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
- Strings
- Array of structs (visual)
- Array of arrays - this is aspirational....

most likely tomorrow....

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/26T1/CODE/WEEK_4/

ASSIGNMENT 1

RELEASED TODAY

- Assignment 1 will be released on Tuesday after the lecture
- 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

A QUICK RUNDOWN

- 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
- A quick rundown of the setup will be shown on a short video

REVISION SESSIONS

IN-PERSON AND HYBRID

Week 4 Revision Sessions:

(Brass) Monday 9th Mar 2pm-4pm

(Online) Thursday 12th March 12pm-2pm

- You can sign up for the revision sessions by following this link:

https://buytickets.at/comp1511unsw/2103137

- Vote on the topics you want rehashed!

https://discourse02.cse.unsw.edu.au/26T1/COMP1511/t/week-4-revision-sessions/90
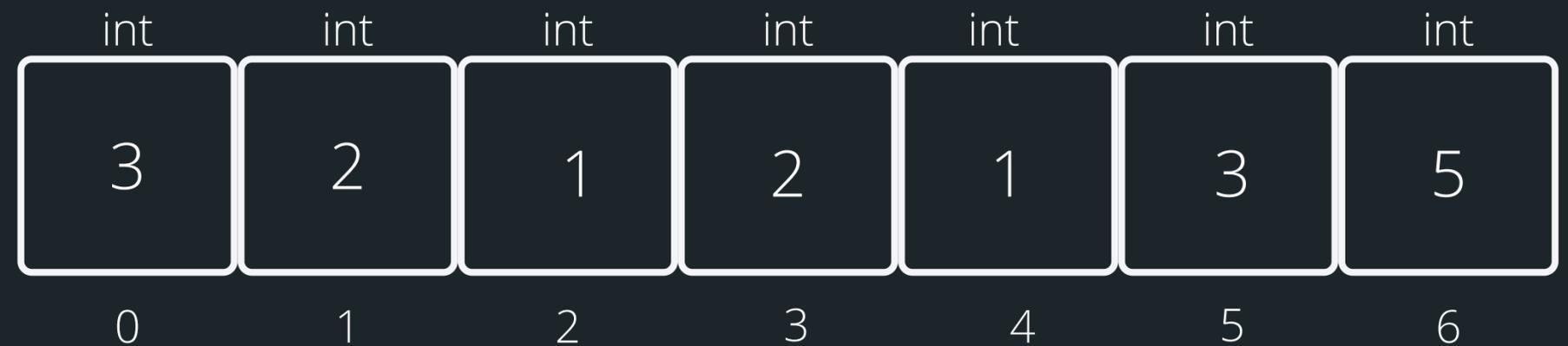
RECAP OF ARRAYS

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:

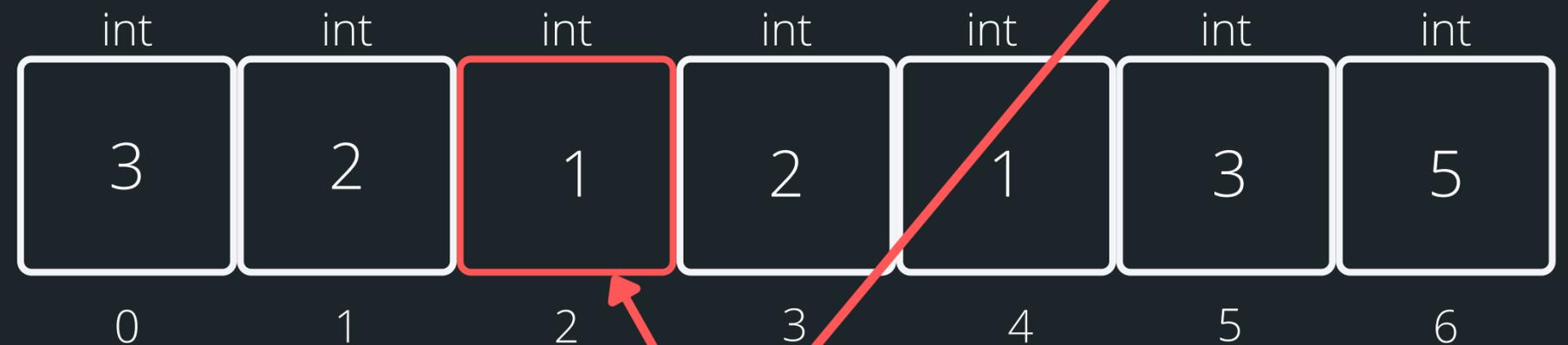| int	int	int	int	int	int	int
3	2	1	2	1	3	5
0	1	2	3	4	5	6

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};
```

int	int	int	int	int	int	int
3	2	1	2	1	3	5
0	1	2	3	4	5	6

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 count the number of times a 6 or a 7 occurs and output the count to the user.

`count_67.c`

Problem: A user is asked to enter 10 numbers. We will then go through these numbers and find the sum of the odd numbers only.

`sum_odd.c`

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:

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

char	char	char	char	char	char
h	e	l	l	o	\0
0	1	2	3	4	5

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
1 #include <stdio.h>
2
3 #define MAX_LENGTH 15
4
5 int main(void) {
6     // Declare an array where you will place the string
7     char array[MAX_LENGTH];
8
9     printf("Type in a string to echo: ");
10    // Read in the string into the array until Ctrl+D is
11    // pressed, which is indicated by the NULL keyword
12    while (fgets(array, MAX_LENGTH, stdin) != NULL) {
13        printf("The string is: \n");
14        printf("%s", array);
15        printf("Type in a string to echo: ");
16    }
17    return 0;
18 }
```

SOME OTHER INTERESTING STRING FUNCTIONS

<STRING.H> STANDARD LIBRARY

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
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_LENGTH 15
5
6 int main(void) {
7     // Declare an array
8     char word_array[MAX_LENGTH];
9
10    // Example using strcpy to copy from one string
11    // to another (destination, source)
12    strcpy(word_array, "Jax");
13    printf("%s\n", word_array);
14
15    // Example using strlen to find string length
16    // returns the int length NOT including '\0'
17    int length = strlen("Sasha");\n
18    printf("The size of string 'Sasha' is %d chars\n", length);
19
20    // Example using strcmp to compare two strings character
21    // by character - function will return:
22    // 0 = two strings are equal
23    // other int if not the same
24
25    int compare_string = strcmp("Jax", "Juno");
26    printf("The two strings are the same: %d\n", compare_string);
27
28    compare_string = strcmp(word_array, "Jax");
29    printf("The two strings are the same: %d\n", compare_string);
30    return 0;
31 };
```

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?

YOU CAN HAVE AN ARRAY OF ANYTHING
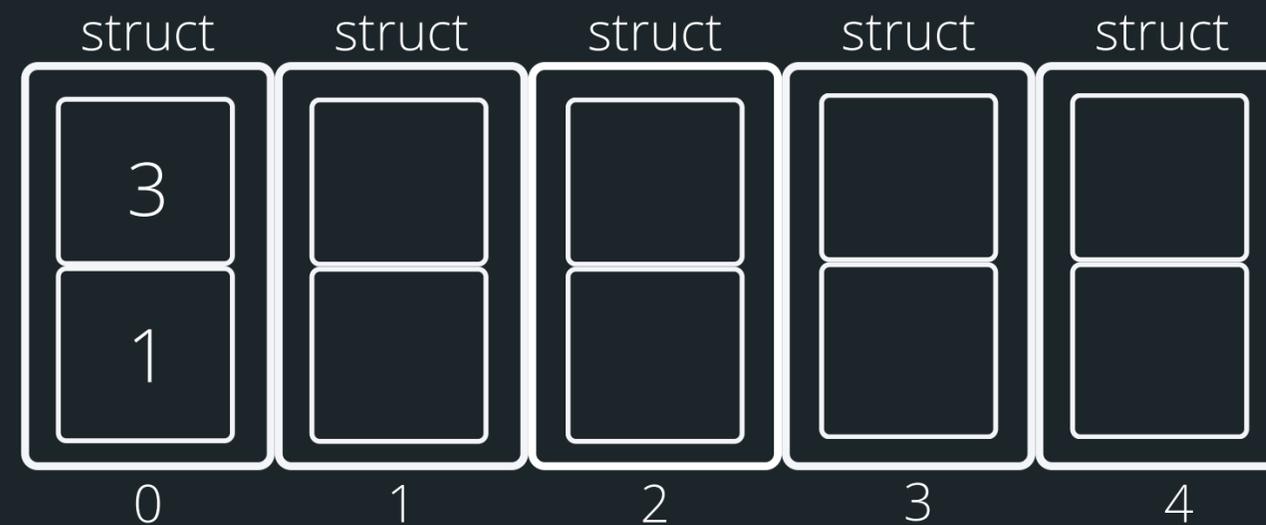
AN ARRAY OF STRUCTS

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 tea consumption for a week. What if I want to do this for a month (a week at a time)?

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

	col 0	col 1	col 2	col 3	col 4	col 5	col 6
row 0							
row 1							
row 2							
row 3							

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 with our while inside a while?

```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
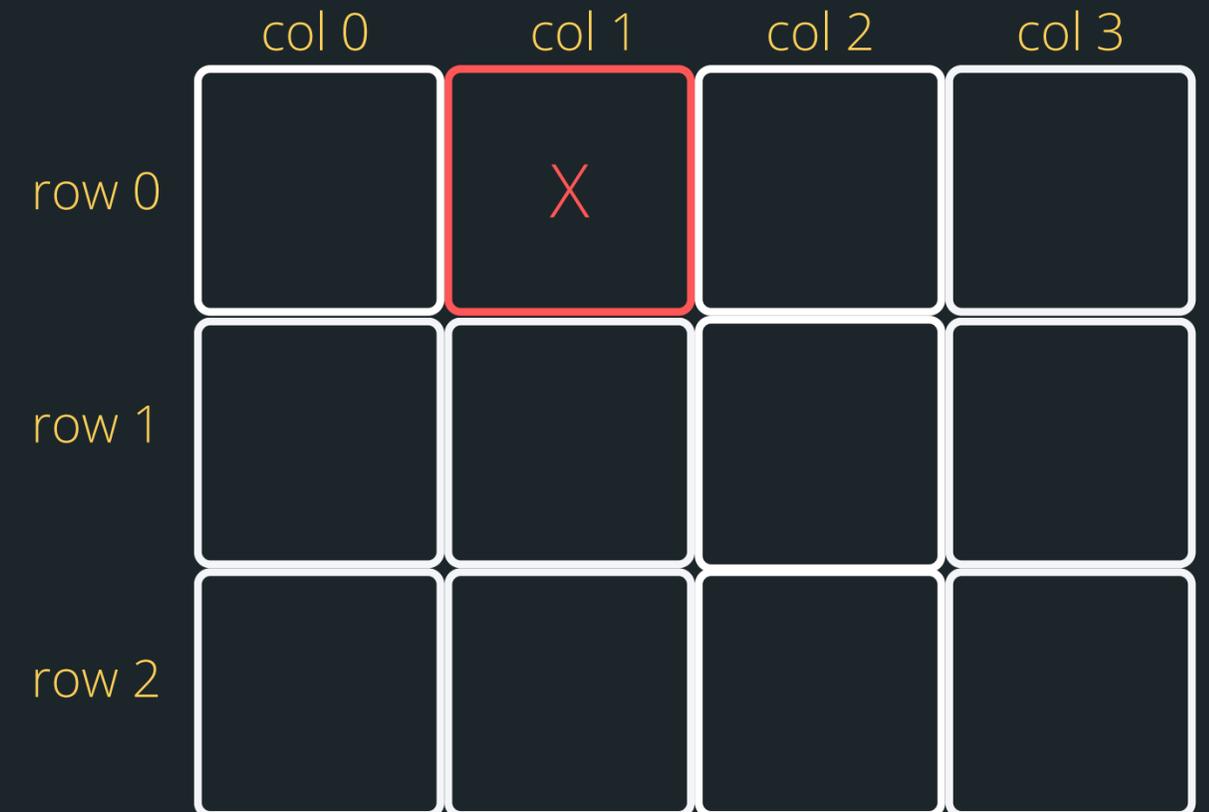INSIDE WHILE
COL = 0

```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++;
}
```

TRANSFER THIS TO AN ARRAY:

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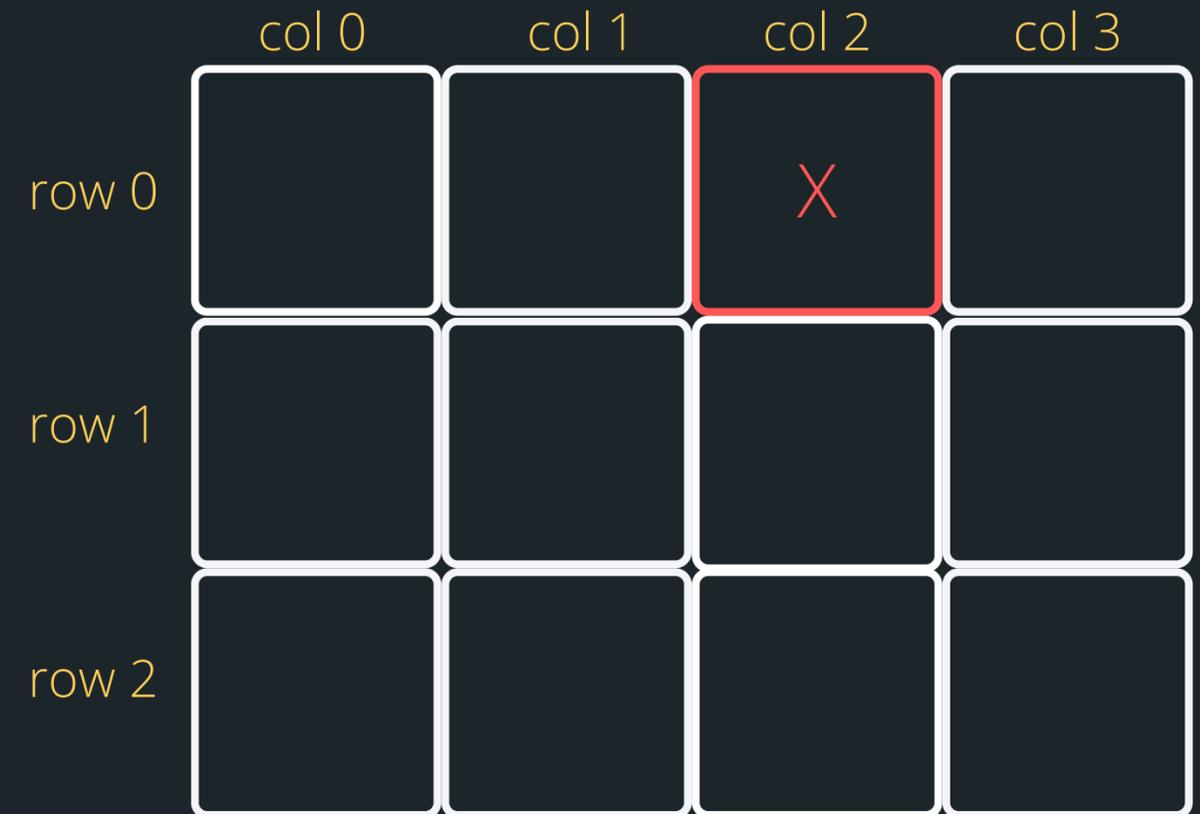INSIDE WHILE
COL = 1

```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++;
}
```

	col 0	col 1	col 2	col 3
row 0		X		
row 1				
row 2				

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

TRANSFER THIS TO AN ARRAY:

FIRST RUN AROUND THE SUN:
OUTSIDE WHILE
   ROW = 0
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++;
}
```

	col 0	col 1	col 2	col 3
row 0				X
row 1				
row 2				

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

	col 0	col 1	col 2	col 3
row 0				
row 1	X			
row 2				

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

	col 0	col 1	col 2	col 3
row 0				
row 1		X		
row 2				

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

	col 0	col 1	col 2	col 3
row 0				
row 1			X	
row 2				

TRANSFER THIS TO AN ARRAY:

SECOND RUN AROUND THE SUN:
OUTSIDE WHILE
ROW = 1
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++;
}
```

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

	col 0	col 1	col 2	col 3
row 0				
row 1				
row 2		X		

TRANSFER THIS TO AN ARRAY:

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

```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++;
}
```

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++;
}
```

	col 0	col 1	col 2	col 3
row 0				
row 1				
row 2				X

PROBLEM TIME

- Let's try our hand at using a 2D array to solve a problem!
- Declare and create a 3 x 3 two-dimensional array of integer numbers with the numbers read in from the user. Then loop through the two-dimensional array, printing out the values in the first row followed by those in the second row and so on.
    - Now loop through the array to count the number of even numbers in the 2D array

`2D_Arrays.c`

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://forms.office.com/r/S3M9ahJhhh

WHAT DID WE LEARN TODAY?

ASSIGNMENT 1
IS RELEASED

RECAP 1D
ARRAYS

count_67.c

sum_odd.c

STRINGS

string.c
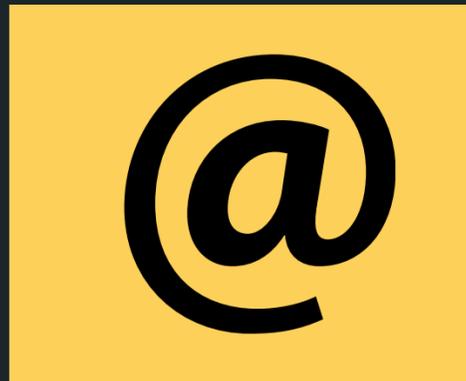
AN ARRAY OR
ARRAYS (2D)

2D_array.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au

@