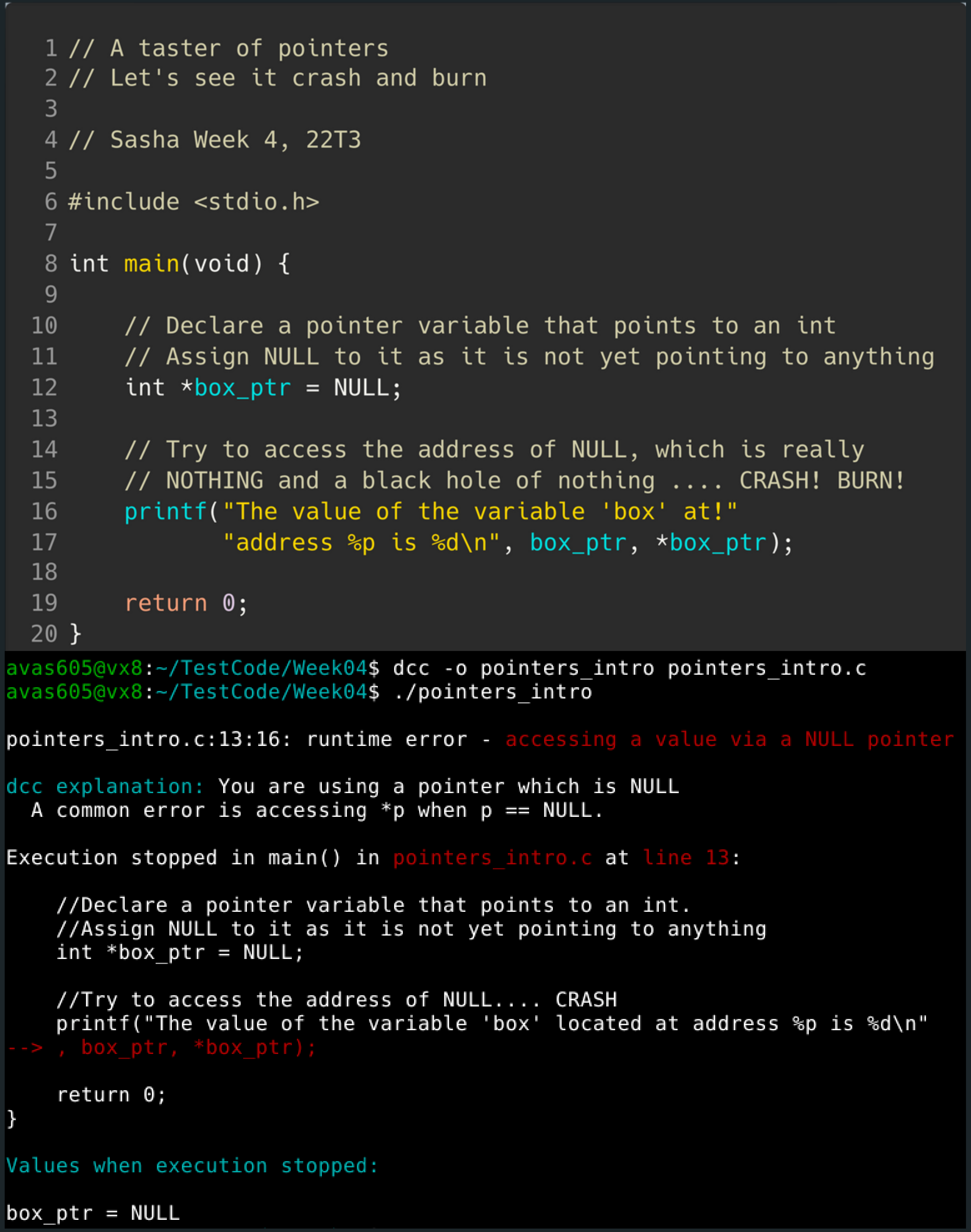
INITIALISING POINTERS WHEN YOU DON'T HAVE ANYTHING TO INITIALISE THEM WIHT YET

NULL POINTER

- Pointers are just another type of variable, and just like our other variables it should be initialised after it is declared.
- Generally, we will initialise a pointer, by pointing it at a variable
- If we need to initialise a pointer that is not yet pointing to anything, we use: **`NULL`**
- This is a special word in a C library which is #define
- It is basically a value of 0, but for a pointer, we use this keyword **`NULL`**

WHAT HAPPENS IF YOU FORGET TO EVER GIVE THIS NULL POINTER AN ACTUAL ADDRESS WITH SOMETHING AND THEN TRY AND DEREFERENCE A NULL POINTER?

COMPILES THAN CHAOS...



```c
 1 // A taster of pointers
 2 // Let's see it crash and burn
 3
 4 // Sasha Week 4, 22T3
 5
 6 #include <stdio.h>
 7
 8 int main(void) {
 9
10     // Declare a pointer variable that points to an int
11     // Assign NULL to it as it is not yet pointing to anything
12     int *box_ptr = NULL;
13
14     // Try to access the address of NULL, which is really
15     // NOTHING and a black hole of nothing .... CRASH! BURN!
16     printf("The value of the variable 'box' at!"
17            "address %p is %d\n", box_ptr, *box_ptr);
18
19     return 0;
20 }
```

```
avas605@vx8:~/TestCode/Week04$ dcc -o pointers_intro pointers_intro.c
avas605@vx8:~/TestCode/Week04$ ./pointers_intro

pointers_intro.c:13:16: runtime error - accessing a value via a NULL pointer

dcc explanation: You are using a pointer which is NULL
  A common error is accessing *p when p == NULL.

Execution stopped in main() in pointers_intro.c at line 13:

    //Declare a pointer variable that points to an int.
    //Assign NULL to it as it is not yet pointing to anything
    int *box_ptr = NULL;

    //Try to access the address of NULL.... CRASH
    printf("The value of the variable 'box' located at address %p is %d\n"
--> , box_ptr, *box_ptr);

    return 0;
}

Values when execution stopped:

box_ptr = NULL
```

TIME TO CODE AND SEE A POINTER IN ACTION!

`pointers_intro.c`

- Best way to learn about pointers is to start using them

Feedback please!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://www.menti.com/alkfepqss1ze

WHAT DID WE LEARN TODAY?

ASSIGNMENT 1 IS RELEASED

LIVESTREAM on Friday 12:30pm:

RECAP 1D ARRAYS

lowest_number.c

AN ARRAY OR ARRAYS (2D)

print_2Darray.c

TASTE OF POINTERS

hello_pointer.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au