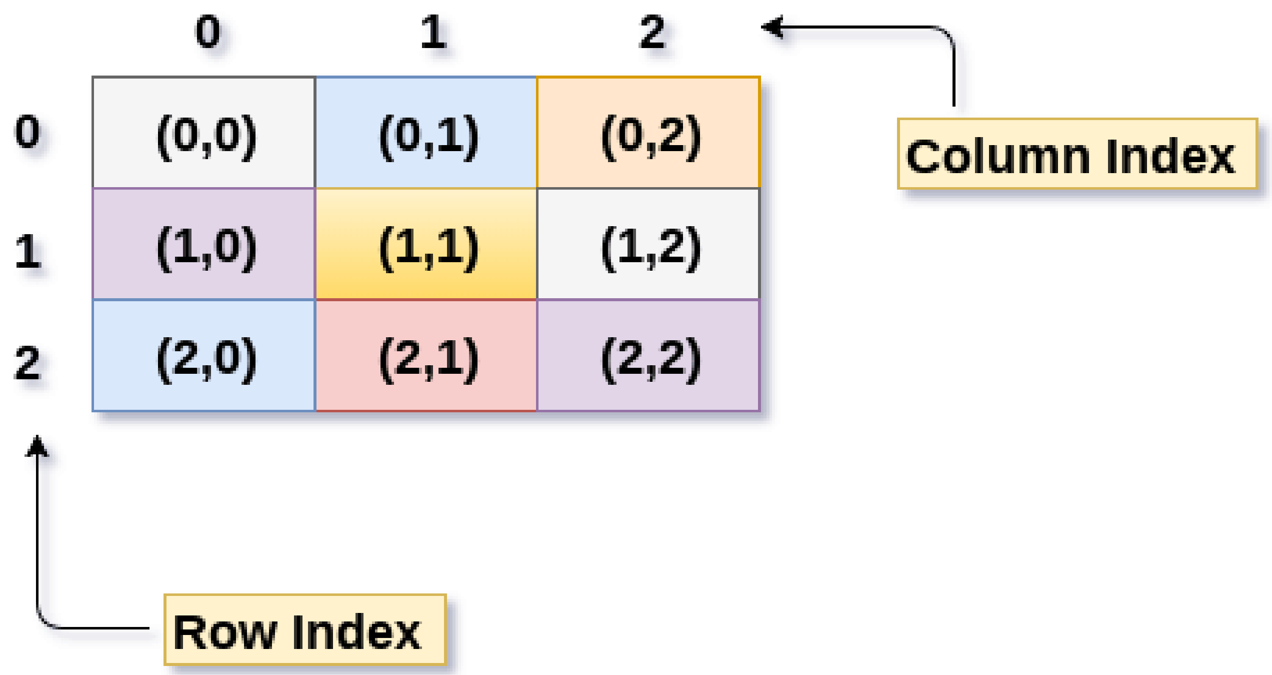
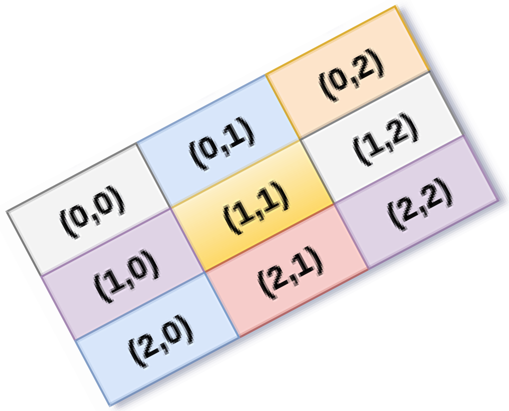
ALGORITHM AND COMPUTATIONAL THINKING 2

WEEK 4 – 2D Arrays



Session objectives

- ✓ **Use 2D arrays** to represent and manipulate grids of data
- ✓ **Pass** array 2D to **functions**
- ✓ Apply the **top-down design** to array 2D problems



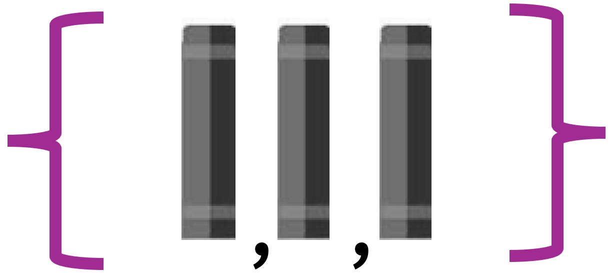
EXPLAIN

Multidimensional Arrays

MORE INFO

A multidimensional array is basically an array of arrays…

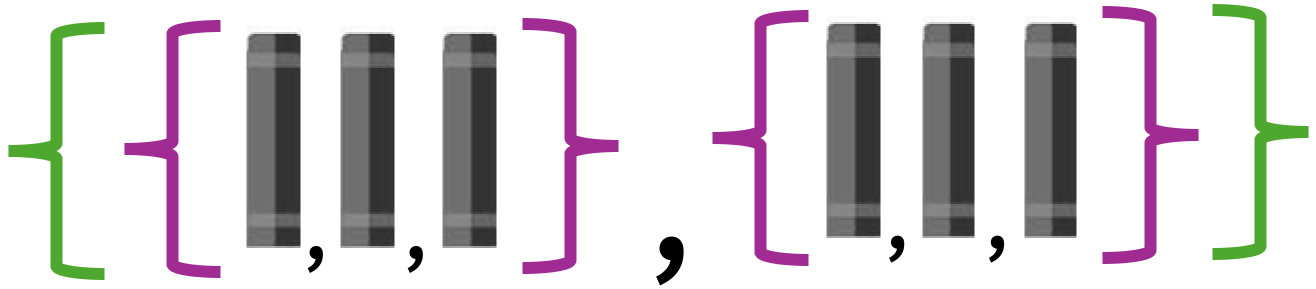
Single dimension arrays



```
{85, 90, 78}
```

*A single dimension array contain **primitive values***
Such as integer, boolean, double..

Two-Dimensional Arrays



```
{ {85, 90, 78}, {44, 2, 103} }
```

A 2D array, also known as a matrix, is a two-dimensional array

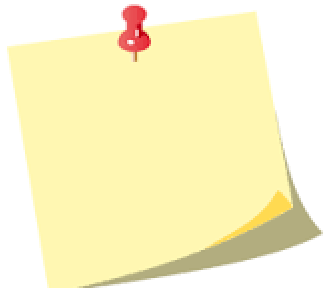
REFLECT

Think about a **real-life situation**

That can be represented with a 2D array

```
{
      {00, 00, 00},
      {00, 90, 00},
      {00, 44, 99},
      {00, 44, 99},
}
```

*What kind of problem in real life can required such a **2D data structure**?*



1 IDEA =1 PAPERS PER TEAM

*Bring your papers to the **white board** !*

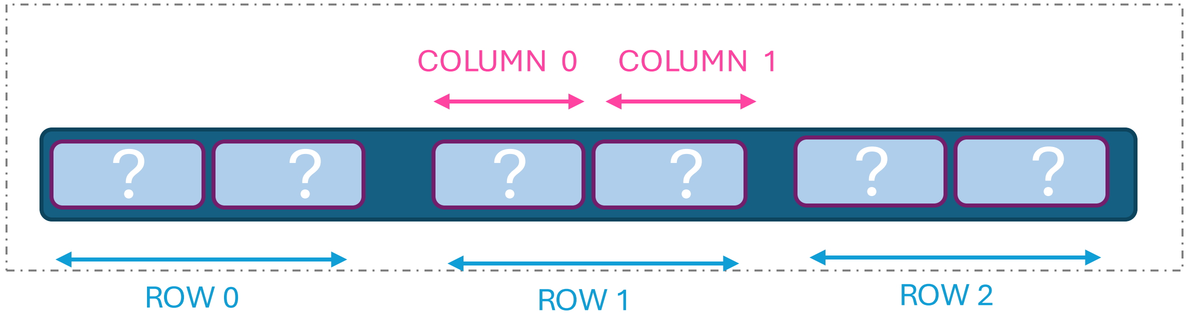
EXPLAIN

2D Arrays in **memory**

*Computers have **linear memory** : array 2D are stored one **row following another.***

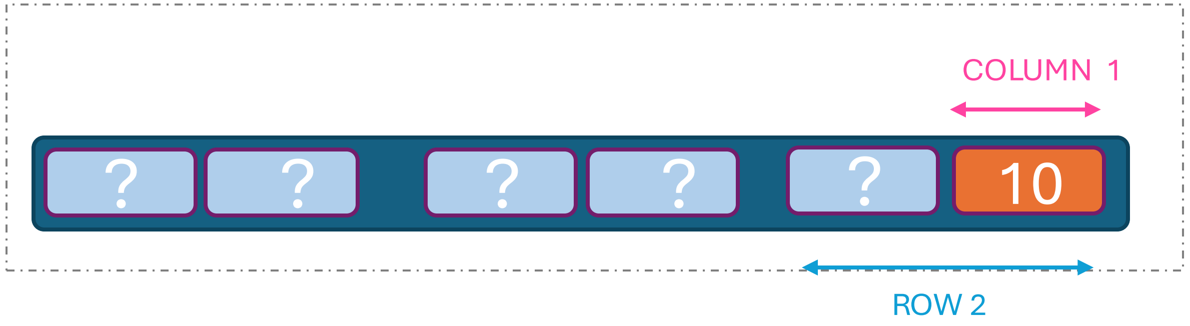
✓ When a 2D array is **declared**: memory is linearly assigned, **row by row**:

```
// 1 - Declare the 2D array
int prices[3][2] = {}
```



✓ To access an element of a 2D array, we must specify the index number of both the **row** and **column :**

```
// 2 - Change a cell value
prices[2][1] = 10;
```



Q1

What will this code print?

```
int matrix[3][3] = { {1, 4, 2}, {3, 6, 8}, {5, 9, 0}};

printf("%d", matrix[0][2]);
```

A. 2

B. 4

C. 5

D. 9

ANSWER

What will this code print?

COLUMN 2

```
int matrix[3][3] = { {1, 4, 2}, {3, 6, 8}, {5, 9, 0}};
```

ROW 0

```
printf("%d", matrix[0][2]);
```

ROW 0 COLUMN 2

[✓ A.] 2

B. 4

C. 5

D. 9

EXPLAIN

Matrix 2D Array representation

*A 2D array can also be represented with a **matrix***

number of **rows**
number of **Columns**

```
int scores[2][3] = { {1,2,3}, {4,5,6} };
```

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	2	3
ROW 1	4	5	6

Q2

Fill up the gaps to match with the table

```
int matrix[3][3] = {0};
matrix[____][___] = 77;
```

A. `matrix[2][1]`

B. `matrix[1][2]`

C. `matrix[2][3]`

D. `matrix[3][2]`

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	0	0	0
ROW 1	0	0	0
ROW 2	0	77	0

ANSWER

Fill up the gaps to match with the table

```
int matrix[3][3] = {0};
matrix[____][___] = 77;
```

[✓ A.] matrix[2][1]

B. matrix[1][2]

C. matrix[2][3]

D. matrix[3][2]

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	0	0	0
ROW 1	0	0	0
ROW 2	0	77	0

EXPLAIN

Loop Through a 2D Array

To loop through a 2D array, you need one loop on both its rows, and its columns dimension

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

for (int i = 0; i < 2; i++) {
  for (int j = 0; j < 3; j++) {
    printf("%d\n", matrix[i][j]);
  }
}
```

```
1
4
2
3
6
8
```

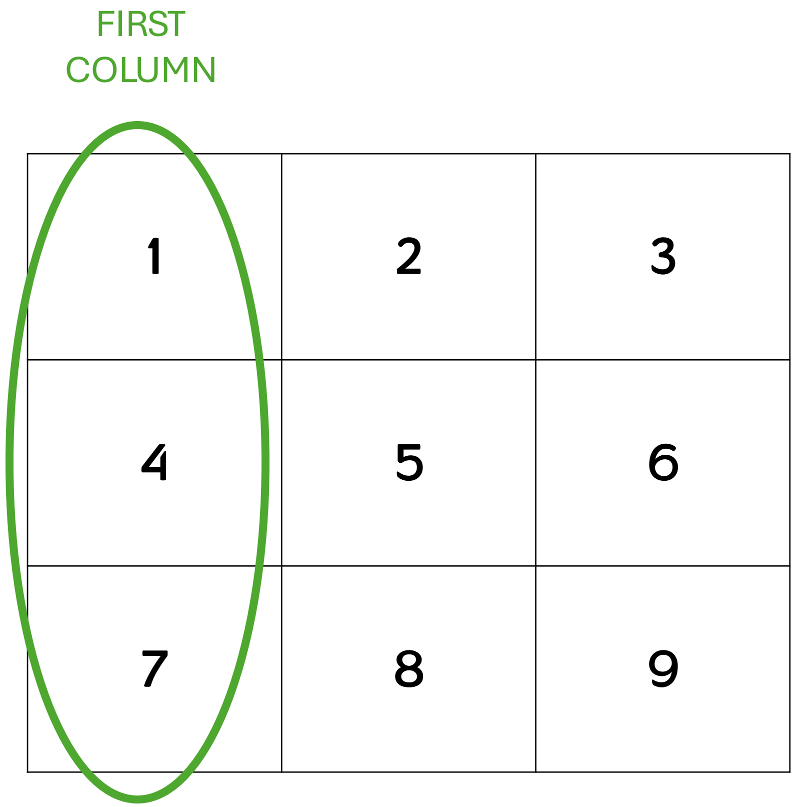
CHALLENGE

Print the **first column** elements

Write the code to print all elements of the first column

PAPER ONLY

```
int matrix[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```



1	2	3
4	5	6
7	8	9

```
1 4 7
```

Expected outcome on console

ANSWER

Print the **first column** elements

Write the code to print all elements of the first column

```c
int matrix[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

for (int row = 0; row < 3; row++) {
     printf("%d ", matrix[row][0]);
}
```

```
1 4 7
```

Expected outcome on console

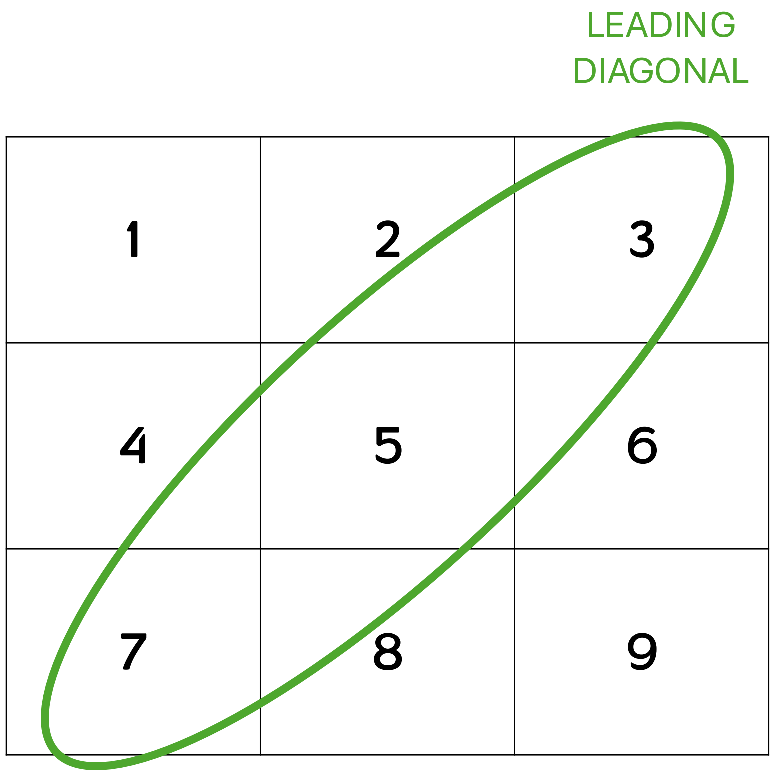
CHALLENGE

Print the **first column** elements

Write the code to print all elements of the leading diagonal

PAPER ONLY

```
int matrix[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```



```
3 5 7
```

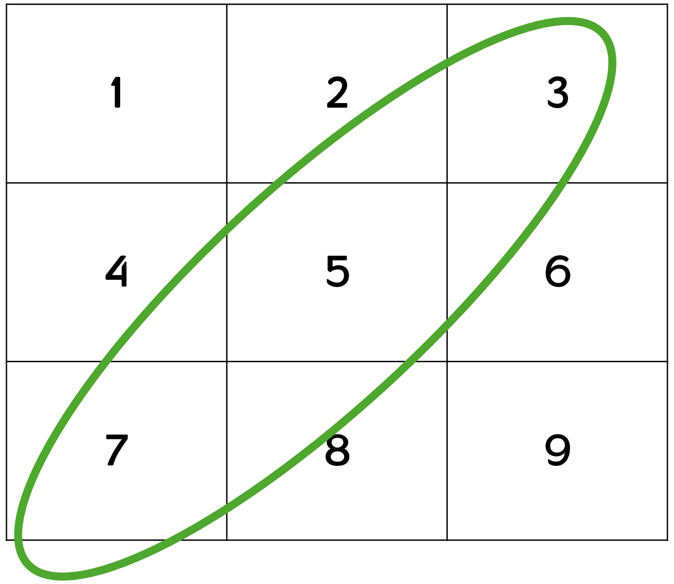
Expected outcome on console

ANSWER

Print the **first column** elements

Write the code to print all elements of the leading diagonal

```
int matrix[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

for (int row = 0; row < 3; row++) {
    printf("%d ", matrix[row][2 -row]);
}
```



```
3 5 7
```

Expected outcome on console

REMINDER

Passing **an array** to a function

Remember that in C, arrays are not passed by value — they **decay to a reference**

```
int main()  {

    int numbers[3] = {10, 20, 30};

    // pass array to the function
    compute(numbers, 3);
}
```

```
void compute(int values[], int size) {

   // do something


}
```

The compiler knows the size of the array

The compiler does NOT know the size of the array

We pass it as a second parameter

EXPLAIN

Passing **a 2D array** to a function

But when passing array 2D to functions, the **compiler** needs to know **the size of each row**

So that he can compute where matrix[i][j] is in memory...

```
void compute(int matrix[][], int cols, int rows) {

}
```

✗ Compiler needs to know the **number of columns** at least

```
void compute(int matrix[][4], int rows) {

}
```

✓ **Number of columns (4) is fixed**

Only number of rows needs to be passed

```
void compute(int rows, int columns, int matrix[row][columns],) {

}
```

✓ **Both row and column sizes are fixed.**

Compiler uses them at runtime

NOTE: C99 compilers and later only !

CHALLENGE

Passing **a 2D array** to a function

PAPER ONLY

Let's pass the matrix to the function has7, with the appropriate row and columns information

```
int main() {
  int matrix[2][4] = { {0, 0, 0, 0}, {0, 7, 0, 0} };
  printf("%d\n", has7(2, 4, matrix);

  return 0;
}
```

```
bool has7(int rows, columns, matrix[rows][columns]) {
    // Return true if the matrix contains at least one 7


}
```

Complete the missing code

REFLECTION

The highest **row Sum**

PAPER ONLY

We want to know which row has the highest sum

The row 0 has the highest sum

10	15	5	SUM = 30
5	5	0	SUM = 10
7	0	0	SUM = 7

*You need to **break down** this problem into small tasks by defining functions:*

1 – Identify the **High level steps**

2 – Sketch out the **functions you want to create**

(the inputs, the output , the function name)

3 – Comment each function of block of code *(but don't code it)*

ANSWER

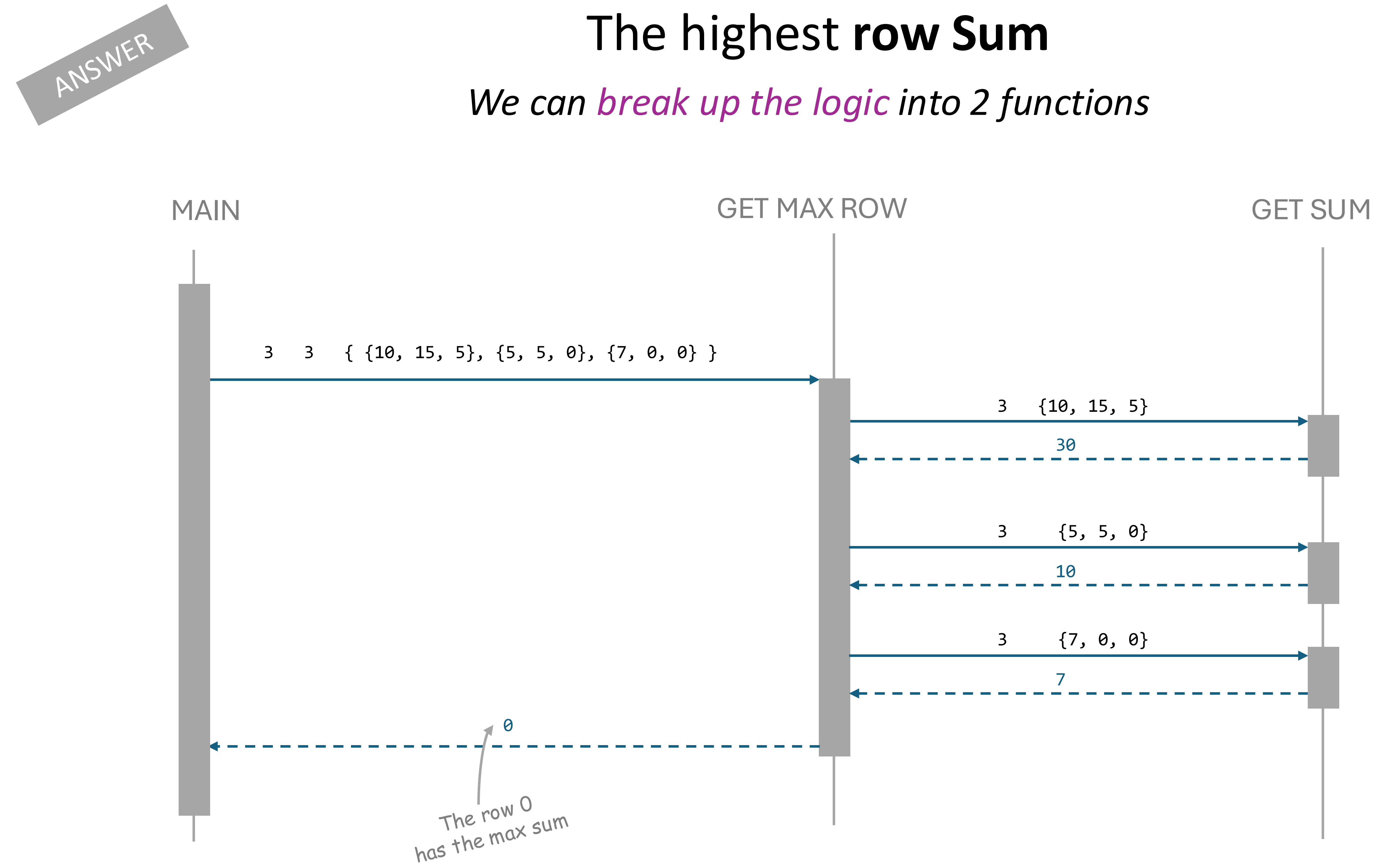
The highest **row Sum**

We can break up the logic into 2 functions

Function	**Parameters**	**Return**	**Example**
getMaxRow	int rows int columns int matrix [rows] [columns]	The **index** of the row with the **highest sum**	INPUT 3 3 { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}} OUPUT 2
getSum	int size int [size]	**Sum of numbers** on given array	INPUT 3 {1, 2, 3}; OUPUT 6

The highest **row Sum**

We can break up the logic into 2 functions



ANSWER

The highest **row Sum**

We can break up the logic into 2 functions

MAIN

```
int main() {

  int matrix[4][4] = {
      {15, 5, 10, 10},
      {7, 5, 0, 0},
      {6, 0, 7, 1},
      {0, 2, 0, 1}
  };

  int maxSum = getMaxRow(4, 4, matrix);

  printf("%d\n", maxSum);
  return 0;
}
```

GET MAX ROW

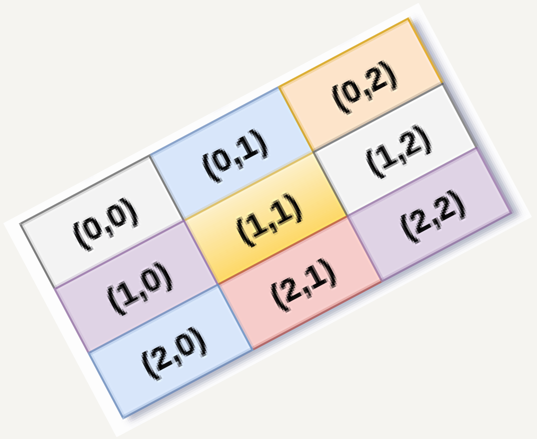
```
int getMaxRow(int rows,int columns, int
matrix[rows][columns]) {

  int maxSum = getSum(columns, matrix[0]);
  int maxSumRow = 0;

  for (int row = 1; row < rows; row++) {
    int rowSum = getSum(columns, matrix[row]);

    if (rowSum > maxSum) {
      maxSum = rowSum;
      maxSumRow = row;
    }
  }
  return maxSumRow;
}
```

GET SUM

```
int getSum(int size,int numbers[size])
{

  int sum = 0;
  for (int i = 0; i < size; i++) {
    sum += numbers[i];
  }

  return sum;

}
```

🥇 What you **know now** 🥇

- ✓ **Use 2D arrays** to represent and manipulate grids of data
- ✓ **Pass** array 2D to **functions**
- ✓ Apply the **top-down design** to array 2D problems



Go further after the class…

Multidimension array C
https://www.w3schools.com/c/c_arrays_multi.php

Understand array decay in C
https://www.geeksforgeeks.org/array-decay-in-c/