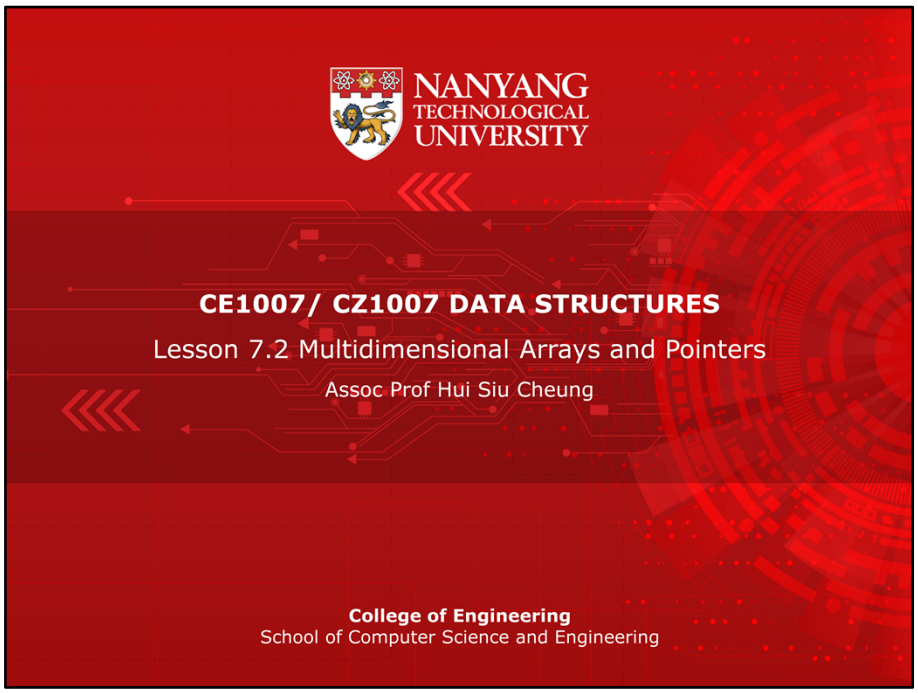
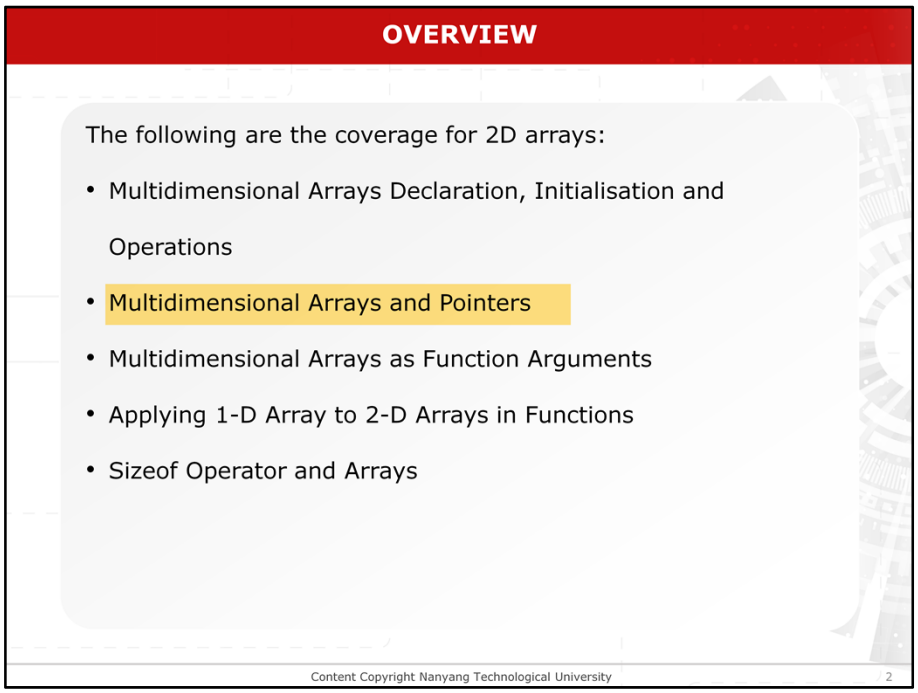
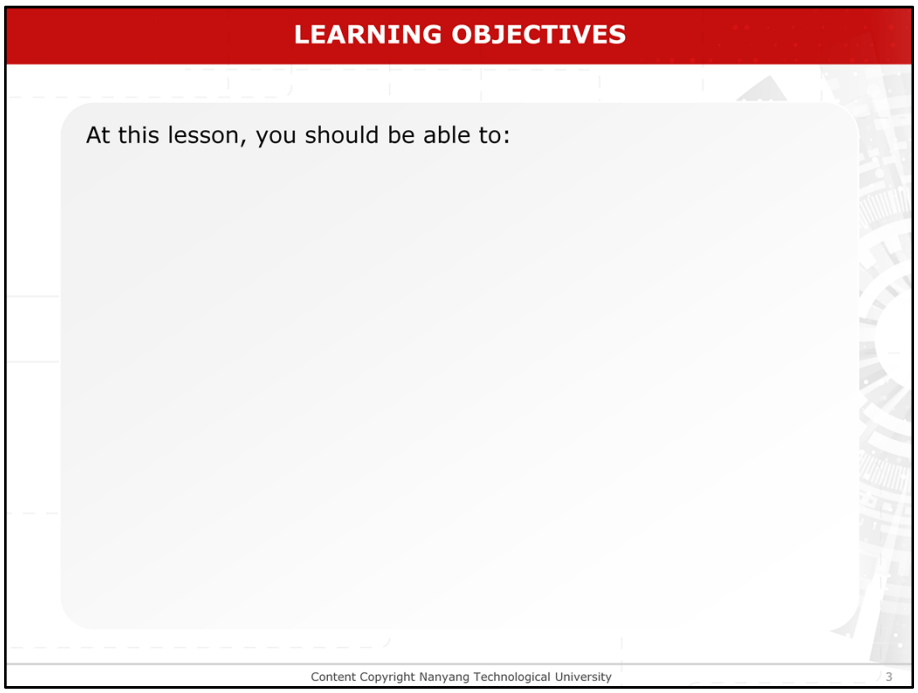
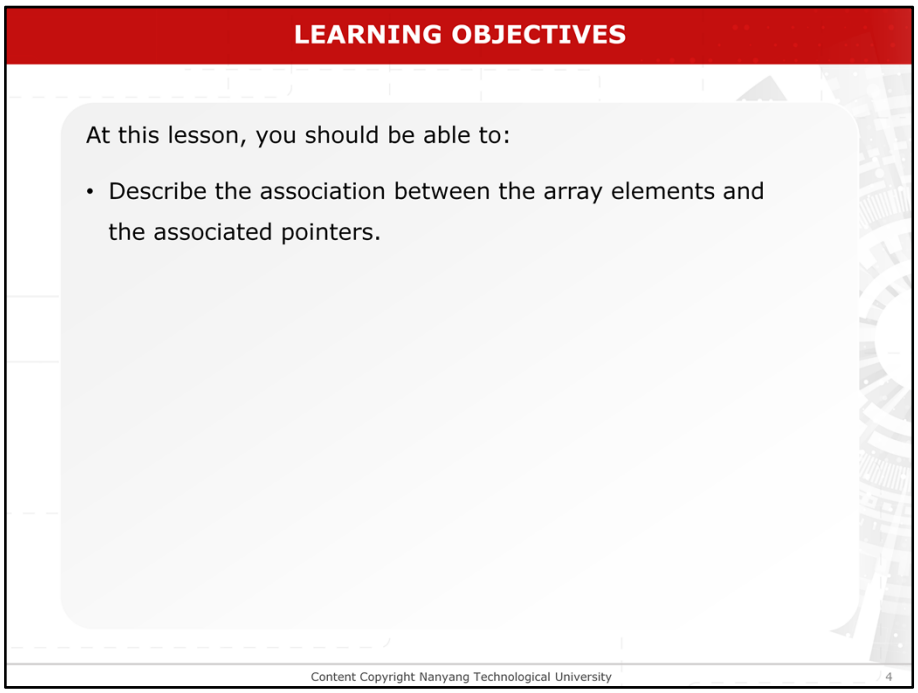

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 7.2 Multidimensional Arrays and Pointers
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering



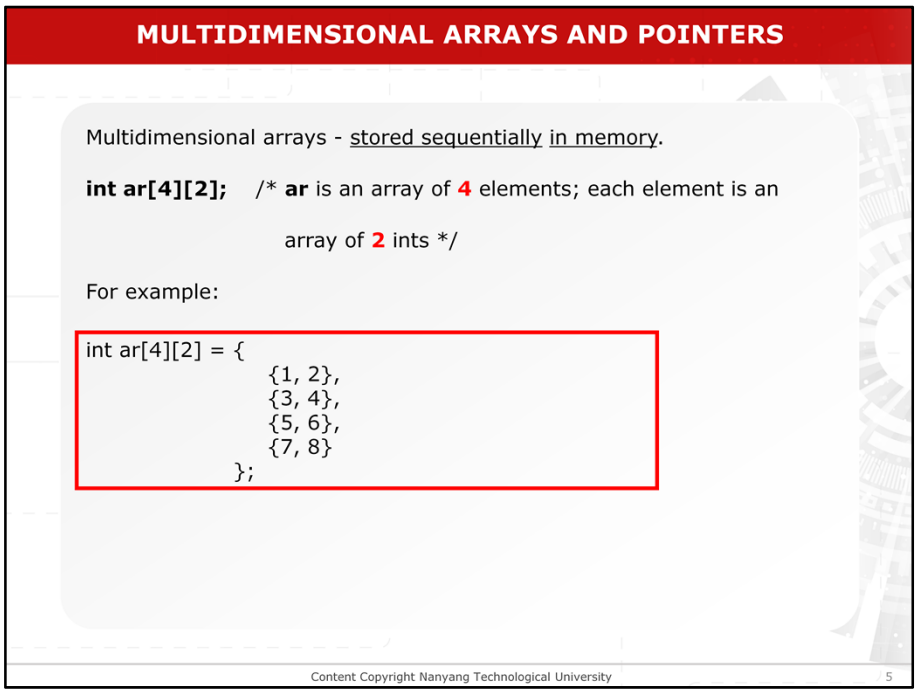
The following are the coverage for 2d ARRAYS. this video focusses on Multidimensional arrays and pointers



LEARNING OBJECTIVES: At this lesson, you should be able to:



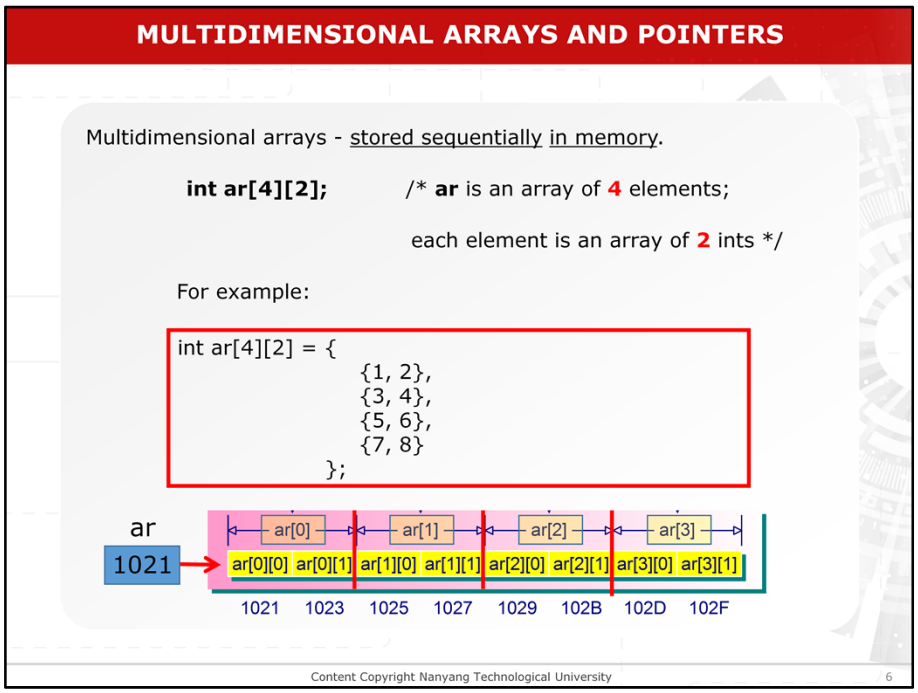
Describe the association between the array elements and the associated pointers.



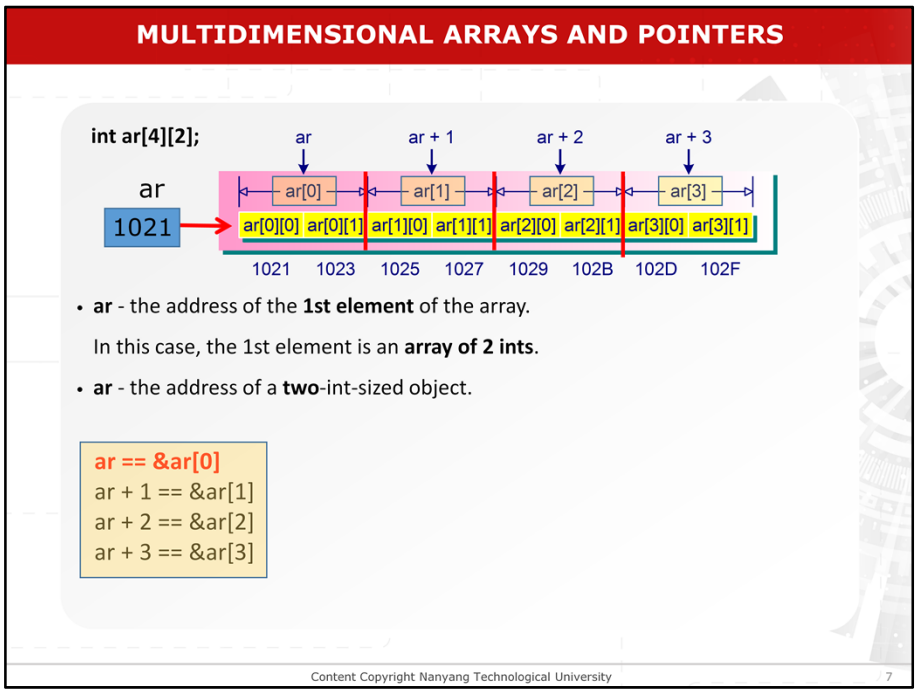
Multi-dimensional Arrays and Pointers

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.

.

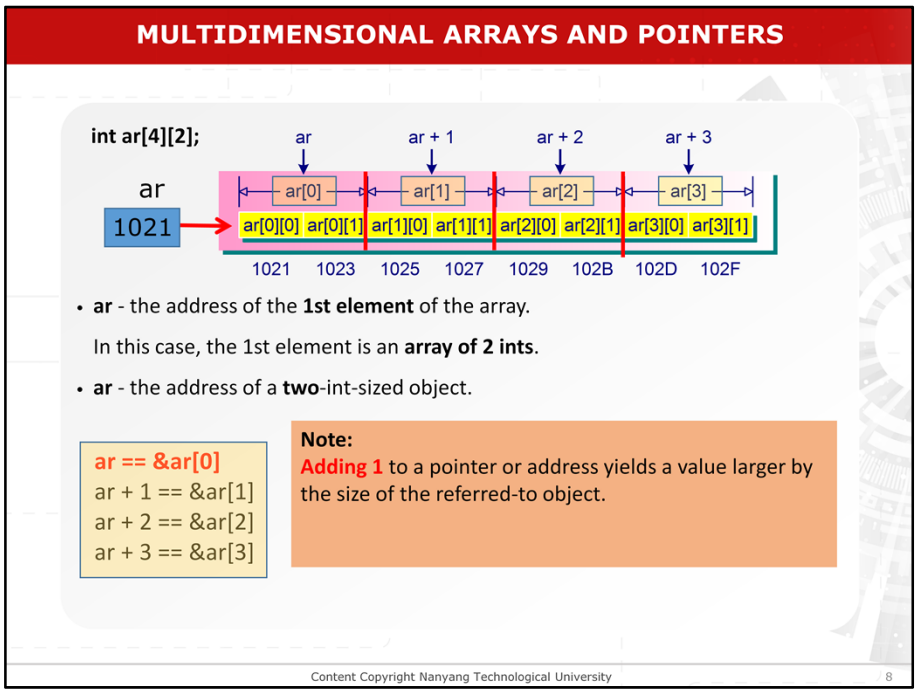


Assume that each integer takes two bytes in the memory, the memory layout of the two-dimensional array is shown with its associated pointers. We will discuss the association between the array elements and the associated pointers in the next few slides.



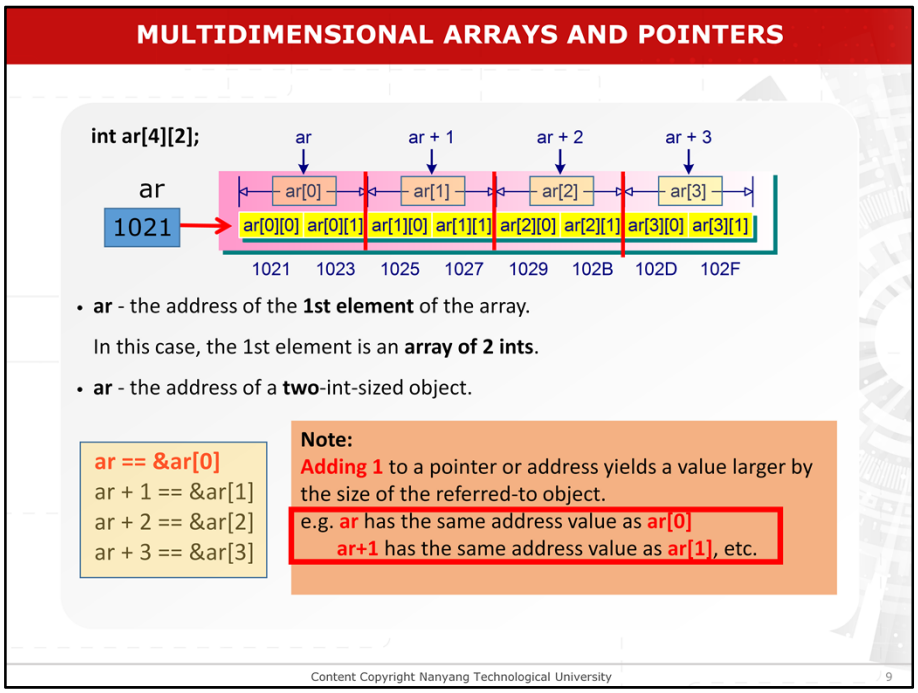
Multi-dimensional Arrays and Pointers

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.

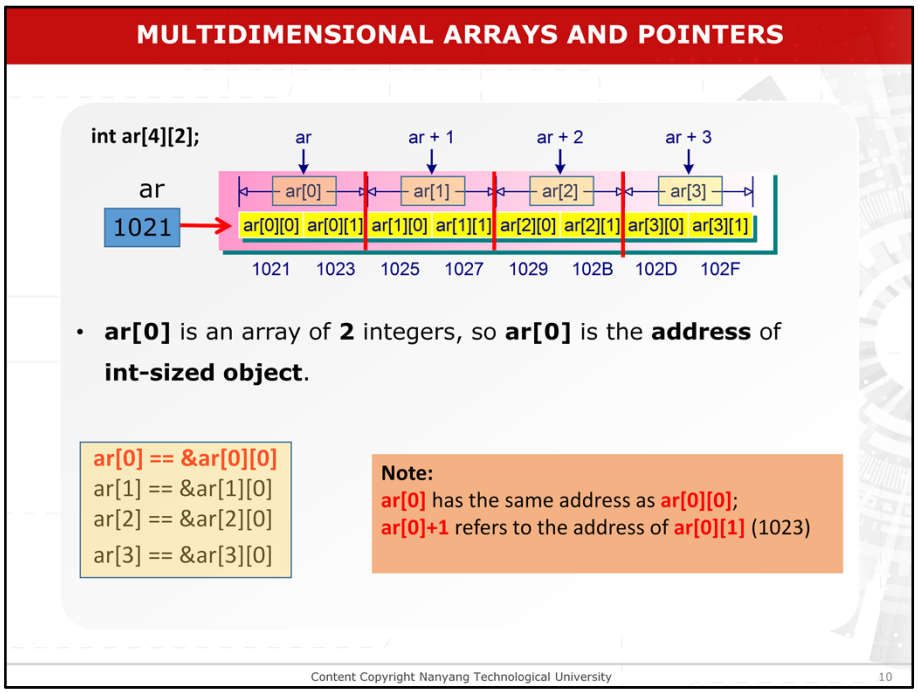


Multi-dimensional Arrays and Pointers

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



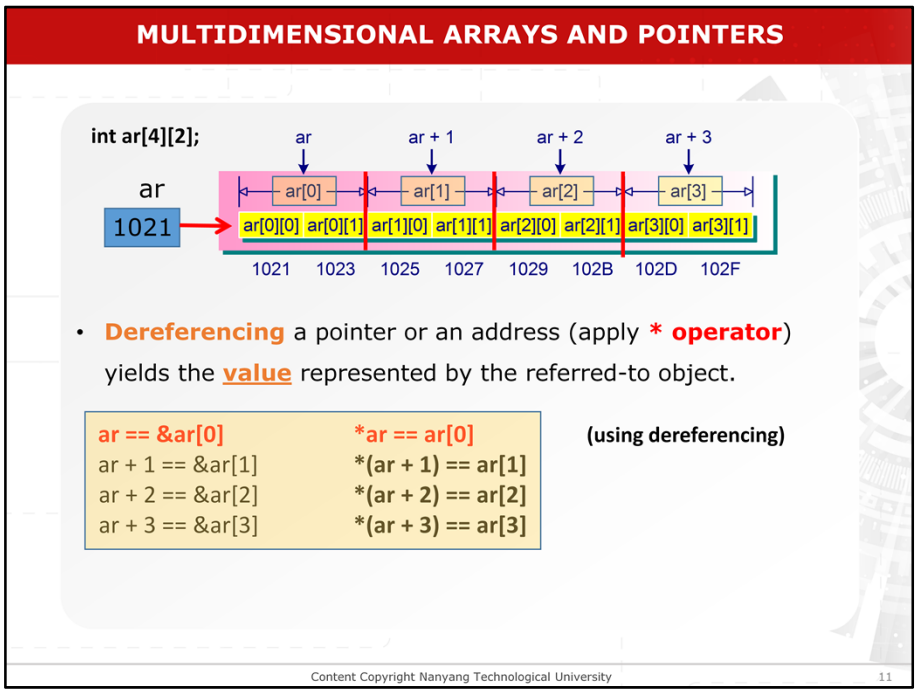
An example is shown here



Multi-dimensional Arrays and Pointers

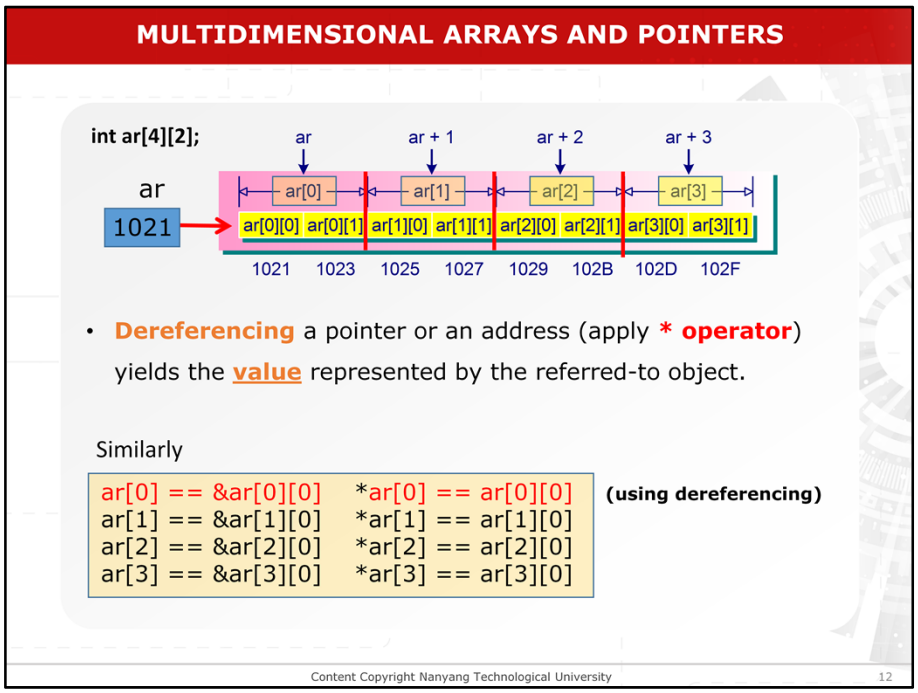
Element of an array is an array of 2 integers, so **element of an array** is the address of an integer sized object. Therefore, we have as shown.

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.



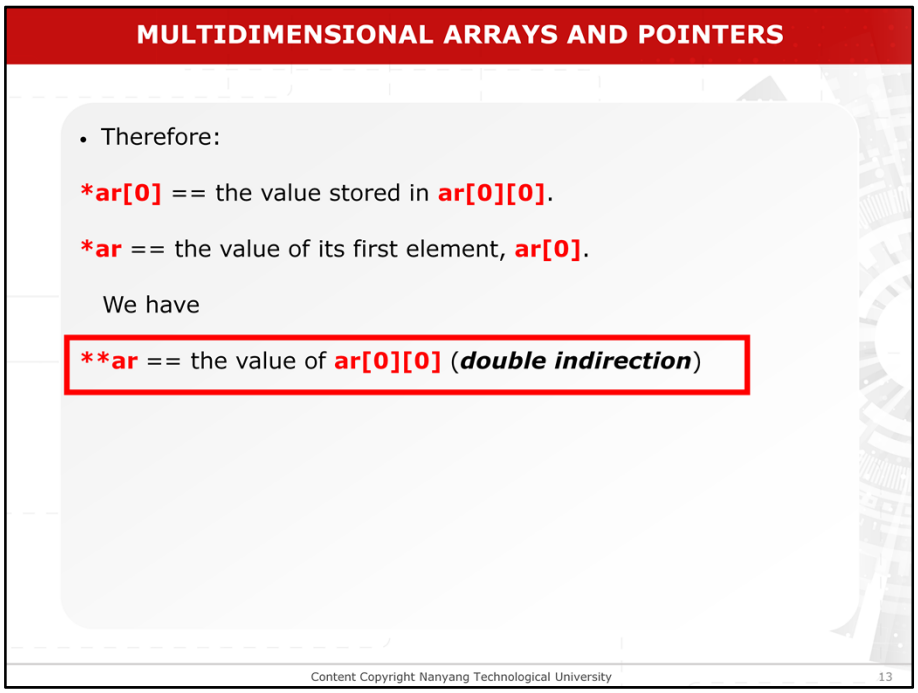
Multi-dimensional Arrays and Pointers

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



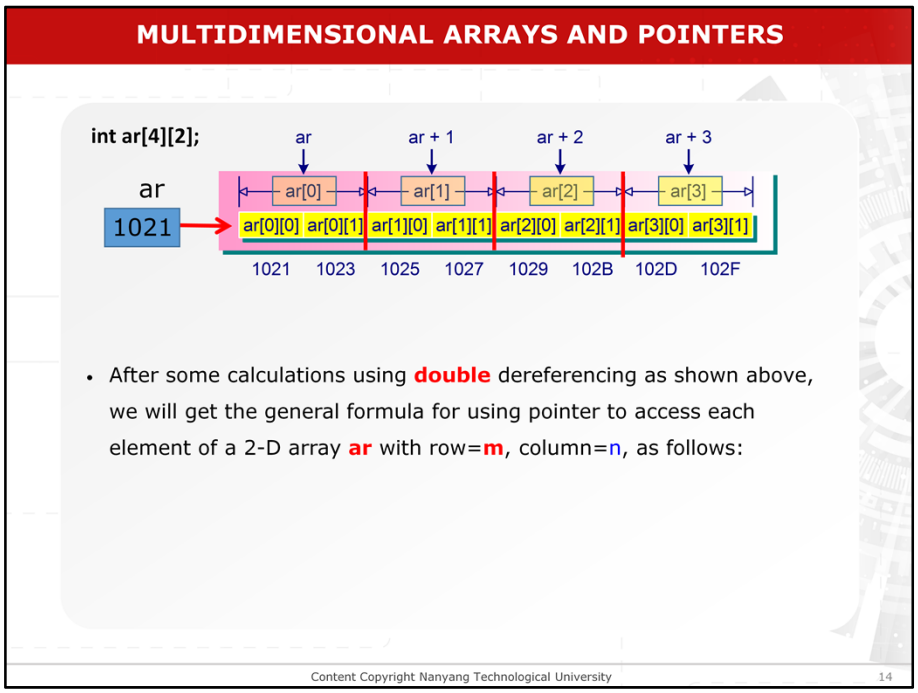
Multi-dimensional Arrays and Pointers

Similarly, we have the following



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*.



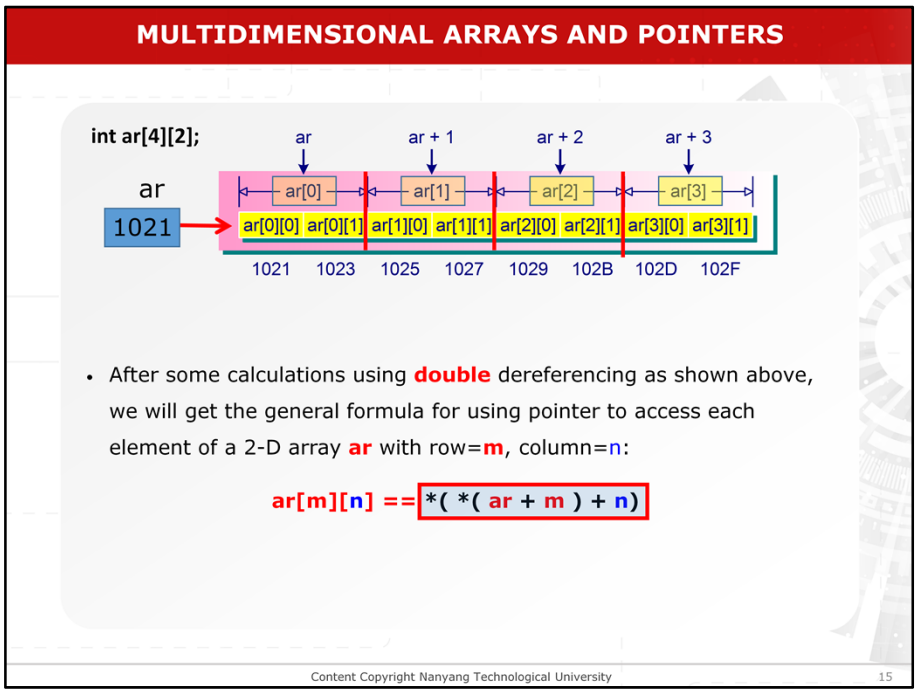
Multi-dimensional Arrays and Pointers

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

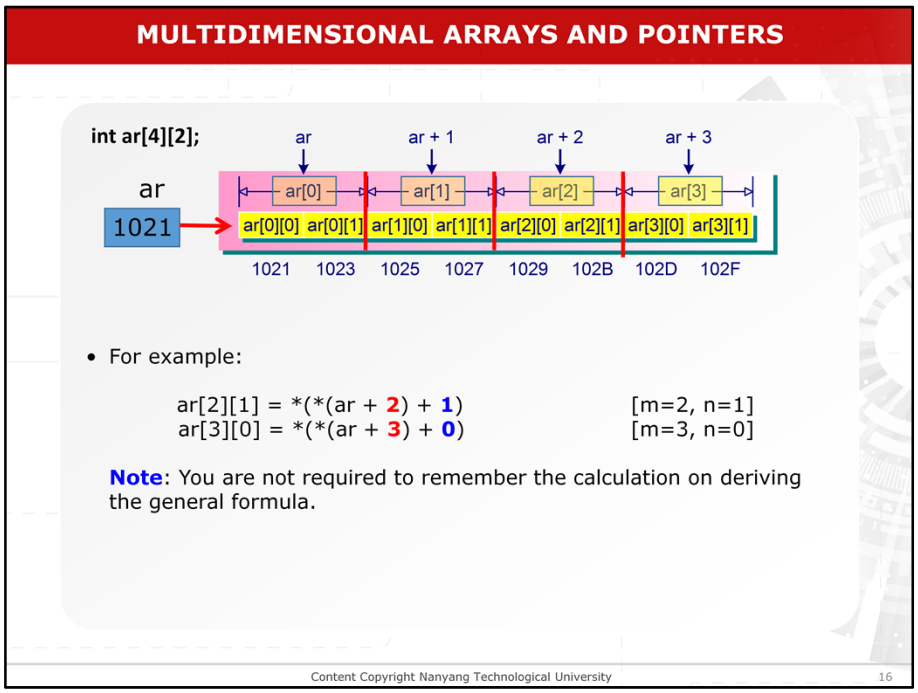
ar[m][n] == *(*(ar+m) + n)

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.



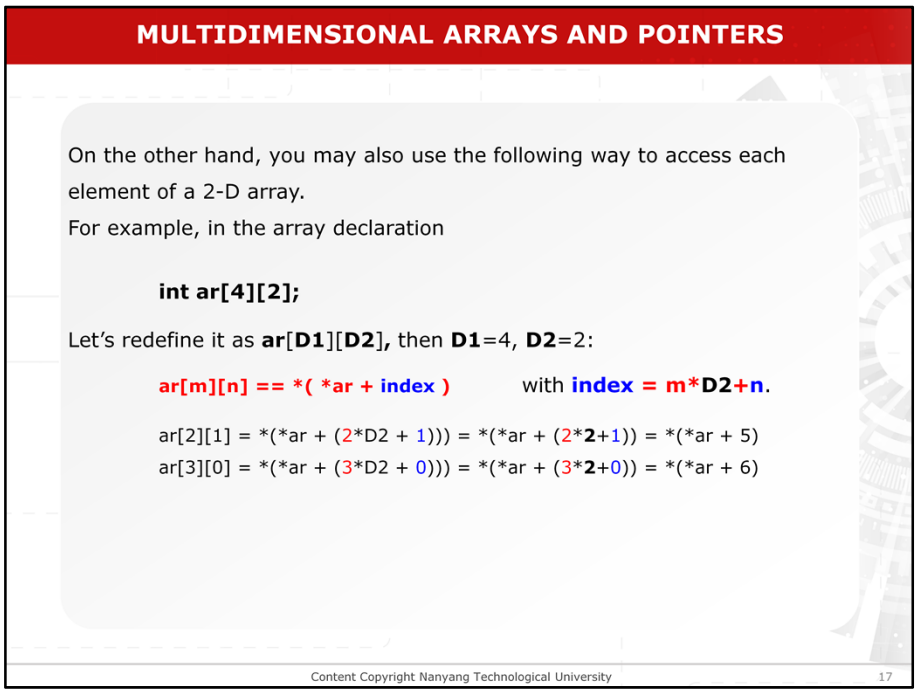
Multi-dimensional Arrays and Pointers

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.



Multi-dimensional Arrays and Pointers

Examples on using the general formula are as shown.



***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

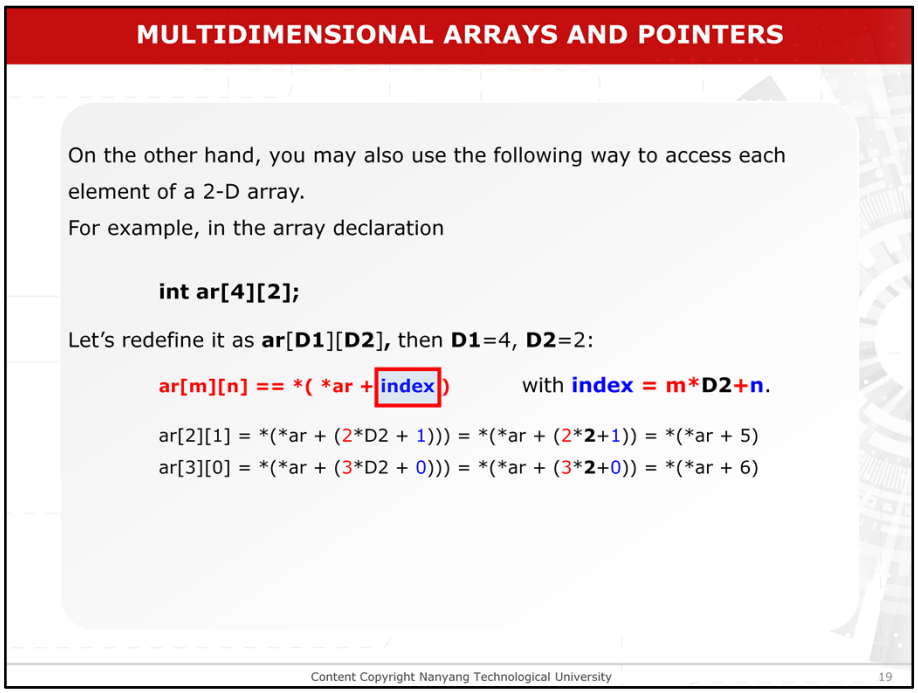
int ar[4][2];

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

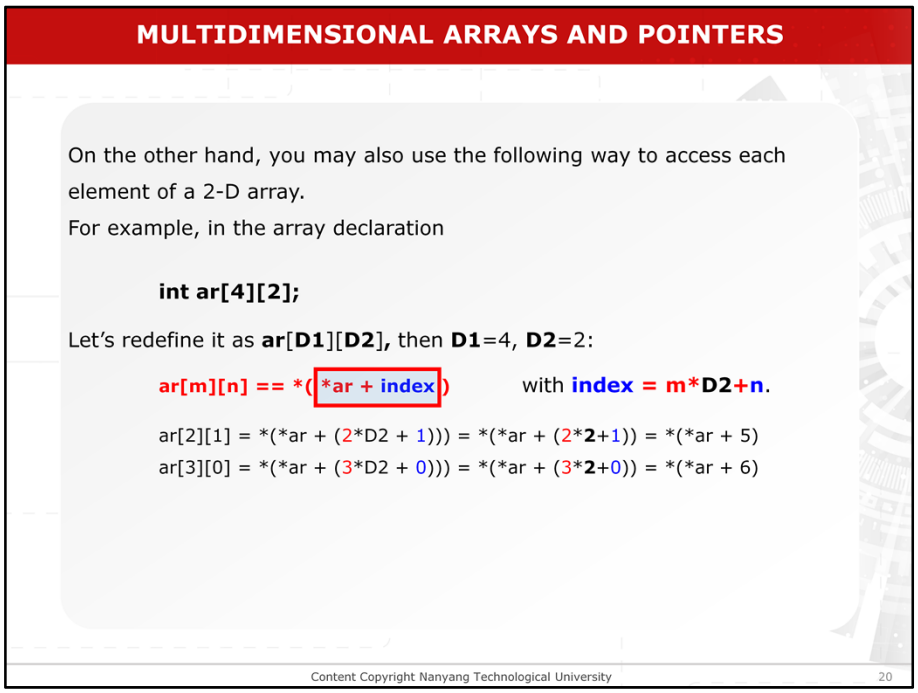
ar[m][n] == *(*ar + index) with **index = m*D2+n**.

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)

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



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



(*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;
}
```


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;
}
```


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;
}
```


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
```


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
```


Using the general formula:

MULTIDIMENSIONAL ARRAYS AND POINTERS: EXAMPLE

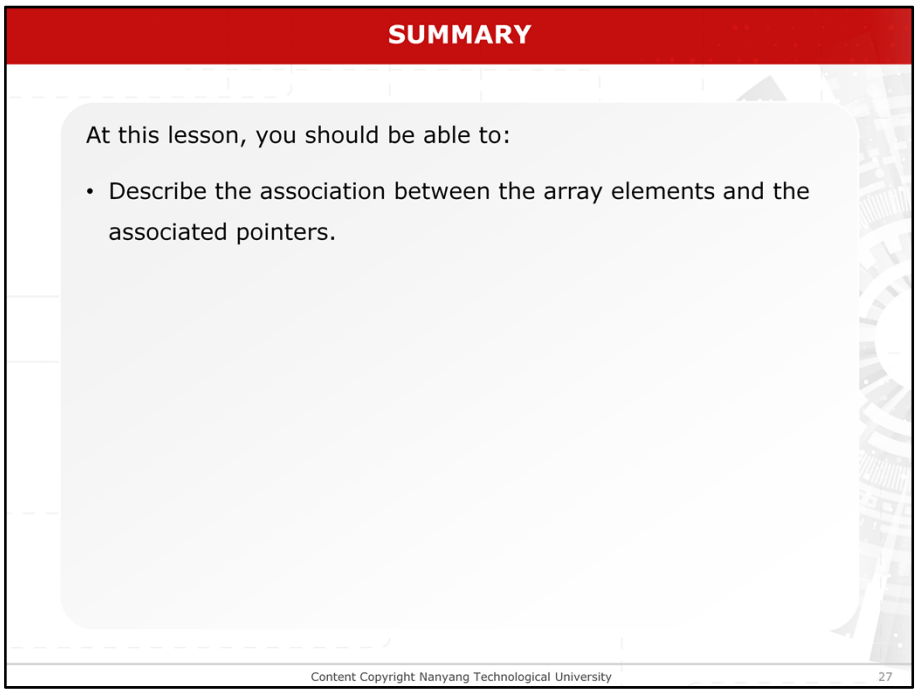
```
#include <stdio.h>
int main() {
  int ar[3][3]=  {
     {5, 10, 15},
     {10, 20, 30},
     {20, 40, 60}
  };
  int index, i, j;
   // (1) using indexes
  for (i=0; i<3; i++)
     for (j=0; j<3; j++)
        printf("%d ", ar[i][j]);
  printf("\n");
    // (2) using the general formula ar[m][n] ==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

 26

- Using the other general formula:



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