



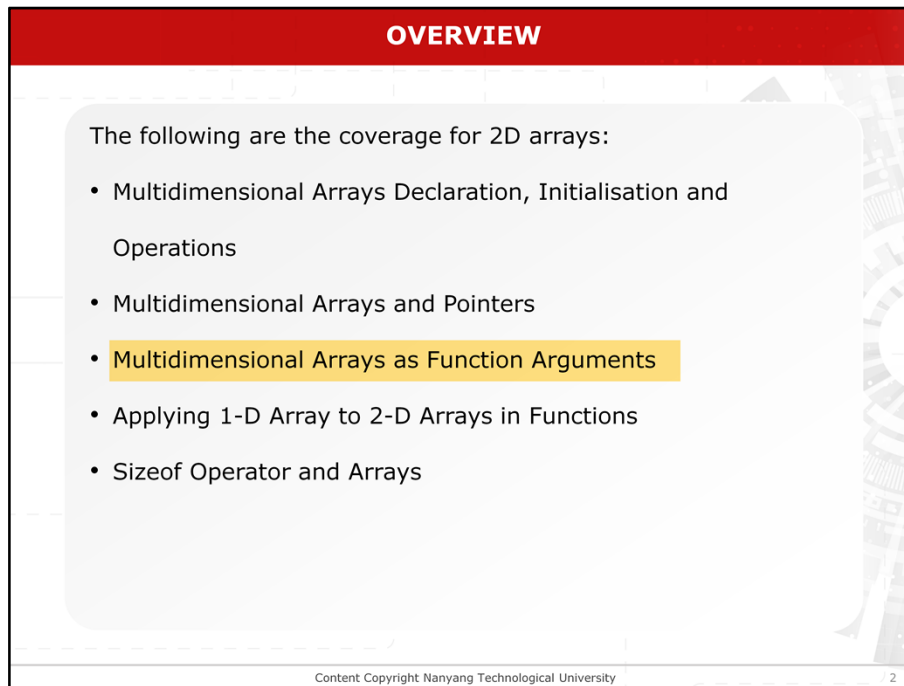
**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**

## **CE1007/ CE1007 DATA STRUCTURES**

### **Lesson 7.3 Multidimensional Arrays as Function Arguments**

Assoc Prof Hui Siu Cheung

**College of Engineering**  
School of Computer Science and Engineering



**OVERVIEW**

The following are the coverage for 2D arrays:

- Multidimensional Arrays Declaration, Initialisation and Operations
- Multidimensional Arrays and Pointers
- **Multidimensional Arrays as Function Arguments**
- Applying 1-D Array to 2-D Arrays in Functions
- Sizeof Operator and Arrays

Content Copyright Nanyang Technological University 2

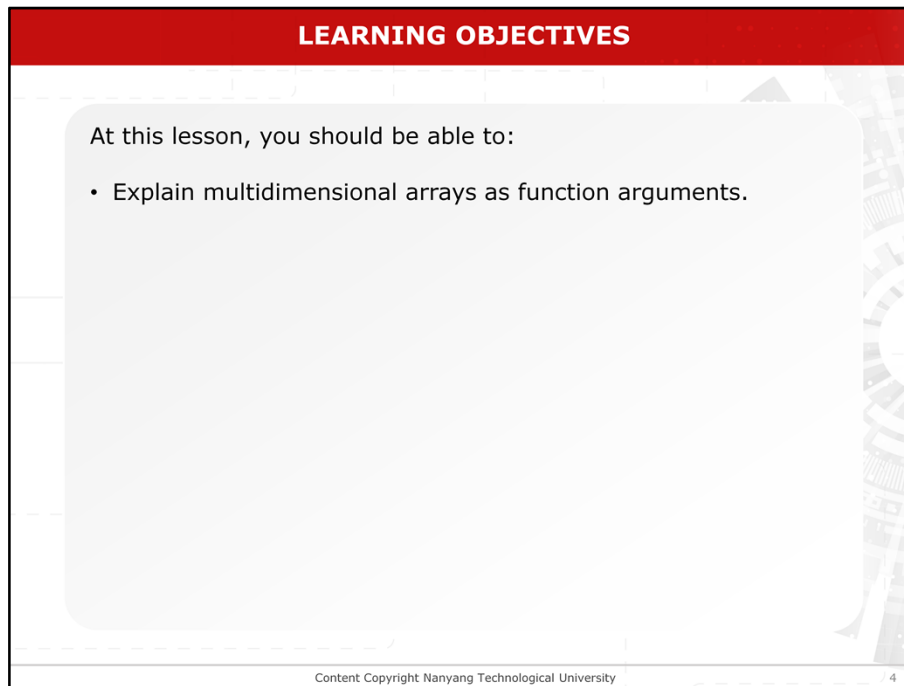
The following are the coverage for 2d ARRAYS. this video focusses on Multidimensional arrays as function arguments

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

Content Copyright Nanyang Technological University 3

Learning Objectives: At this lesson, you should be able to:



**LEARNING OBJECTIVES**

At this lesson, you should be able to:

- Explain multidimensional arrays as function arguments.

Content Copyright Nanyang Technological University 4

- Explain multi-dimensional arrays as function arguments.

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

- Explain multidimensional arrays as function arguments.
- Explain why the first dimension can be excluded from the function definition.

Content Copyright Nanyang Technological University 5

Explain why the first dimension can be excluded from the function definition.

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

- Explain multidimensional arrays as function arguments.
- Explain why the first dimension can be excluded from the function definition.
- Explain the concept of passing 2D arrays as function arguments.

Content Copyright Nanyang Technological University 6

Explain the concept of passing 2D arrays as function arguments.

### MULTIDIMENSIONAL ARRAYS AS FUNCTION ARGUMENTS

The definition of a function with a 2-D array as the argument is:

<pre><b>void fn(int array[2][4])</b> {     .... }</pre>	or	<pre><b>void fn(int array[ ][4])</b> {     .... }</pre>
---	----	---

Content Copyright Nanyang Technological University 7

The definition of a function with a 2-D array as the argument is as shown.

### MULTIDIMENSIONAL ARRAYS AS FUNCTION ARGUMENTS

The definition of a function with a 2-D array as the argument is:

```
void fn(int array[2][4])  
{  
    ....  
}
```

or

```
void fn(int array[ ][4])  
{  
    ....  
}
```

*/\*note that the first dimension can be excluded\*/*

In the above definition, the first dimension can be excluded because the C compiler needs the information of all but the first dimension.

Content Copyright Nanyang Technological University 8

In the above definition, the first dimension can be excluded because the C compiler needs the information of all but the first dimension.



**WHY THE FIRST DIMENSION CAN BE OMITTED?**

- The first dimension (i.e. the row information) of the array can be excluded in the function definition because C compiler can determine the first dimension automatically. However, the number of columns must be specified.

Content Copyright Nanyang Technological University 9

**Why the First Dimension can be Omitted?**

The first dimension that is the row information of the array can be excluded in the function definition because C compiler can determine the first dimension automatically. However, the number of columns must be specified.

### WHY THE FIRST DIMENSION CAN BE OMITTED?

- The first dimension (i.e. the row information) of the array can be excluded in the function definition because C compiler can determine the first dimension automatically. However, the number of columns must be specified.
- For example, the assignment operation  

`array[1][3] = 100;`

  
requests the **compiler** to compute the address of **array[1][3]** and then place 100 to that address.

Content Copyright Nanyang Technological University 10

For example, the assignment statement shown requests the compiler to compute the address of and then places a value of 100 to that address.

### WHY THE FIRST DIMENSION CAN BE OMITTED?

- The first dimension (i.e. the row information) of the array can be excluded in the function definition because C compiler can determine the first dimension automatically. However, the number of columns must be specified.
- For example, the assignment operation  

`array[1][3] = 100;`

  
requests the **compiler** to compute the address of **array[1][3]** and then place 100 to that address.
- In order to compute the address, the dimension information must be given to the compiler.

Content Copyright Nanyang Technological University 11

In order to compute the address, the dimension information must be given to the compiler

### WHY THE FIRST DIMENSION CAN BE OMITTED?

Let's redefine **array** (i.e. `int array[2][4]`) as

`int array[D1][D2]; // D1=2, D2=4`

The address of `array[1][3]` is computed as

$$\text{baseAddress} + \text{row} * \text{D2} + \text{column}$$

$$\Rightarrow \text{baseAddress} + 1 * 4 + 3$$

$$\Rightarrow \text{baseAddress} + 7$$

Content Copyright Nanyang Technological University

Let us redefine **array** as shown here.

### WHY THE FIRST DIMENSION CAN BE OMITTED?

Let's redefine **array** (i.e. `int array[2][4]`) as

```
int array[D1][D2]; // D1=2, D2=4
```

The address of `array[1][3]` is computed as

$$\text{baseAddress} + \text{row} * \text{D2} + \text{column}$$

Memory

array[0][0]	baseAddress + row * D2
array[0][1]	
array[0][2]	
array[0][3]	
array[1][0]	+ column
array[1][1]	
array[1][2]	
array[1][3]	

==>  $\text{baseAddress} + 1 * 4 + 3$

==>  $\text{baseAddress} + 7$

Content Copyright Nanyang Technological University 13

The address of **array with 1 row and 3 column** is computed as shown where the base Address is the address pointing to the beginning of **array**.

### WHY THE FIRST DIMENSION CAN BE OMITTED?

Let's redefine **array** (i.e. **int array[2][4]**) as

```
int array[D1][D2]; // D1=2, D2=4
```

The address of array[1][3] is computed as

$$\text{baseAddress} + \text{row} * D2 + \text{column}$$

==>  $\text{baseAddress} + 1 * 4 + 3$

==>  $\text{baseAddress} + 7$

Memory

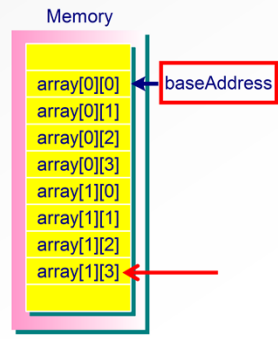
array[0][0]	← baseAddress
array[0][1]	+ row * D2
array[0][2]	
array[0][3]	
array[1][0]	←
array[1][1]	+ column
array[1][2]	
array[1][3]	←

Content Copyright Nanyang Technological University 14

The address of array one three is computed as shown here.

### WHY THE FIRST DIMENSION CAN BE OMITTED? (CONTINUED)

- The **baseAddress** is the address pointing to the beginning of array.



Memory

array[0][0]  
array[0][1]  
array[0][2]  
array[0][3]  
array[1][0]  
array[1][1]  
array[1][2]  
array[1][3]

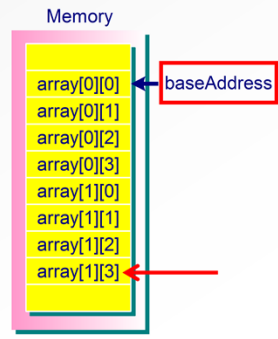
baseAddress

Content Copyright Nanyang Technological University 15

The **base Address** is the address pointing to the beginning of array.

### WHY THE FIRST DIMENSION CAN BE OMITTED?

- The **baseAddress** is the address pointing to the beginning of array.
- Because **D1** is **not needed in computing the address**, one can omit the first dimension value in defining a function which takes arrays as its formal arguments.



Memory

array[0][0]  
array[0][1]  
array[0][2]  
array[0][3]  
array[1][0]  
array[1][1]  
array[1][2]  
array[1][3]

baseAddress

Content Copyright Nanyang Technological University 16

Because **D1** is **not needed in computing the address**, one can omit the first dimension value in defining a function which takes arrays as its formal arguments.



### WHY THE FIRST DIMENSION CAN BE OMITTED?

- The **baseAddress** is the address pointing to the beginning of array.
- Because **D1** is **not needed in computing the address**, one can omit the first dimension value in defining a function which takes arrays as its formal arguments.
- Therefore, the prototype of the function could be:
 

```
void fn(int array[2][4]);
or
void fn(int array[ ][4]);
```

Memory

Content Copyright Nanyang Technological University

17

#### Why the First Dimension can be Omitted?

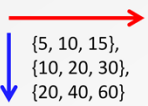
Therefore, the function prototype of the function can be as shown here. Similar to one-dimensional array, the name of the array **table** is specified as the argument without any subscripts in the function call. The above discussion and definition of two-dimensional arrays can be generalized to the arrays of higher dimensions.

**PASSING 2-D ARRAY AS FUNCTION ARGUMENTS: EXAMPLE**

```

#include <stdio.h>
int sum_all_rows(int array[ ][3]);
int sum_all_columns(int array[ ][3]);
int main()
{
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int total_row, total_column;
    total_row = sum_all_rows(ar); // sum of all rows
    total_column = sum_all_columns(ar); // sum of all columns
    printf("The sum of all elements in rows is %d\n", total_row);
    printf("The sum of all elements in columns is %d\n", total_column);
    return 0;
}

```



Content Copyright Nanyang Technological University
18

### Passing 2-D Array as Function Arguments: Example

The program determines the total sum of all the rows and the total sum of all the columns of a two-dimensional array. Two functions **sum all rows** and **sum all columns** are written to compute the total sums. Both functions take an array as its argument:

### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS: EXAMPLE

```
#include <stdio.h>
int sum_all_rows(int array[ ][3]);
int sum_all_columns(int array[ ][3]);
int main()
{
    int ar[3][3] = {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int total_row, total_column;
    total_row = sum_all_rows(ar); // sum of all rows
    total_column = sum_all_columns(ar); // sum of all columns
    printf("The sum of all elements in rows is %d\n", total_row);
    printf("The sum of all elements in columns is %d\n", total_column);
    return 0;
}
```

Diagram illustrating the 2D array `ar` with dimensions `3x3`. The array is initialized with the following values:

5	10	15
10	20	30
20	40	60

The diagram shows a red arrow pointing from the `3` in `ar[3][3]` to the first dimension of the array, and a blue arrow pointing from the `3` in `ar[3][3]` to the second dimension of the array.

Content Copyright Nanyang Technological University 19

Note that the first dimension of the array parameter in the function prototype can be omitted.

### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS: EXAMPLE

```

#include <stdio.h>
int sum_all_rows(int array[ ][3]);
int sum_all_columns(int array[ ][3]);
int main()
{
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int total_row, total_column;
    total_row = sum_all_rows(ar); // sum of all rows
    total_column = sum_all_columns(ar); // sum of all columns
    printf("The sum of all elements in rows is %d\n", total_row);
    printf("The sum of all elements in columns is %d\n", total_column);
    return 0;
}

```

Content Copyright Nanyang Technological University 20

When calling the functions, the name of the array is passed to the calling functions:

The total values are computed in the two functions and placed in the two variables **total row** and **total column** respectively.

### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS: EXAMPLE

```
#include <stdio.h>
int sum_all_rows(int array[ ][3]);
int sum_all_columns(int array[ ][3]);
int main()
{
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int total_row, total_column;
    total_row = sum_all_rows(ar); // sum of all rows
    total_column = sum_all_columns(ar); // sum of all columns
    printf("The sum of all elements in rows is %d\n", total_row);
    printf("The sum of all elements in columns is %d\n", total_column);
    return 0;
}
```

**Output**  
The sum of all elements in rows is 210  
The sum of all elements in columns is 210

Content Copyright Nanyang Technological University 21

The output of the program is shown here.

**PASSING 2-D ARRAY AS FUNCTION ARGUMENTS**

```

int sum_all_rows(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (row = 0; row < 3; row++)
    {
        for (column = 0; column < 3; column++)
            sum += array[row][column];
    }
    return sum;
}
int sum_all_columns(int array[ ][3])
{
    }

```

←
**1<sup>st</sup> Dimension can be omitted**

Content Copyright Nanyang Technological University 22

### Passing 2-D Array as Function Arguments: Example

Note that the first dimension of the array parameter **array** in the function **sum all rows** can be omitted.

### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS

```
int sum_all_rows(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (row = 0; row < 3; row++)
    {
        for (column = 0; column < 3; column++)
            sum += array[row][column];
    }
    return sum;
}

int sum_all_columns(int array[ ][3])
{
    // ...
}
```

1<sup>st</sup> Dimension can be omitted

for (row = 0; row < 3; row++)  
{  
 for (column = 0; column < 3; column++)  
 sum += array[row][column];  
}

Content Copyright Nanyang Technological University

23

#### Passing 2-D Array as Function Arguments: Example

A nested **for** loop is used to traverse the 2-dimensional array in order to compute the sum of all rows.

### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS

```

int sum_all_rows(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (row = 0; row < 3; row++)
    {
        for (column = 0; column < 3; column++)
            sum += array[row][column];
    }
    return sum;
}

int sum_all_columns(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (column = 0; column < 3; column++)
    {
        for (row = 0; row < 3; row++)
            sum += array[row][column];
    }
    return sum;
}

```

1<sup>st</sup> Dimension can be omitted

Content Copyright Nanyang Technological University 24

#### Passing 2-D Array as Function Arguments: Example

Note that the first dimension of the array parameter **array** in the function **sum all columns** can be omitted.



### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS

```

int sum_all_rows(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (row = 0; row < 3; row++)
    {
        for (column = 0; column < 3; column++)
            sum += array[row][column];
    }
    return sum;
}

int sum_all_columns(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (column = 0; column < 3; column++)
    {
        for (row = 0; row < 3; row++)
            sum += array[row][column];
    }
    return sum;
}

```

1<sup>st</sup> Dimension can be omitted

Content Copyright Nanyang Technological University

25

#### Passing 2-D Array as Function Arguments: Example

A nested **for** loop is used to traverse the 2-dimensional array in order to compute the sum of all columns.

### PASSING 2-D ARRAY AS FUNCTION ARGUMENTS

```
int sum_all_rows(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (row = 0; row < 3; row++)
    {
        for (column = 0; column < 3; column++)
            sum += array[row][column];
    }
    return sum;
}

int sum_all_columns(int array[ ][3])
{
    int row, column;
    int sum=0;
    for (column = 0; column < 3; column++)
    {
        for (row = 0; row < 3; row++)
            sum += array[row][column];
    }
    return sum;
}
```

1<sup>st</sup> Dimension can be omitted

Content Copyright Nanyang Technological University 26

The result **sum** is then returned to the calling **main** function.

**SUMMARY**

At this lesson, you should be able to:

- Explain multi-dimensional arrays as function arguments.
- Explain why the first dimension can be excluded from the function definition.
- Explain the concept of passing 2D arrays as function arguments.

Content Copyright Nanyang Technological University 27

In summary, after watching this video lesson, you should be able to do the listed.