

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

CE1007/ CZ1007 DATA STRUCTURES

Lesson 7.2 Multidimensional Arrays and Pointers

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 and pointers

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:

- Describe the association between the array elements and the associated pointers.

Describe the association between the array elements and the associated pointers.

MULTIDIMENSIONAL ARRAYS AND POINTERS

Multidimensional arrays - stored sequentially in memory.

```
int ar[4][2]; /* ar is an array of 4 elements; each element is an  
array of 2 ints */
```

For example:

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

Content Copyright Nanyang Technological University

5

Multi-dimensional Arrays and Pointers

Multi-dimensional arrays are also stored sequentially in the memory. For example, consider the following two-dimensional array, where **array** is also the address of the first element of the array.

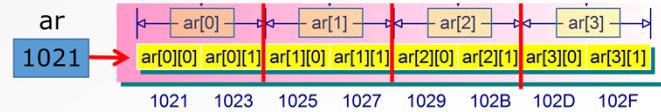
MULTIDIMENSIONAL ARRAYS AND POINTERS

Multidimensional arrays - stored sequentially in memory.

```
int ar[4][2];      /* ar is an array of 4 elements;
each element is an array of 2 ints */
```

For example:

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



Content Copyright Nanyang Technological University

6

Assume that each integer takes two bytes in the memory, the memory layout of the two-dimensional array is shown with its associated pointers. We will discuss the association between the array elements and the associated pointers in the next few slides.

MULTIDIMENSIONAL ARRAYS AND POINTERS

`int ar[4][2];`

• **ar** - the address of the **1st element** of the array.
In this case, the 1st element is an **array of 2 ints**.

• **ar** - the address of a **two-int-sized object**.

ar == &ar[0]
 $ar + 1 == \&ar[1]$
 $ar + 2 == \&ar[2]$
 $ar + 3 == \&ar[3]$

Content Copyright Nanyang Technological University 7

Multi-dimensional Arrays and Pointers

A R is the address of the first element is an array of 2 integers. It is the address of a two-integer sized object.

MULTIDIMENSIONAL ARRAYS AND POINTERS

```
int ar[4][2];
```

- **ar** - the address of the **1st element** of the array.
In this case, the 1st element is an **array of 2 ints**.
- **ar** - the address of a **two-int-sized object**.

ar == &ar[0]	Note: Adding 1 to a pointer or address yields a value larger by the size of the referred-to object.
ar + 1 == &ar[1]	
ar + 2 == &ar[2]	
ar + 3 == &ar[3]	

Content Copyright Nanyang Technological University 8

Multi-dimensional Arrays and Pointers

Note that adding 1 to a pointer or address yields a value larger by the size of the referred-to object.

MULTIDIMENSIONAL ARRAYS AND POINTERS

`int ar[4][2];`

- **ar** - the address of the **1st element** of the array.
In this case, the 1st element is an **array of 2 ints**.
- **ar** - the address of a **two-int-sized object**.

<code>ar == &ar[0]</code>	Note: Adding 1 to a pointer or address yields a value larger by the size of the referred-to object. e.g. ar has the same address value as ar[0] ar+1 has the same address value as ar[1] , etc.
-------------------------------	--

Content Copyright Nanyang Technological University 9

An example is shown here

MULTIDIMENSIONAL ARRAYS AND POINTERS

```
int ar[4][2];
ar
```

- **ar[0]** is an array of 2 integers, so **ar[0]** is the **address of int-sized object**.

**ar[0] == &ar[0][0]
ar[1] == &ar[1][0]
ar[2] == &ar[2][0]
ar[3] == &ar[3][0]**

Note:
ar[0] has the same address as **ar[0][0]**;
ar[0]+1 refers to the address of **ar[0][1]** (1023)

Content Copyright Nanyang Technological University 10

Multi-dimensional Arrays and Pointers

Element of an array is an array of 2 integers, so **element of an array** is the address of an integer sized object. Therefore, we have as shown.

This is due to the fact that **array** is a two-integer sized object, while **element of an array** is an integer sized object. Adding 1 to **array** increases by 4 bytes.

MULTIDIMENSIONAL ARRAYS AND POINTERS

`int ar[4][2];`

- **Dereferencing** a pointer or an address (apply *** operator**) yields the value represented by the referred-to object.

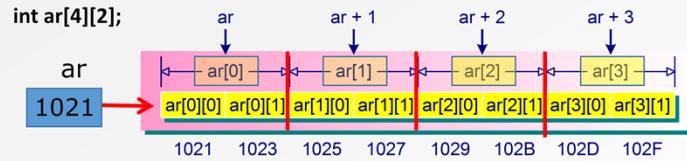
<code>ar == &ar[0]</code>	<code>*ar == ar[0]</code>	(using dereferencing)
<code>ar + 1 == &ar[1]</code>	<code>*(ar + 1) == ar[1]</code>	
<code>ar + 2 == &ar[2]</code>	<code>*(ar + 2) == ar[2]</code>	
<code>ar + 3 == &ar[3]</code>	<code>*(ar + 3) == ar[3]</code>	

Content Copyright Nanyang Technological University 11

Multi-dimensional Arrays and Pointers

Dereferencing a pointer or an address yields the value represented by the referred-to object.

MULTIDIMENSIONAL ARRAYS AND POINTERS



- **Dereferencing** a pointer or an address (apply *** operator**) yields the value represented by the referred-to object.

Similarly

<code>ar[0] == &ar[0][0]</code>	<code>*ar[0] == ar[0][0]</code>	(using dereferencing)
<code>ar[1] == &ar[1][0]</code>	<code>*ar[1] == ar[1][0]</code>	
<code>ar[2] == &ar[2][0]</code>	<code>*ar[2] == ar[2][0]</code>	
<code>ar[3] == &ar[3][0]</code>	<code>*ar[3] == ar[3][0]</code>	

Content Copyright Nanyang Technological University

12

Multi-dimensional Arrays and Pointers

Similarly, we have the following

MULTIDIMENSIONAL ARRAYS AND POINTERS

- Therefore:

`*ar[0]` == the value stored in `ar[0][0]`.

`*ar` == the value of its first element, `ar[0]`.

We have

`**ar` == the value of `ar[0][0]` (*double indirection*)

Content Copyright Nanyang Technological University

13

Multi-dimensional Arrays and Pointers

We can use the `*` operator to dereference a pointer or an address to yield the value represented by the referred-to object:

This is called *double indirection*.

MULTIDIMENSIONAL ARRAYS AND POINTERS

```

int ar[4][2];
    
```

- After some calculations using **double** dereferencing as shown above, we will get the general formula for using pointer to access each element of a 2-D array **ar** with row=m, column=n, as follows:

Content Copyright Nanyang Technological University

14

Multi-dimensional Arrays and Pointers

After some calculations using **double** dereferencing as shown above, we will get the general formula for using pointer to access each element of a 2-D array with row **m** and column **n**

In general, we can represent individual elements of a two-dimensional array as

$$\text{ar}[m][n] == *(\text{*(ar}+m\text{)} + n)$$

where **m** is the index associated with **ar**, and **n** is the index associated with the sub-array **ar[m]**. When **n** is added to ***(ar+m)**, i.e. ***(ar+m)+n**, it becomes the address of the element of **ar[m][n]**. Applying the ***** operator to that address gives the content at that address location, i.e. the array element content.

MULTIDIMENSIONAL ARRAYS AND POINTERS

```
int ar[4][2];
      ar      ar + 1    ar + 2    ar + 3
      |       |       |       |
      +-----+-----+-----+
ar 1021 → [ar[0][0] ar[0][1] ar[1][0] ar[1][1] ar[2][0] ar[2][1] ar[3][0] ar[3][1]]
```

1021 1023 1025 1027 1029 102B 102D 102F

- After some calculations using **double** dereferencing as shown above, we will get the general formula for using pointer to access each element of a 2-D array **ar** with row=**m**, column=**n**:

$$\text{ar}[m][n] == *(*(\text{ar} + m) + n)$$

Content Copyright Nanyang Technological University 15

Multi-dimensional Arrays and Pointers

In general, we can represent individual elements of a two-dimensional array as shown where **m** is the index associated with **array**, and **n** is the index associated with the sub-array **array of element m**. When **n** is added, it becomes the address of the element of **array of element m and n**.

Applying the ***** operator to that address gives the content at that address location, i.e. the array element content.

MULTIDIMENSIONAL ARRAYS AND POINTERS

```
int ar[4][2];
```

• For example:

$$\text{ar}[2][1] = *(*(\text{ar} + 2) + 1) \quad [\text{m}=2, \text{n}=1]$$

$$\text{ar}[3][0] = *(*(\text{ar} + 3) + 0) \quad [\text{m}=3, \text{n}=0]$$

Note: You are not required to remember the calculation on deriving the general formula.

Content Copyright Nanyang Technological University 16

Multi-dimensional Arrays and Pointers

Examples on using the general formula are as shown.

MULTIDIMENSIONAL ARRAYS AND POINTERS

On the other hand, you may also use the following way to access each element of a 2-D array.

For example, in the array declaration

```
int ar[4][2];
```

Let's redefine it as **ar[D1][D2]**, then **D1=4**, **D2=2**:

ar[m][n] == *(*ar + index) with **index = m*D2+n.**

$ar[2][1] = (*ar + (2*D2 + 1)) = (*ar + (2*2+1)) = (*ar + 5)$

$ar[3][0] = (*ar + (3*D2 + 0)) = (*ar + (3*2+0)) = (*ar + 6)$

Content Copyright Nanyang Technological University

17

*Multi-dimensional Arrays and Pointers

By using the similar idea, you may also use the following way to access each element of 2-D array with index as shown.

MULTIDIMENSIONAL ARRAYS AND POINTERS

On the other hand, you may also use the following way to access each element of a 2-D array.

For example, in the array declaration

```
int ar[4][2];
```

Let's redefine it as **ar[D1][D2]**, then **D1=4**, **D2=2**:

ar[m][n] == (*ar + index) with **index = m*D2+n.**

$\text{ar}[2][1] = (*\text{ar} + (2*D2 + 1)) = (*\text{ar} + (2*2+1)) = (*\text{ar} + 5)$

$\text{ar}[3][0] = (*\text{ar} + (3*D2 + 0)) = (*\text{ar} + (3*2+0)) = (*\text{ar} + 6)$

Content Copyright Nanyang Technological University

18

***array** refers to the array base address.

MULTIDIMENSIONAL ARRAYS AND POINTERS

On the other hand, you may also use the following way to access each element of a 2-D array.

For example, in the array declaration

```
int ar[4][2];
```

Let's redefine it as **ar[D1][D2]**, then **D1=4**, **D2=2**:

ar[m][n] == *(*ar + index) with **index = m*D2+n**.

$\text{ar}[2][1] = (*\text{ar} + (2*D2 + 1)) = (*\text{ar} + (2*2+1)) = (*\text{ar} + 5)$

$\text{ar}[3][0] = (*\text{ar} + (3*D2 + 0)) = (*\text{ar} + (3*2+0)) = (*\text{ar} + 6)$

Content Copyright Nanyang Technological University

19

The **index** indicates the relative position of the memory location of the array element from the base address in the memory.

MULTIDIMENSIONAL ARRAYS AND POINTERS

On the other hand, you may also use the following way to access each element of a 2-D array.

For example, in the array declaration

```
int ar[4][2];
```

Let's redefine it as **ar[D1][D2]**, then **D1=4**, **D2=2**:

ar[m][n] == *(ar + index) with **index = m*D2+n**.

$\text{ar}[2][1] = *(\text{ar} + (2*D2 + 1)) = *(\text{ar} + (2*2+1)) = *(\text{ar} + 5)$

$\text{ar}[3][0] = *(\text{ar} + (3*D2 + 0)) = *(\text{ar} + (3*2+0)) = *(\text{ar} + 6)$

Content Copyright Nanyang Technological University

20

(*array + index) refers to the memory location of the element of the array. Using dereferencing, we will then be able to obtain the content of the array element accordingly.

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

```
#include <stdio.h>
int main() {
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int index, i, j;
    // (1) using indexes
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", ar[i][j]);
    printf("\n");
    // (2) using the general formula ar[m][n] == *( *ar + index )
    for (index = 0; index < 9; index++)
        printf("%d ", *(*ar + index));
    printf("\n");
    // (3) using the general formula ar[m][n] == *( *( ar + m ) + n )
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", *(*(ar+i)+j));
    return 0;
}
```

Content Copyright Nanyang Technological University

21

The program aims to print the value of each array element in a 2-D array.

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

```
#include <stdio.h>
int main() {
    int ar[3][3] = {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int index, i, j;
    // (1) using indexes
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", ar[i][j]);
    printf("\n");
    // (2) using the general formula ar[m][n] == *( *ar + index )
    for (index = 0; index < 9; index++)
        printf("%d ", *(*ar + index));
    printf("\n");
    // (3) using the general formula ar[m][n] == *( *( ar + m ) + n )
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", *(*(ar+i)+j));
    return 0;
}
```

Content Copyright Nanyang Technological University

22

In the program, it first initializes each array element of the array of elements with 3 rows and 3 columns

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

```
#include <stdio.h>
int main() {
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int index, i, j;
    // (1) using indexes
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", ar[i][j]);
    printf("\n");
    // (2) using the general formula ar[m][n] == *( *ar + index )
    for (index = 0; index < 9; index++)
        printf("%d ", *(*ar + index));
    printf("\n");
    // (3) using the general formula ar[m][n] == *( *( ar + m ) + n )
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", *(*(ar+i)+j));
    return 0;
}
```

Content Copyright Nanyang Technological University

23

There are three ways to access each element of the array with a **for** loop:

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

```
#include <stdio.h>
int main() {
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int index, i, j;
    // (1) using indexes
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", ar[i][j]);
    printf("\n");
    // (2) using the general formula ar[m][n] == *( *ar + index )
    for (index = 0; index < 9; index++)
        printf("%d ", *(*ar + index));
    printf("\n");
    // (3) using the general formula ar[m][n] == *( *( ar + m ) + n )
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", *(*(ar+i)+j));
    return 0;
}
```

Output

```
5 10 15 10 20 30 20 40 60
5 10 15 10 20 30 20 40 60
5 10 15 10 20 30 20 40 60
```

Content Copyright Nanyang Technological University

24

Using the index approach

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

```
#include <stdio.h>
int main() {
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int index, i, j;
    // (1) using indexes
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", ar[i][j]);
    printf("\n");
    // (2) using the general formula ar[m][n] == *( *ar + index )
    for (index = 0; index < 9; index++)
        printf("%d ", *(*ar + index));
    printf("\n");
    // (3) using the general formula ar[m][n] == *( *( ar + m ) + n )
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", *(*(ar+i)+j));
    return 0;
}
```

Output

```
5 10 15 10 20 30 20 40 60
5 10 15 10 20 30 20 40 60
5 10 15 10 20 30 20 40 60
```

Content Copyright Nanyang Technological University

25

Using the general formula:

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

```
#include <stdio.h>
int main() {
    int ar[3][3]= {
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };
    int index, i, j;
    // (1) using indexes
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", ar[i][j]);
    printf("\n");
    // (2) using the general formula ar[m][n] == *( *ar + index )
    for (index = 0; index < 9; index++)
        printf("%d ", *(*ar + index));
    printf("\n");
    // (3) using the general formula ar[m][n] == *( *( ar + m ) + n )
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            printf("%d ", *(*(ar+i)+j));
    return 0;
}
```

Output
5 10 15 10 20 30 20 40 60 5 10 15 10 20 30 20 40 60 5 10 15 10 20 30 20 40 60

Content Copyright Nanyang Technological University 26

- Using the other general formula:

SUMMARY

At this lesson, you should be able to:

- Describe the association between the array elements and the associated pointers.

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