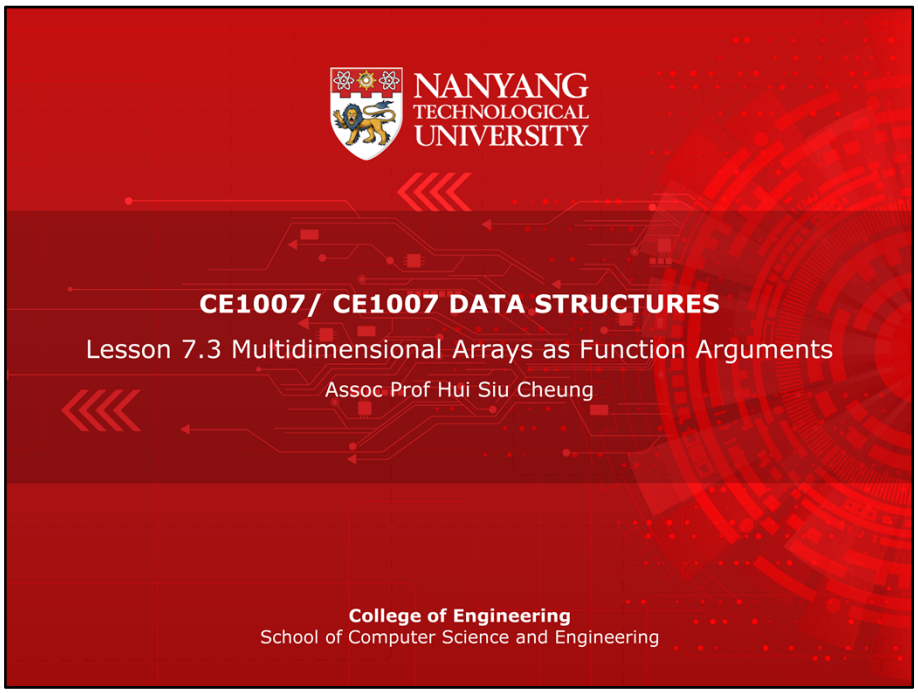
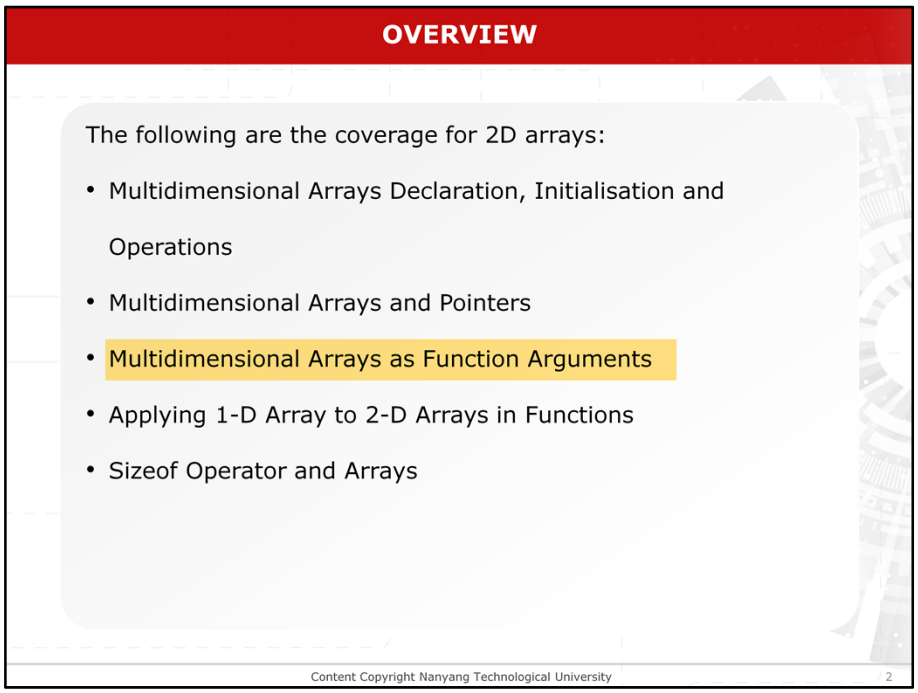
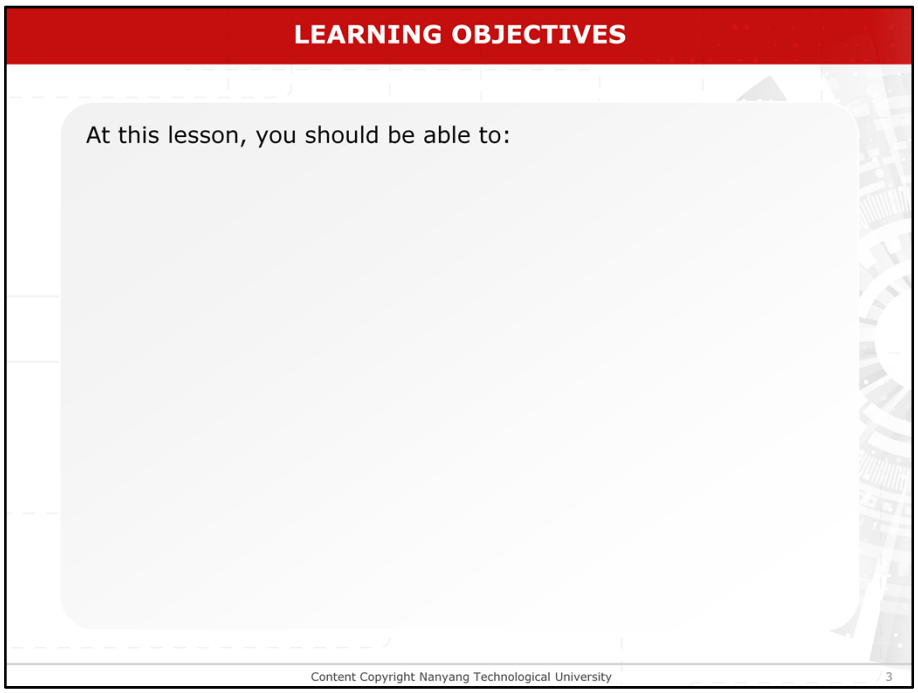
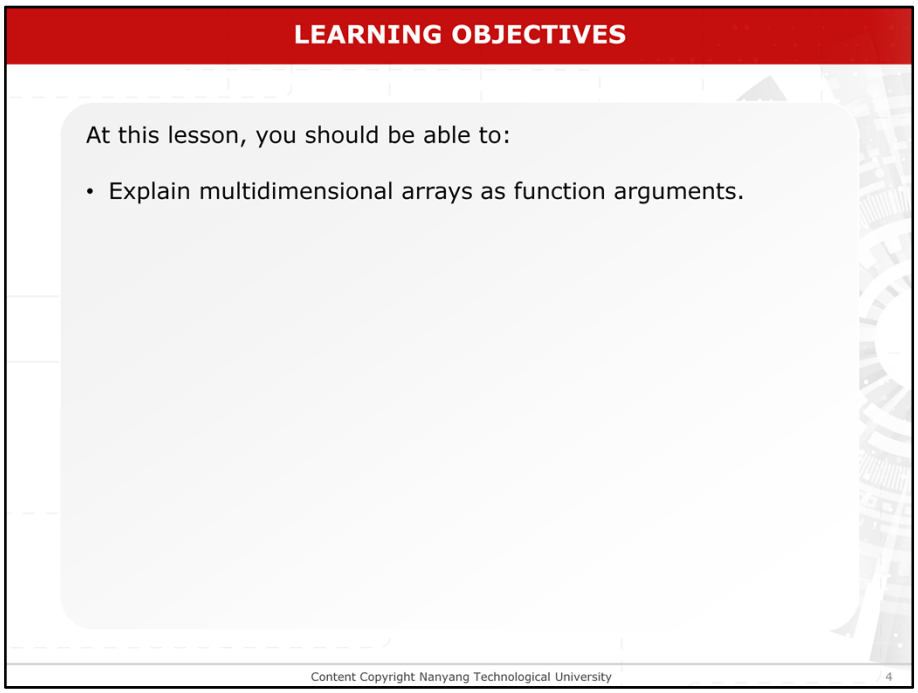

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CE1007 DATA STRUCTURES
Lesson 7.3 Multidimensional Arrays as Function Arguments
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering



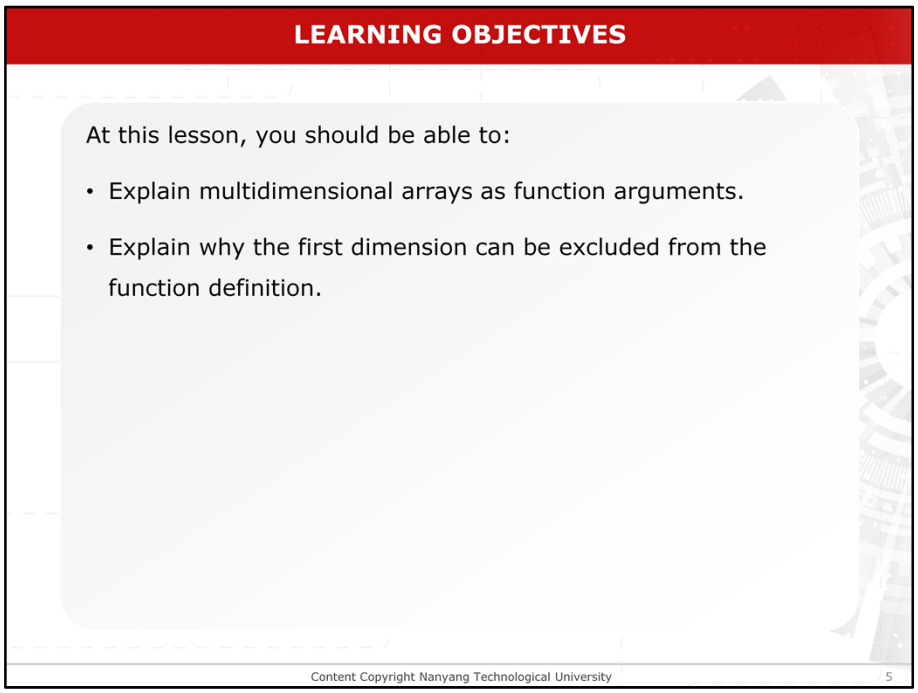
The following are the coverage for 2d ARRAYS. this video focusses on Multidimensional arrays as function arguments



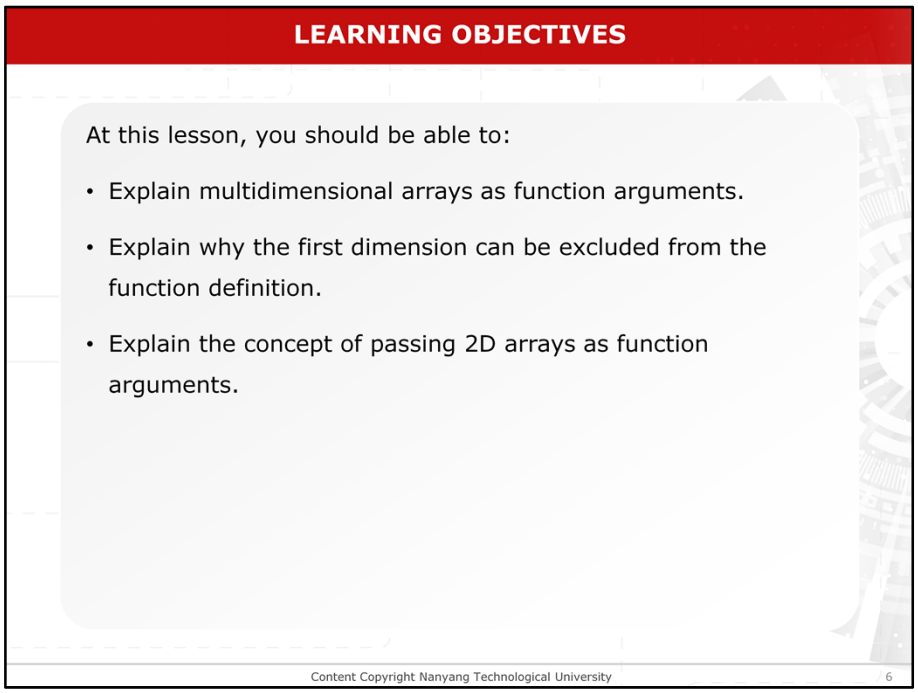
Learning Objectives: At this lesson, you should be able to:



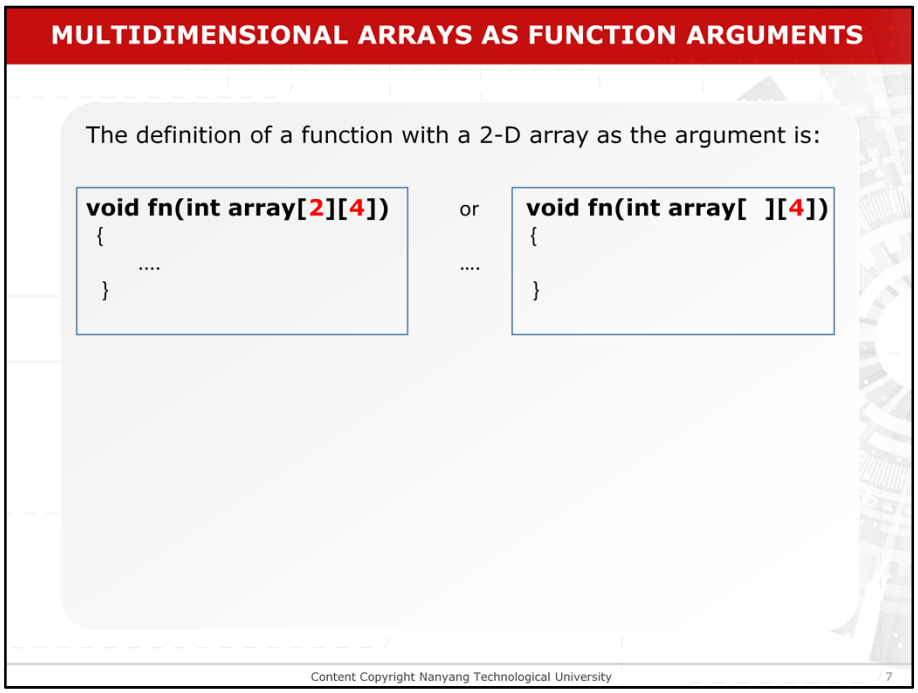
- Explain multi-dimensional arrays as function arguments.



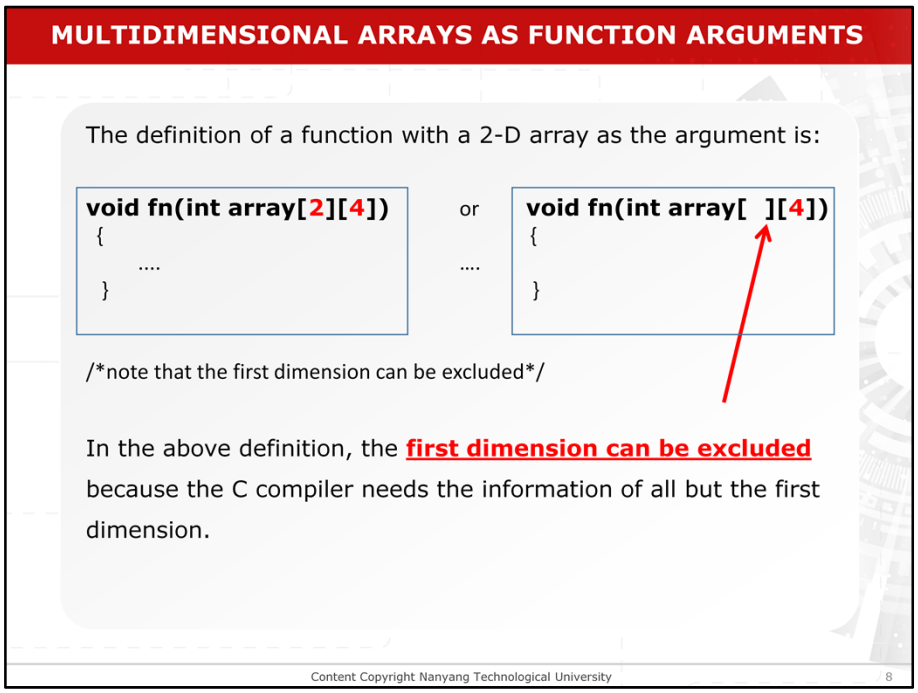
Explain why the first dimension can be excluded from the function definition.



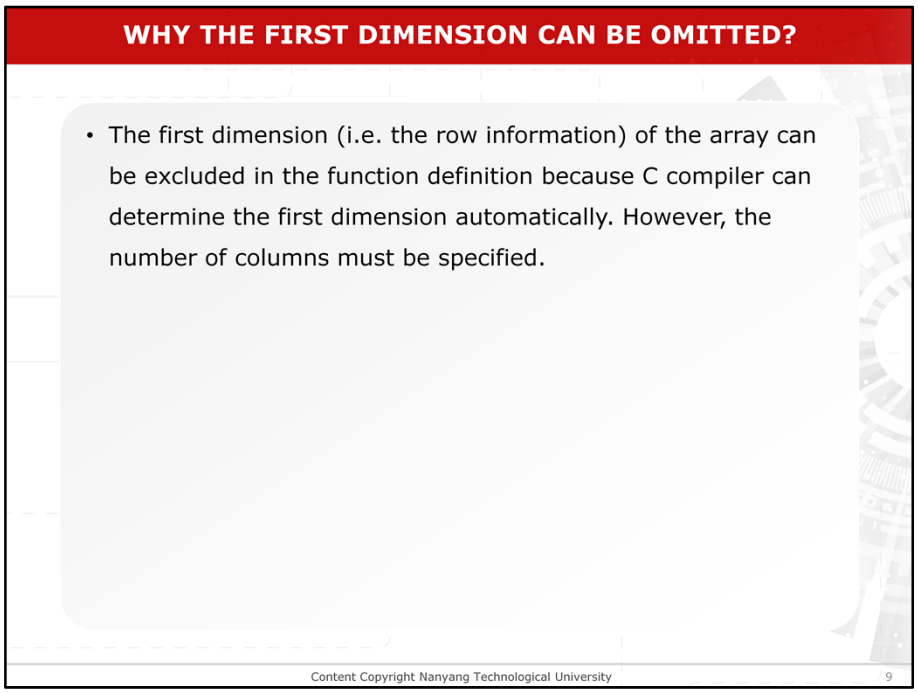
Explain the concept of passing 2D arrays as function arguments.



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

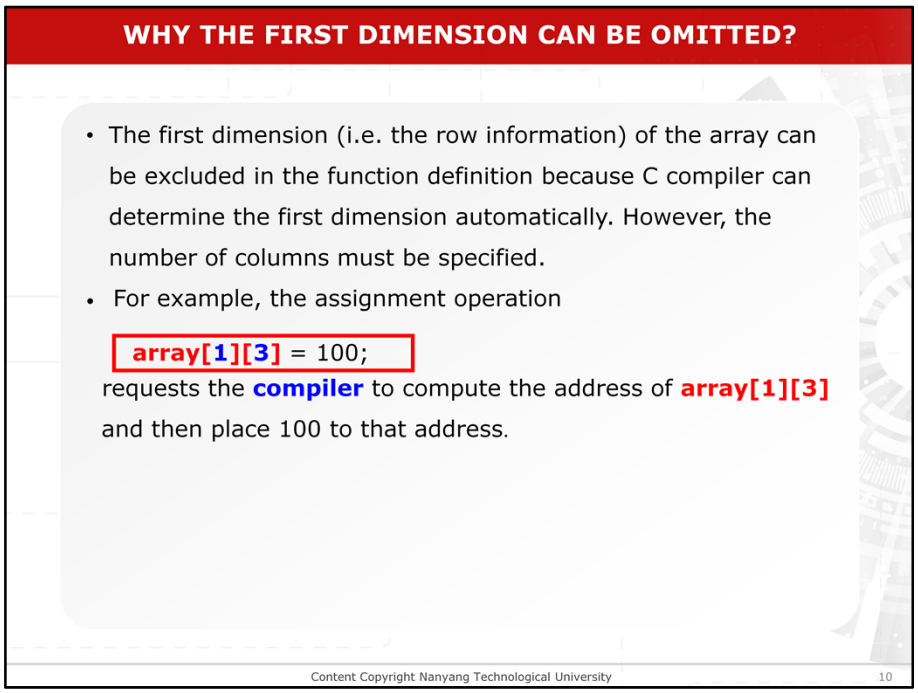


In the above definition, the **first dimension can be excluded** because the C compiler needs the information of all but the first dimension.

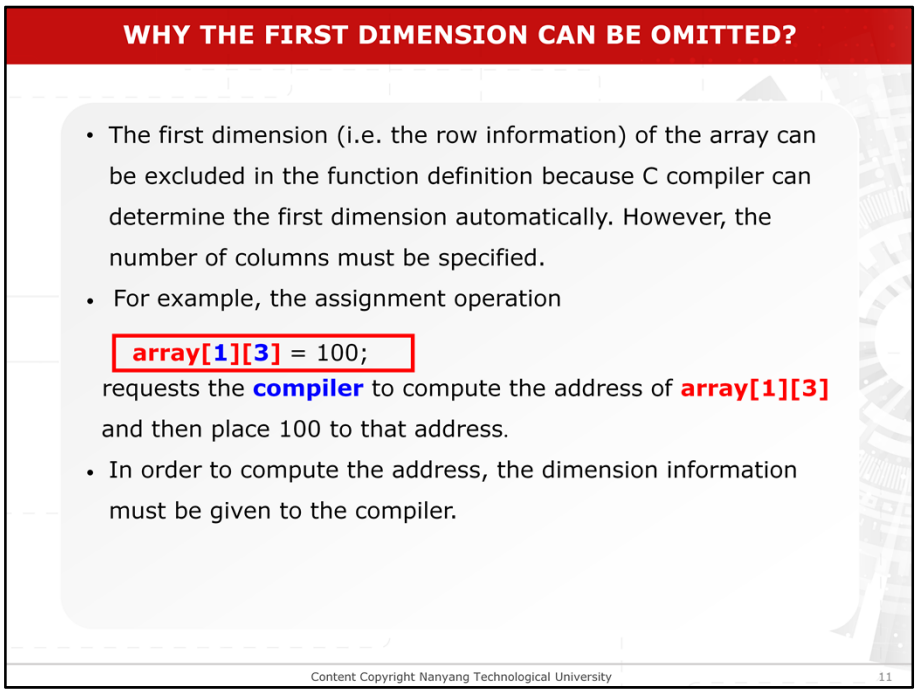


Why the First Dimension can be Omitted?

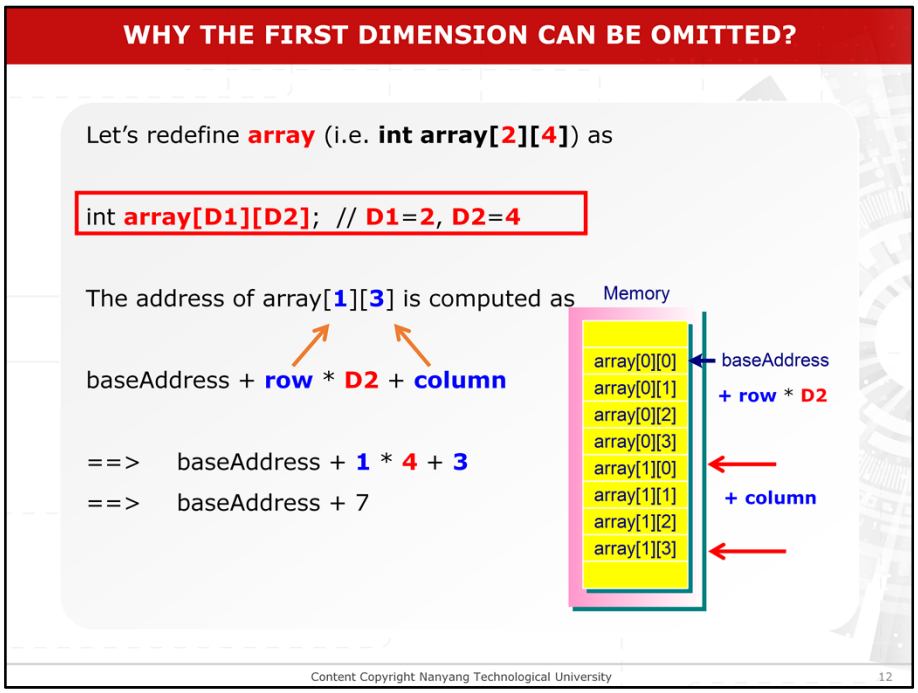
The first dimension that is the row information of the array can be excluded in the function definition because C compiler can determine the first dimension automatically. However, the number of columns must be specified.



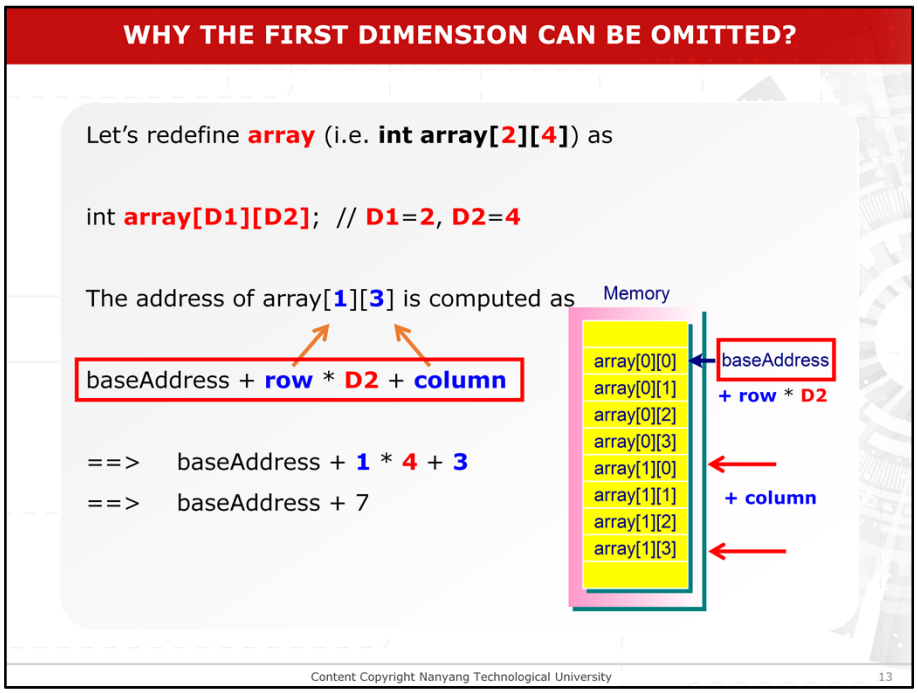
For example, the assignment statement shown requests the compiler to compute the address of and then places a value of 100 to that address.



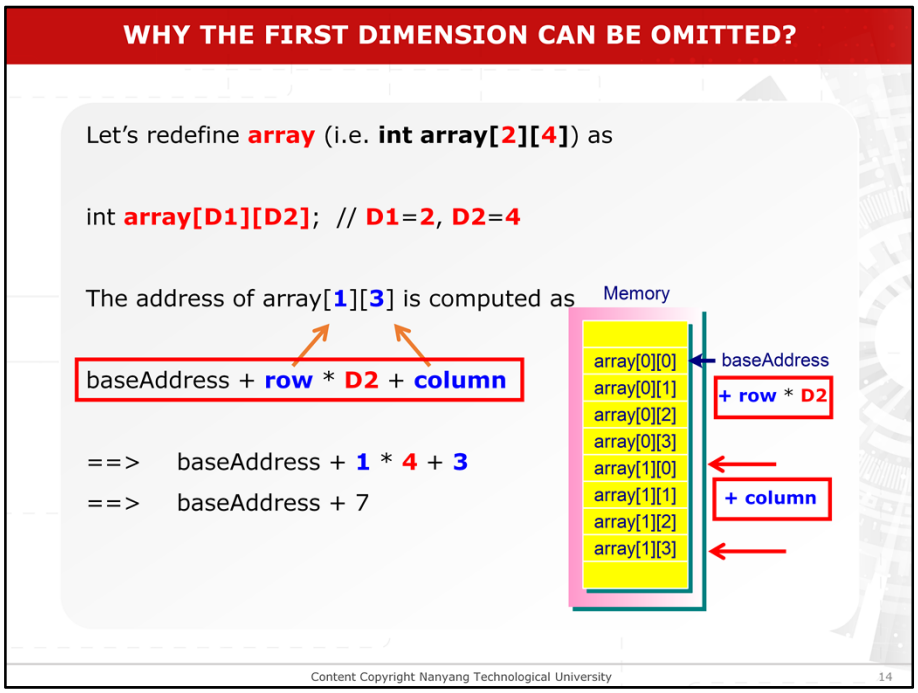
In order to compute the address, the dimension information must be given to the compiler



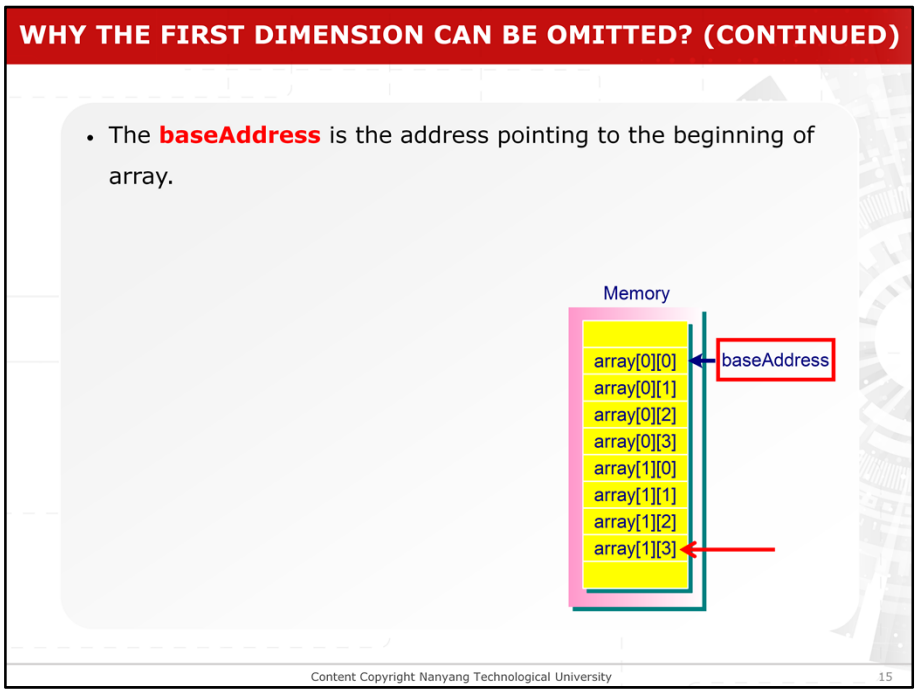
Let us redefine **array** as shown here.



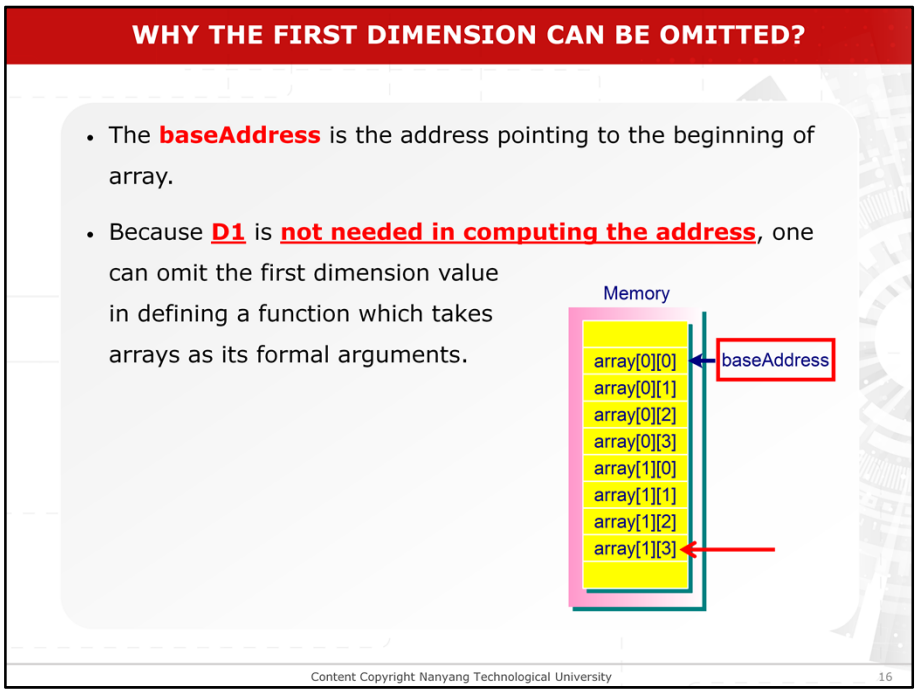
The address of **array with 1 row and 3 column** is computed as shown where the base Address is the address pointing to the beginning of **array**.



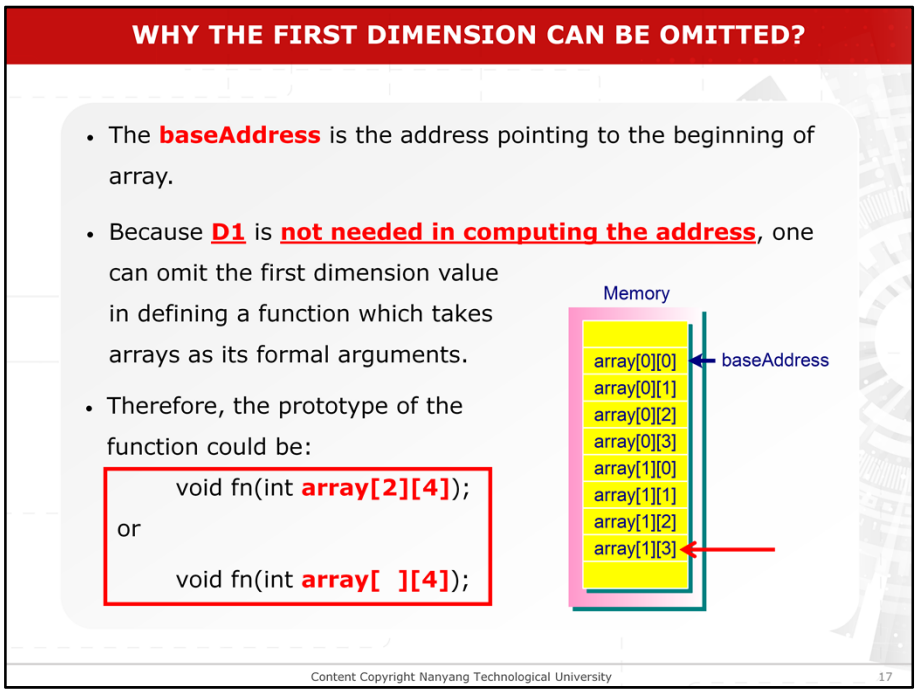
The address of array one three is computed as shown here.



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

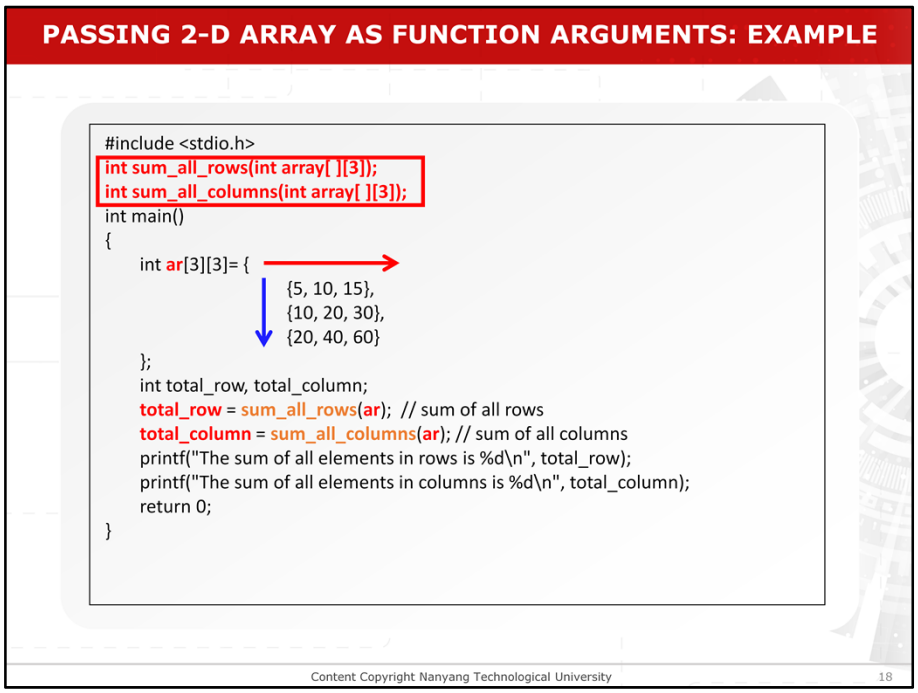


Because **D1** is **not needed in computing the address**, one can omit the first dimension value in defining a function which takes arrays as its formal arguments.



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

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;
}
```


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;
}
```


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

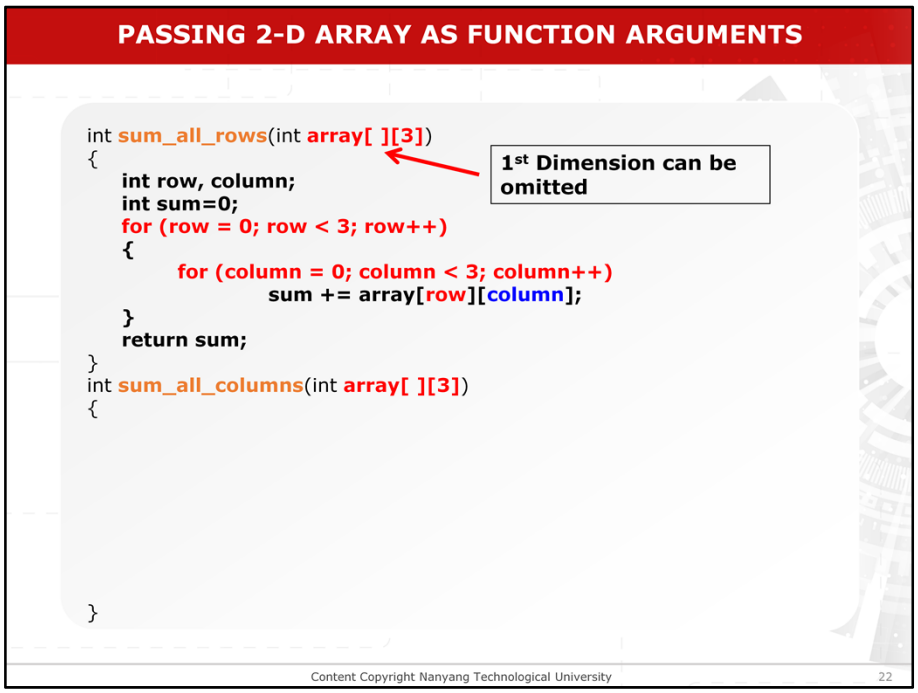
The total values are computed in the two functions and placed in the two variables **total row** and **total column** respectively.

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;
}
```

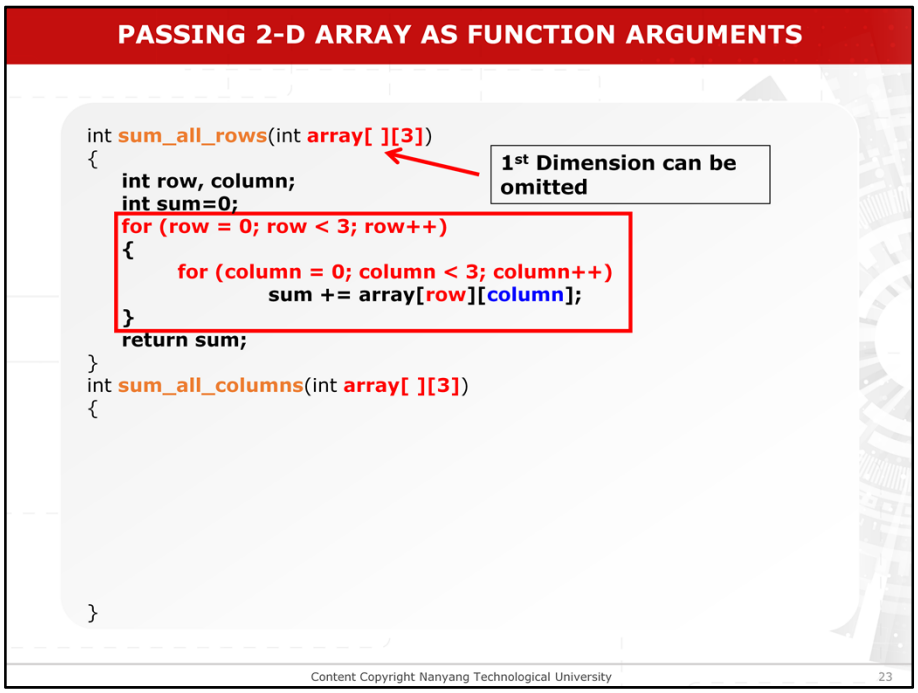
Output
The sum of all elements in rows is 210
The sum of all elements in columns is 210

The output of the program is shown here.



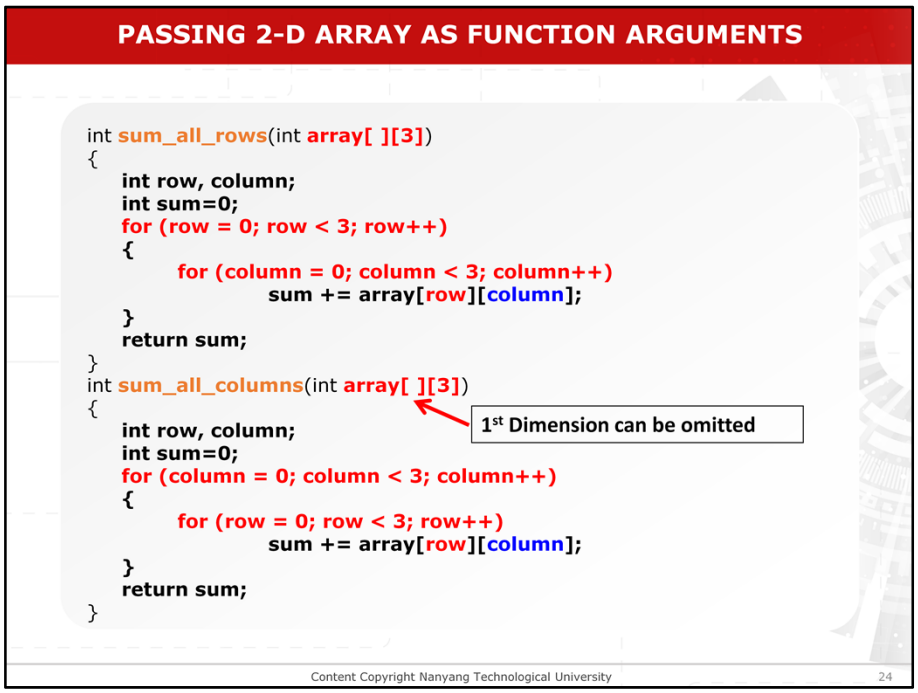
Passing 2-D Array as Function Arguments: Example

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



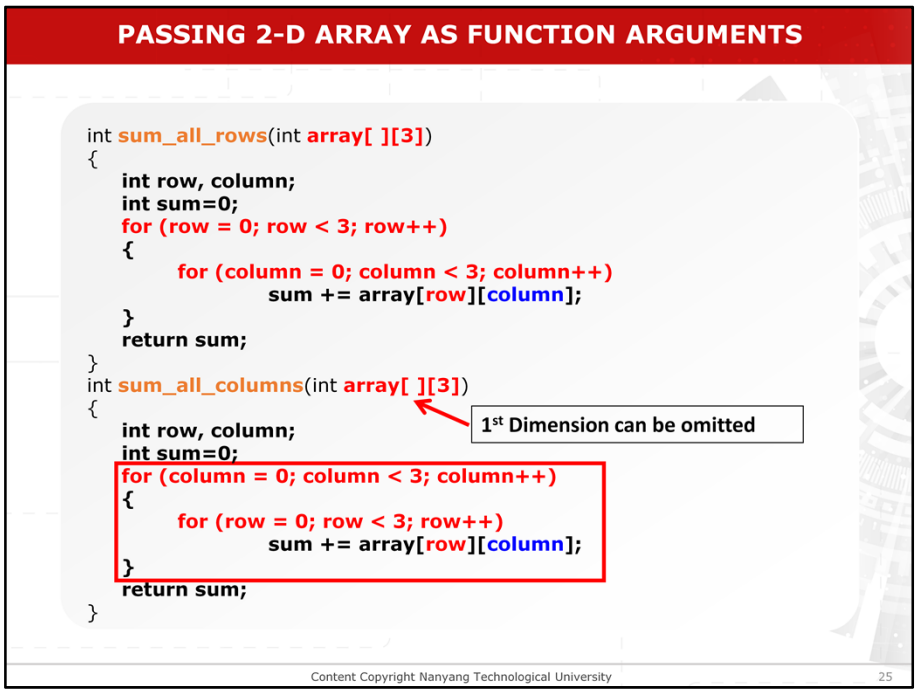
Passing 2-D Array as Function Arguments: Example

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: 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: Example

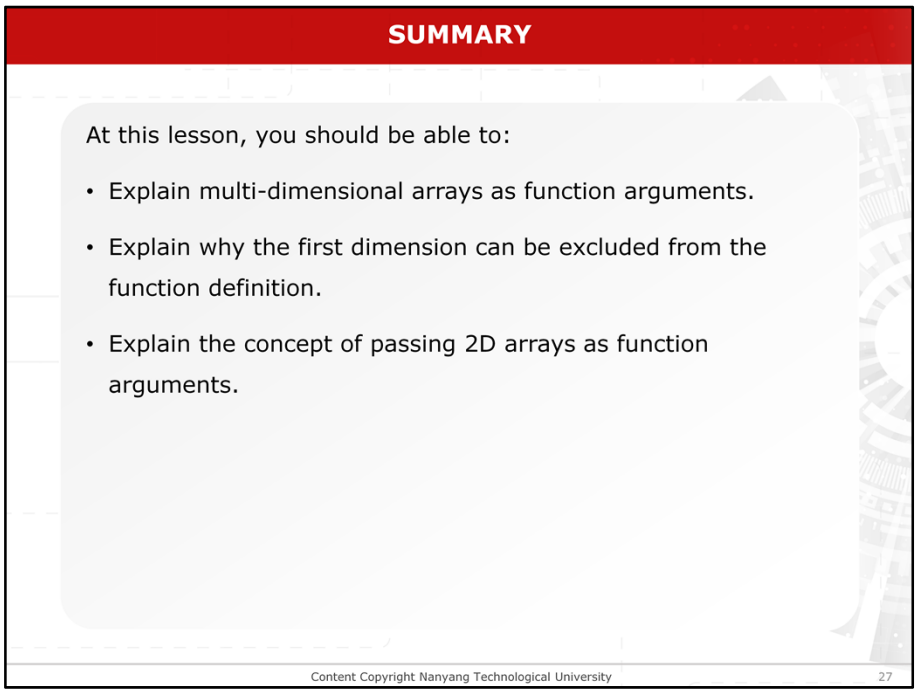
A nested **for** loop is used to traverse the 2-dimensional array in order to compute the sum of all columns.

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;
}
```

1st Dimension can be omitted

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



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