Féidearthachtaí as Cuimse
Infinite Possibilities

Semester 2
Week 3 - Tutorial

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Programming - Week 3 – 10th February 2025

Overview

- Passing 1D and 2D Arrays to Functions
- Labwork Sample Solutions
- Mandatory Question

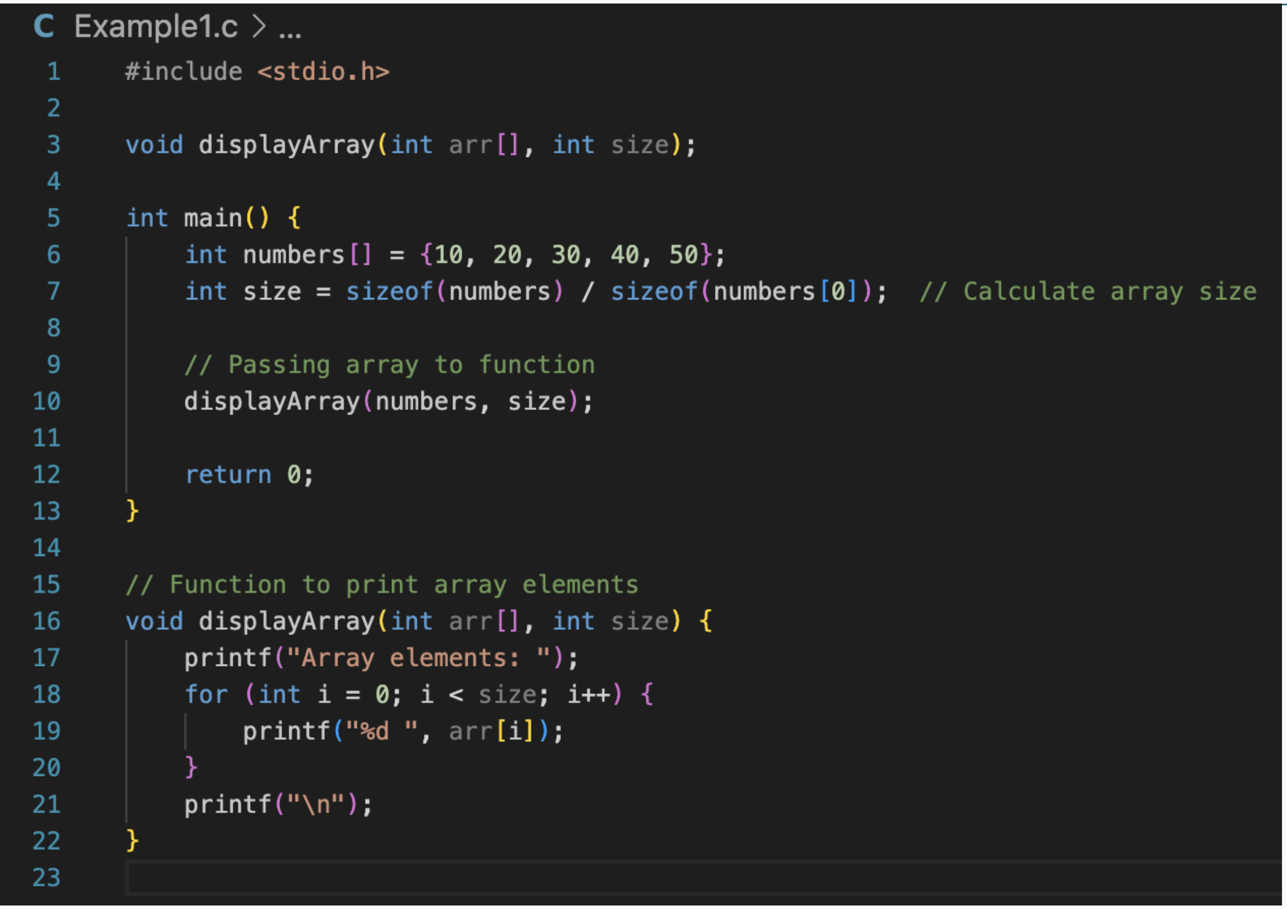
Passing a 1D Array to a Function

- A **1-D array** can be passed to a function using only its **name** (which acts as a pointer to the first element). This allows the function to directly access and modify the array elements.

Basic Operation

- When an array is passed to a function the function receives a **pointer** to the first element of the array.
- **No copy** of the array is made, so changes inside the function affect the original array.
- The array size is **not automatically passed**, so we must explicitly provide it if needed.

Passing an Array to a Function

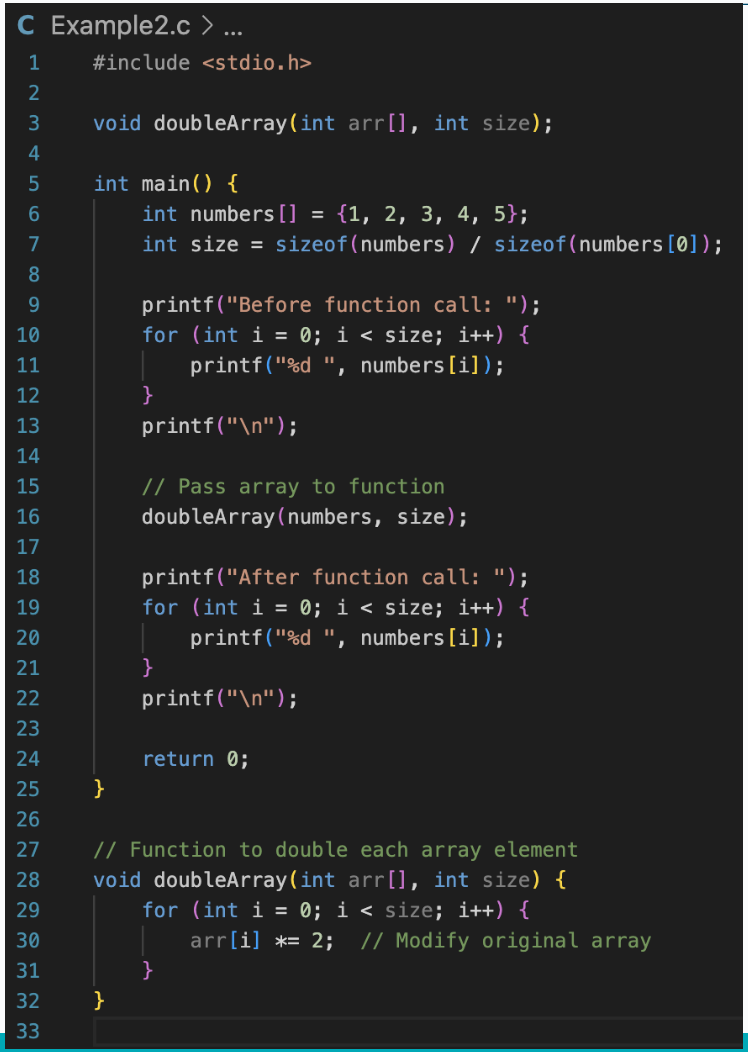


C Example1.c > ...

```c
#include <stdio.h>

void displayArray(int arr[], int size);

int main() {
    int numbers[] = {10, 20, 30, 40, 50};
    int size = sizeof(numbers) / sizeof(numbers[0]);  // Calculate array size

    // Passing array to function
    displayArray(numbers, size);

    return 0;
}

// Function to print array elements
void displayArray(int arr[], int size) {
    printf("Array elements: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

```

- The function displayArray() receives an **array name** (arr[]), which acts as a pointer to numbers[].
- The function can access all elements without returning anything.
- The total size of the array **arr** (20 bytes) is divided by the size of a single element in the array (4 bytes). This gives the number of elements in the array, which is **20/4 = 5**

Modifying the Array

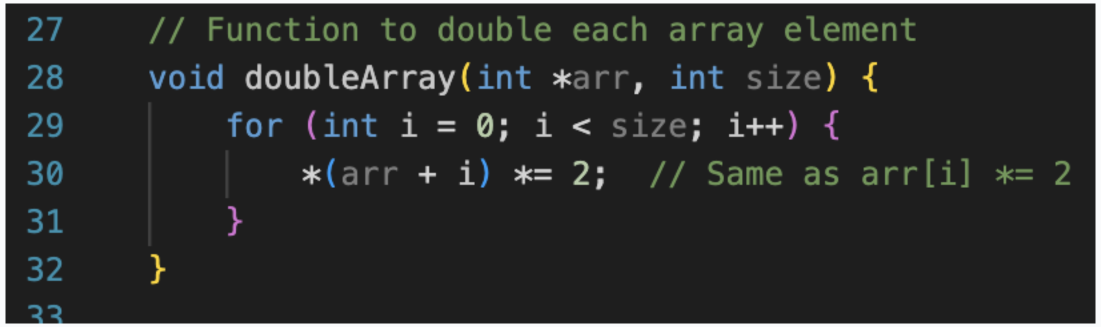


Example2.c > ...

```c
#include <stdio.h>

void doubleArray(int arr[], int size);

int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);

    printf("Before function call: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    // Pass array to function
    doubleArray(numbers, size);

    printf("After function call: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    return 0;
}

// Function to double each array element
void doubleArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] *= 2;  // Modify original array
    }
}

```

- Create a function to double each value in the array.
- The function modifies the **original array** since it operates on the same memory location.

Using Pointer Notation



```c
// Function to double each array element
void doubleArray(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        *(arr + i) *= 2;  // Same as arr[i] *= 2
    }
}
```

- The array name acts as a pointer, we can use **pointer notation** instead of arr[i]

1D Array Summary

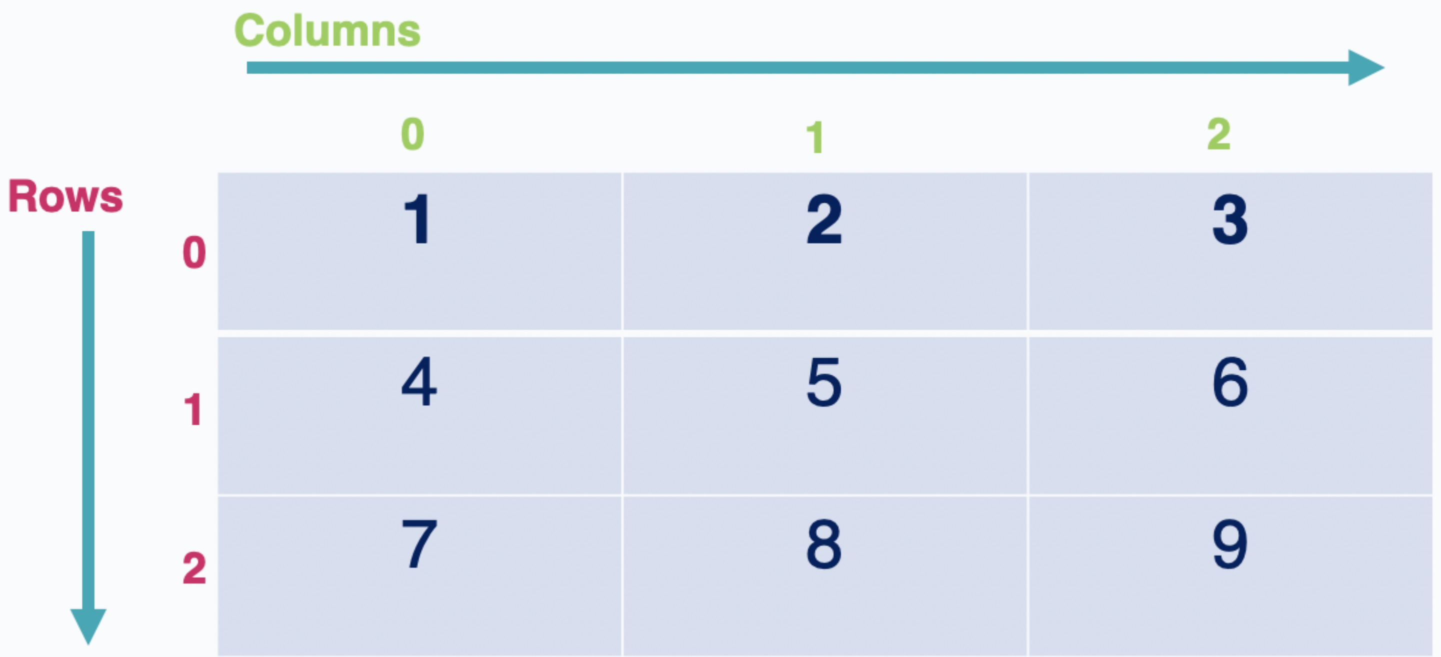
Feature	Passing a 1-D Array
How to pass it?	Use the array name (e.g., function(arr, size);)
Is it copied?	No, a pointer to the array is passed
Can it be modified?	Yes, changes inside the function affect the original
Does size need to be passed	Yes, because the function does not know the array size

- Passing a 1-D array as a parameter to a function in C is always **Pass by Reference**. This is because in C, the **name of an array is equivalent to the address of the 1st element of that array** and when you pass the array name in the function call, you are actually **passing the address location** of the 1st element.
- Therefore, if you make any changes to your parameter array in your function, you will be changing the original array back where the function was called.

Passing a 2D Array to a Function

- **2-D arrays** can be passed to a function, but there are some differences compared to **1-D arrays**.
- The function receives a **pointer to the first element** of the array.
- The **number of columns** must be specified in the function **declaration** (or provided via macro values).
- The function can access and modify the original array.

2D Arrays - revision

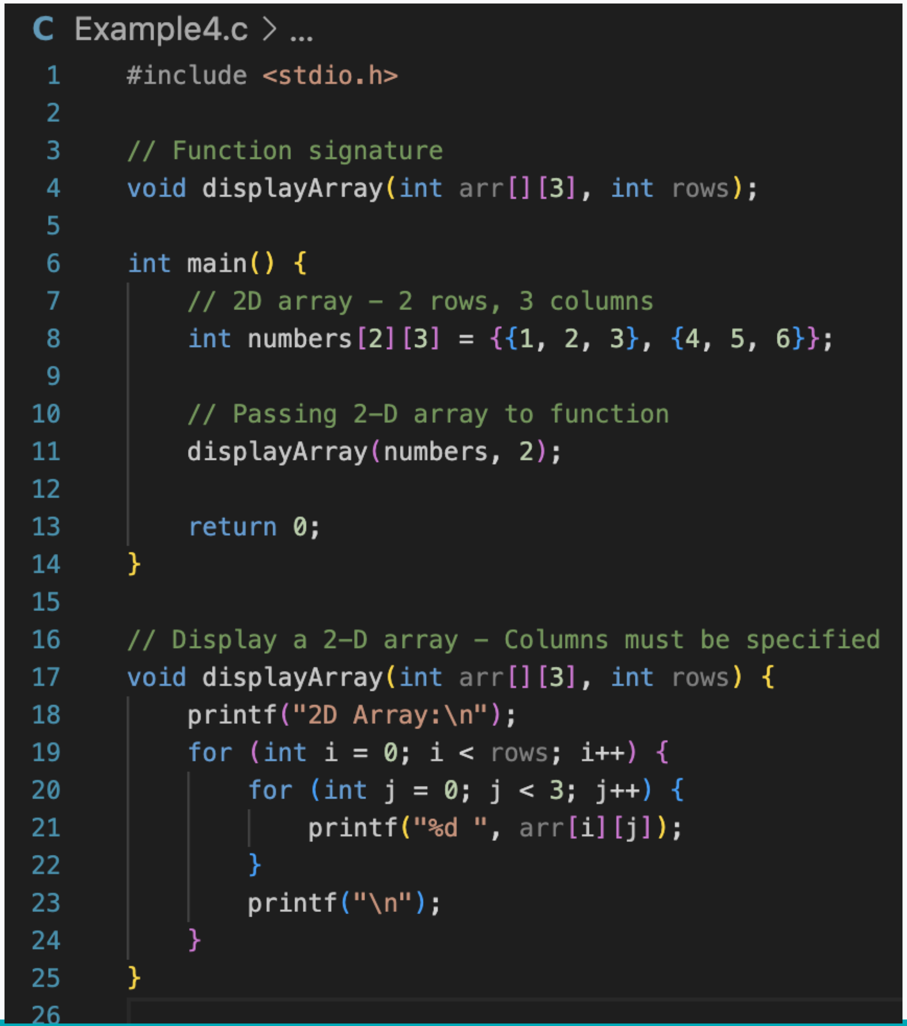


$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

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

matrix[Rows][Columns]

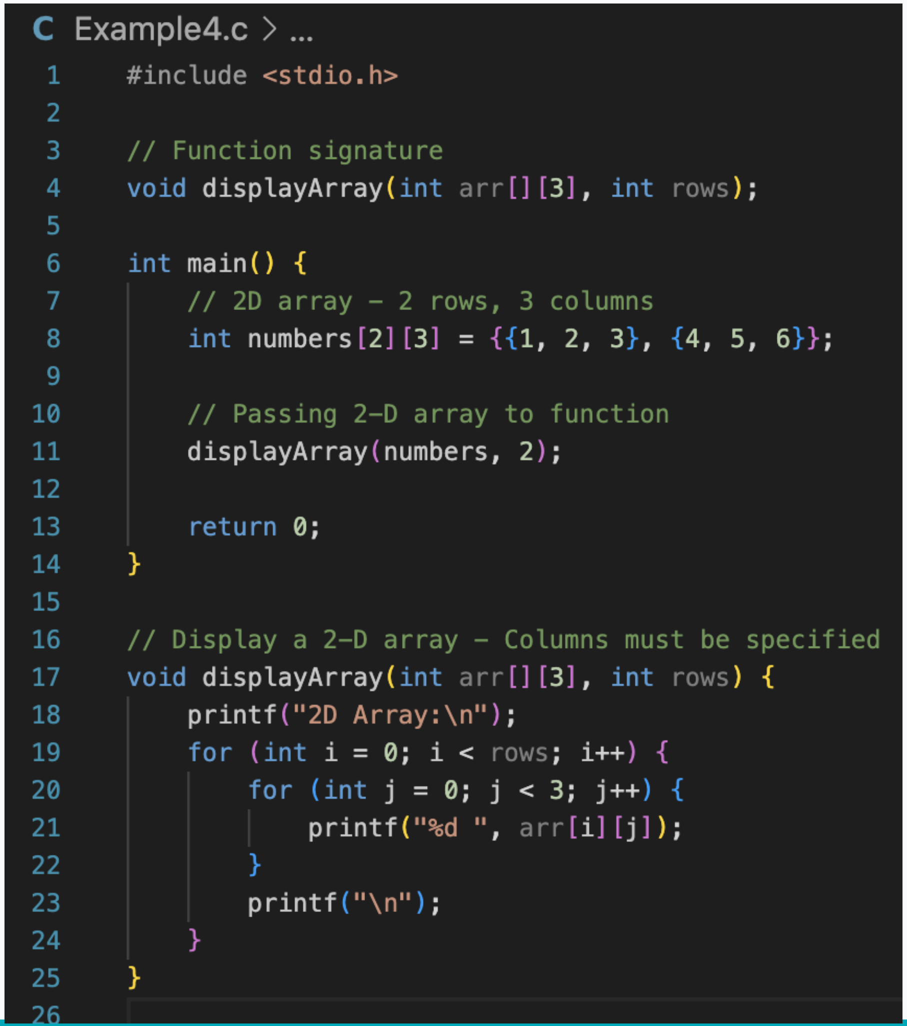
2D - Example 1


Example4.c > ...

```c
#include <stdio.h>

// Function signature
void displayArray(int arr[][3], int rows);

int main() {
    // 2D array - 2 rows, 3 columns
    int numbers[2][3] = {{1, 2, 3}, {4, 5, 6}};

    // Passing 2-D array to function
    displayArray(numbers, 2);

    return 0;
}

// Display a 2-D array - Columns must be specified
void displayArray(int arr[][3], int rows) {
    printf("2D Array:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}

```

- The function displayArray **receives** the array but must have the **number of columns** in its parameter (arr[][3]).
- The function **loops through rows and columns** to access elements.
- There is a fundamental problem with the function displayArray, what do you think it is?

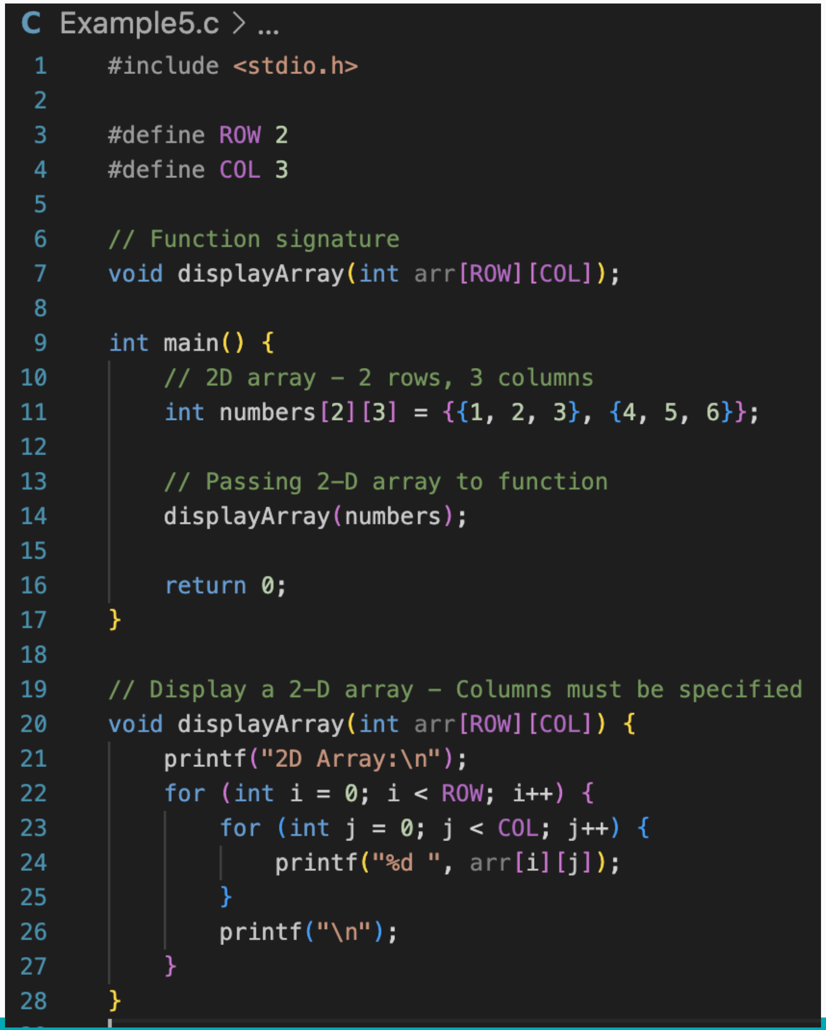
2D - Example 1


Example4.c > ...

```c
#include <stdio.h>

// Function signature
void displayArray(int arr[][3], int rows);

int main() {
    // 2D array - 2 rows, 3 columns
    int numbers[2][3] = {{1, 2, 3}, {4, 5, 6}};

    // Passing 2-D array to function
    displayArray(numbers, 2);

    return 0;
}

// Display a 2-D array - Columns must be specified
void displayArray(int arr[][3], int rows) {
    printf("2D Array:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

- The function displayArray **receives** the array but must have the **number of columns** in its parameter (arr[][3]).
- The function **loops through rows and columns** to access elements.
- There is a fundamental problem with the function displayArray, what do you think it is?
- **The function is catering for 3 columns only, what if this is a different number?**

2D Array - refactored


C Example5.c > ...

```c
#include <stdio.h>

#define ROW 2
#define COL 3

// Function signature
void displayArray(int arr[ROW][COL]);

int main() {
    // 2D array - 2 rows, 3 columns
    int numbers[2][3] = {{1, 2, 3}, {4, 5, 6}};

    // Passing 2-D array to function
    displayArray(numbers);

    return 0;
}

// Display a 2-D array - Columns must be specified
void displayArray(int arr[ROW][COL]) {
    printf("2D Array:\n");
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

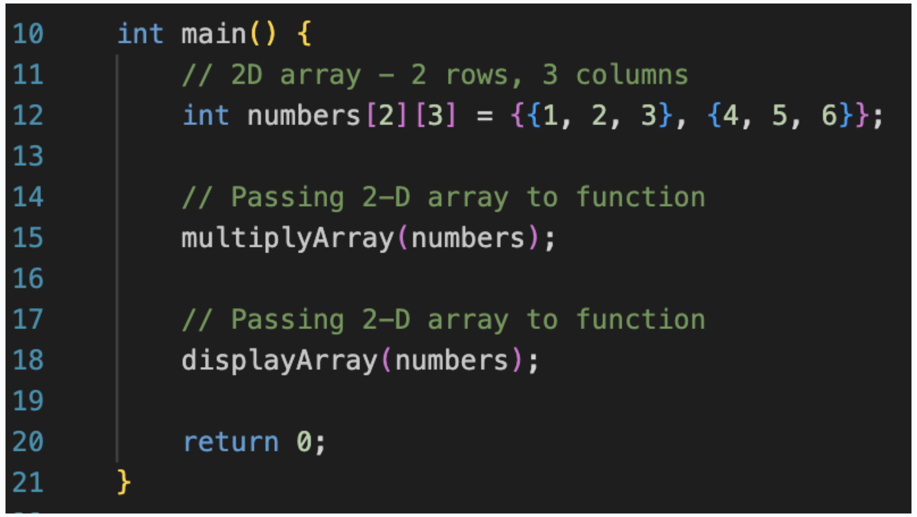
- **ROW** and **COL** have been defined as macros.
- The function needs to know how many rows and cols are contained within our numbers 2D array. This is now provided via ROW and COL.
- We have removed the hard coded values in the function.

Modify the 2D Array

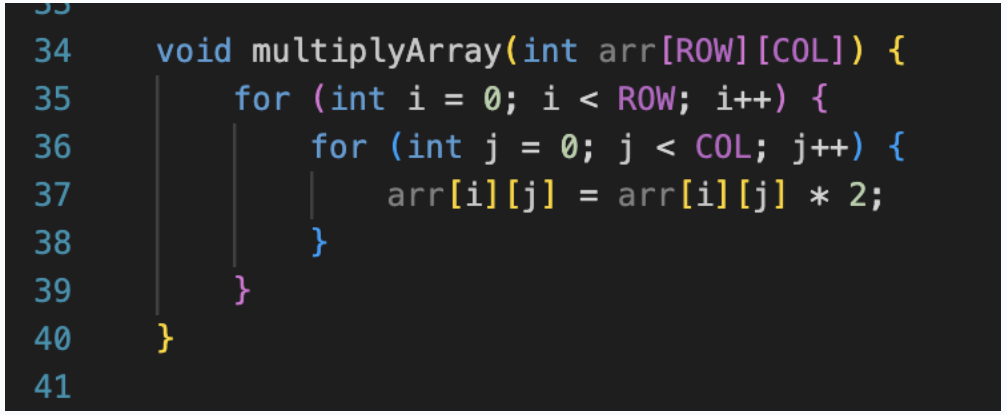
- Based on the previous example, can we create a function to double the value of every item in the 2D array.

Modify the 2D Array

- Based on the previous example, can we create a function to double the value of every item in the 2D array.



```
int main() {
    // 2D array - 2 rows, 3 columns
    int numbers[2][3] = {{1, 2, 3}, {4, 5, 6}};

    // Passing 2-D array to function
    multiplyArray(numbers);

    // Passing 2-D array to function
    displayArray(numbers);

    return 0;
}
```



```
void multiplyArray(int arr[ROW][COL]) {
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            arr[i][j] = arr[i][j] * 2;
        }
    }
}
```

First create the function to double the value of the items in the array. Then we need to call the new function before we call the display function.

2D Array Summary

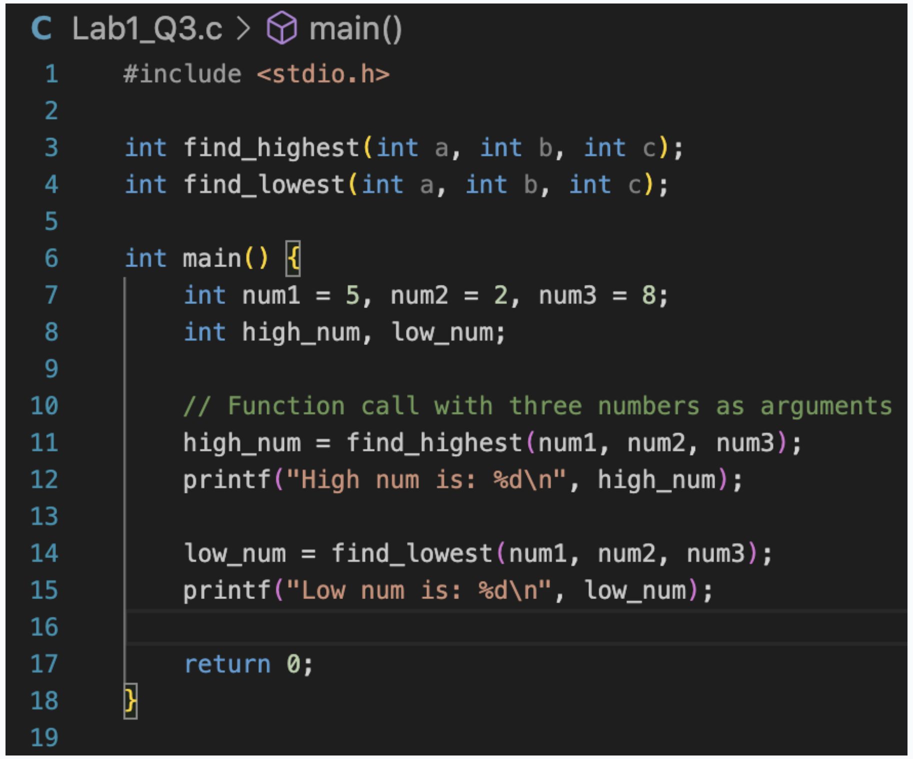
Feature	Passing a 2-D Array
How to pass it?	Use the array name (e.g., function(arr, rows);)
Is it copied?	No, a pointer to the array is passed
Columns must be specified?	Yes, required in the function declaration
Can we use pointers?	Yes, but requires pointer arithmetic

- Passing a 2-D array to a function is very similar to passing a 1-D array. However, there is a small change needed in the code.
- When you pass a 2-D array, the function signature requires you to explicitly state the number of Columns.

Lab 1 – Q3

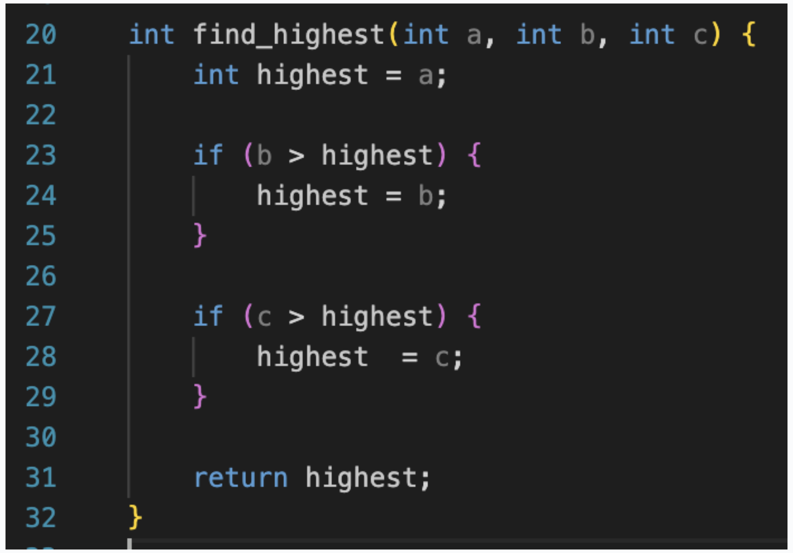
- Write a program that uses a function to find the highest and lowest number of 3 values. These 3 values must be passed as parameters to the function, i.e., function_name(int, int, int). Your function should find these values and display messages stating:
  - The Highest value is x
  - The Lowest value is y

Lab 1 – Q3

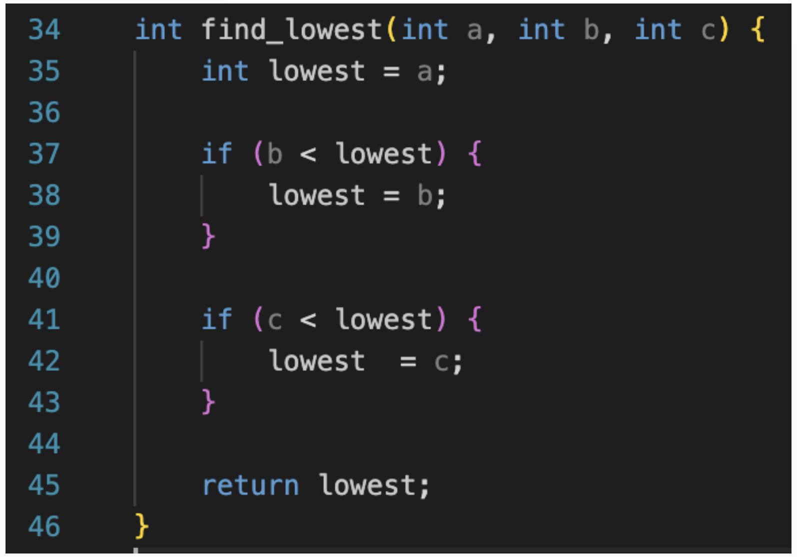


Lab1_Q3.c > main()

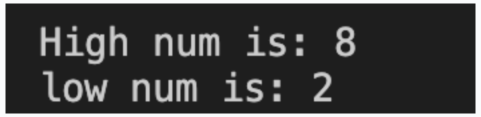
```c
#include <stdio.h>

int find_highest(int a, int b, int c);
int find_lowest(int a, int b, int c);

int main() {
    int num1 = 5, num2 = 2, num3 = 8;
    int high_num, low_num;

    // Function call with three numbers as arguments
    high_num = find_highest(num1, num2, num3);
    printf("High num is: %d\n", high_num);

    low_num = find_lowest(num1, num2, num3);
    printf("Low num is: %d\n", low_num);

    return 0;
}

```



```c
int find_highest(int a, int b, int c) {
    int highest = a;

    if (b > highest) {
        highest = b;
    }

    if (c > highest) {
        highest  = c;
    }

    return highest;
}
```



```c
int find_lowest(int a, int b, int c) {
    int lowest = a;

    if (b < lowest) {
        lowest = b;
    }

    if (c < lowest) {
        lowest  = c;
    }

    return lowest;
}
```



```
High num is: 8
low num is: 2
```

Can we refactor this solution to use Arrays?

Mandatory Question – in class solution

- Pass by Reference. Write a program similar to Q4 above but this time use Pass by
- Reference to modify the integer variable declared in main().
- Q4: Pass by Value. Write a program to demonstrate the use of Pass by Value with a function. Begin by creating an integer variable in your main() and initialise it to 1. Print this value here. Next, call your function and pass this variable as a parameter to the function. Increment the parameter variable in your function by 2 and print this value. Your function should end here, and execution returns back to where the function was called. Finally, display the value of the variable in your main() again and see if it has changed value. Did the function increment the variable in your main()?

Questions