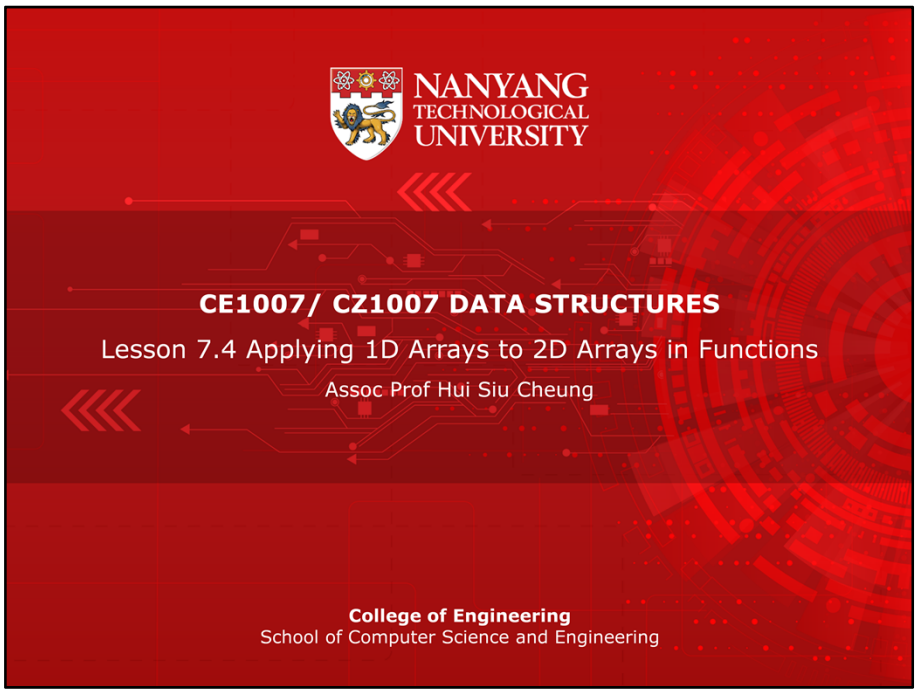
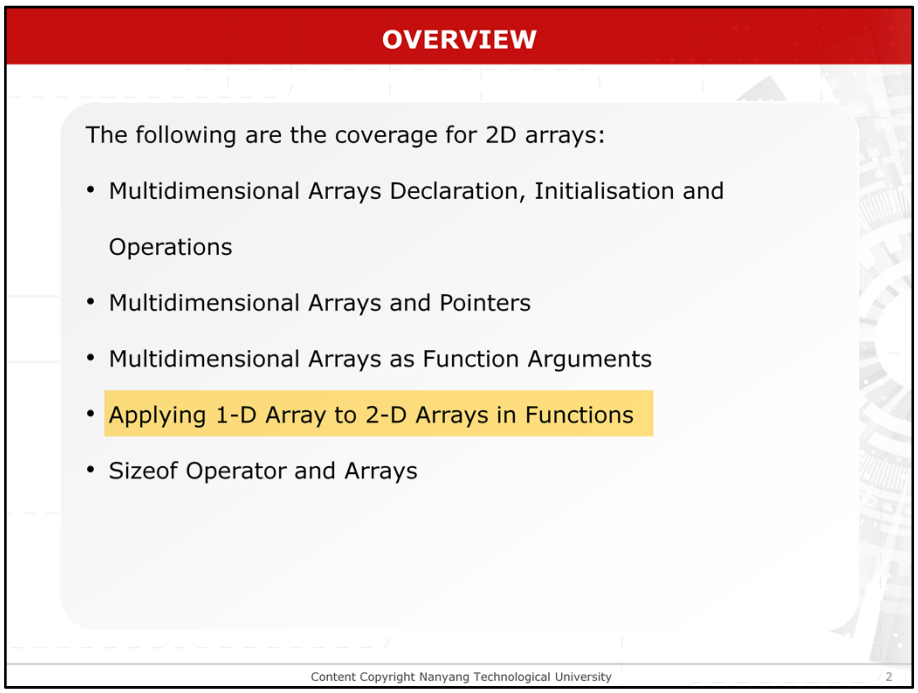
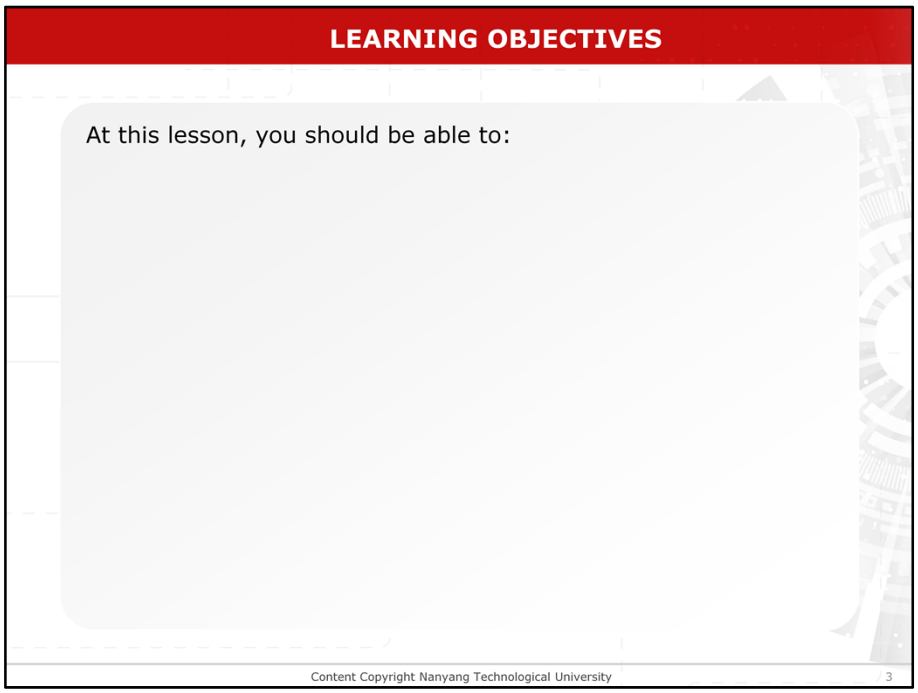
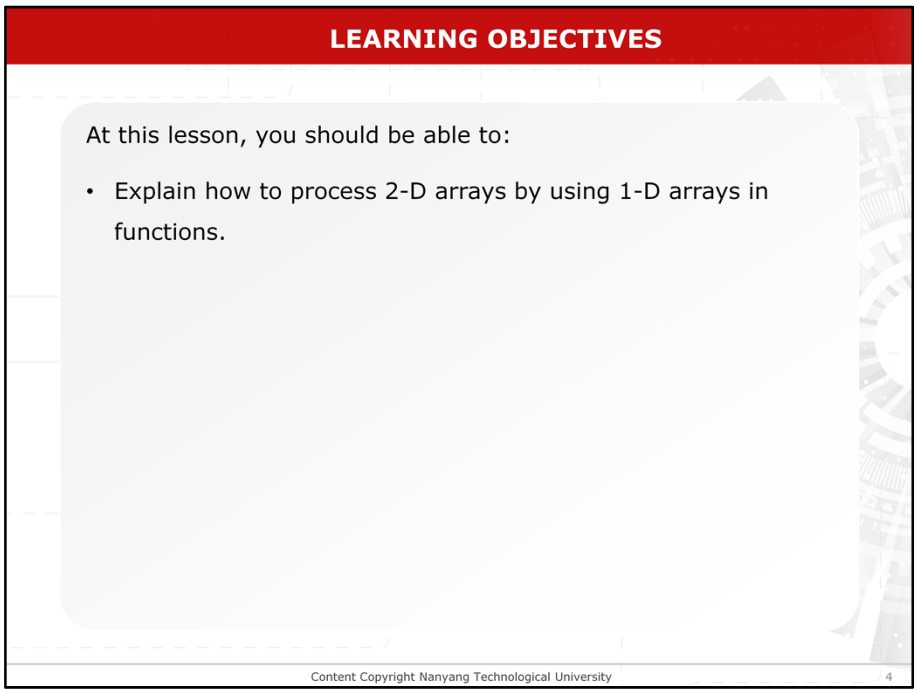

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 7.4 Applying 1D Arrays to 2D Arrays in Functions
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering



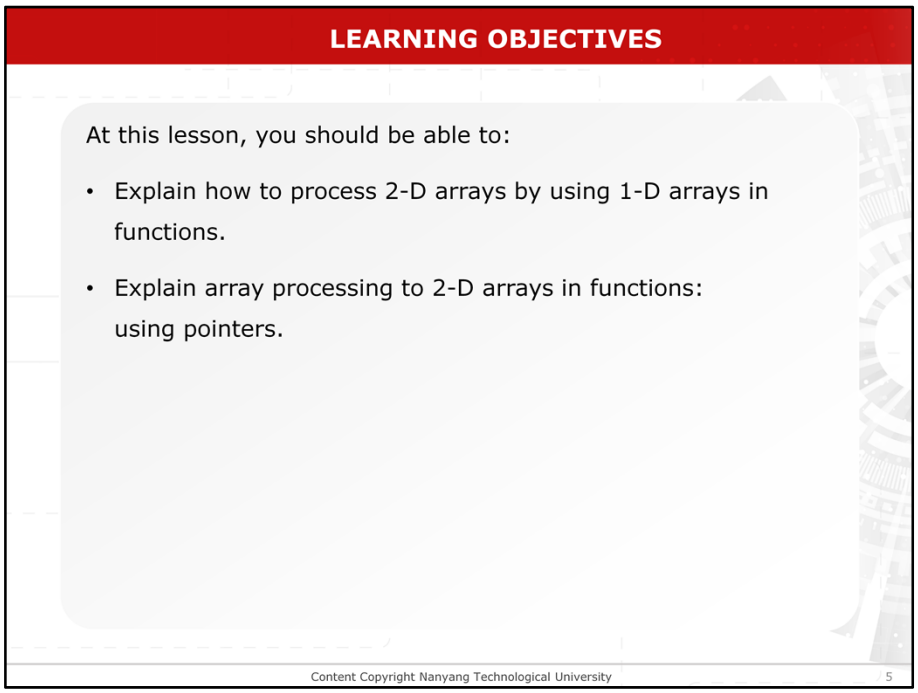
The following are the coverage for 2 ARRAYS. this video focusses on applying 1 arrays to 2 arrays IN FUNCTIONS



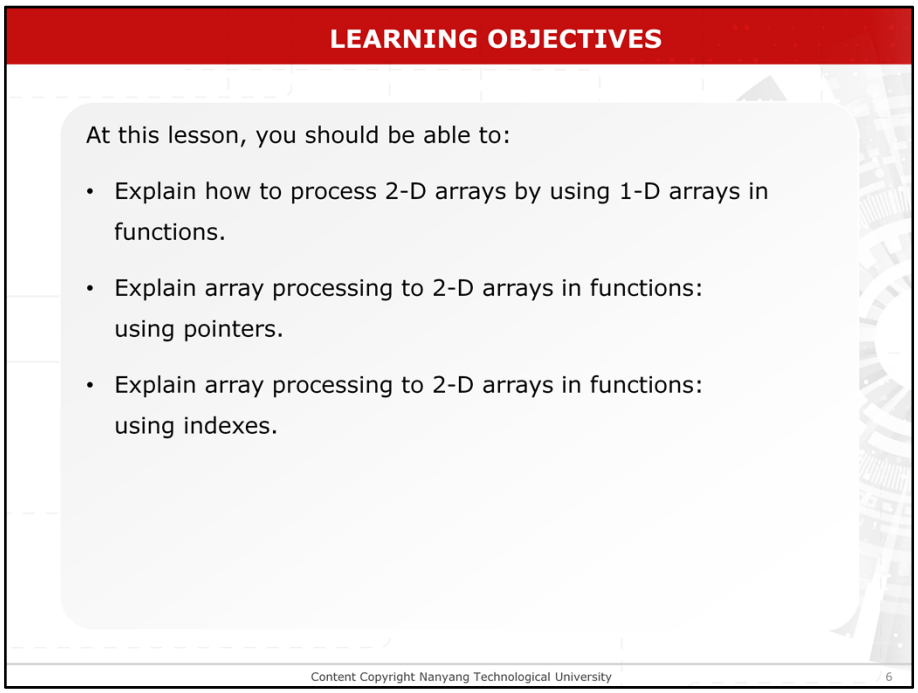
Learning Objectives: At this lesson, you should be able to:



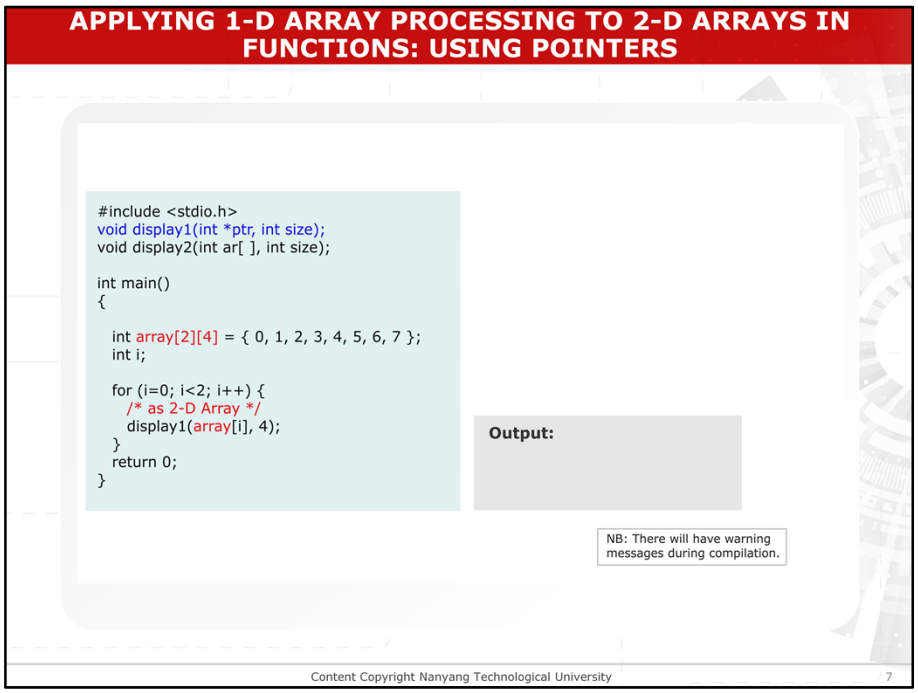
- Explain how to process 2 DIMENSIONAL arrays by using 1 DIMENSIONAL arrays in functions.



- Explain Array processing to 2 dimensional arrays in functions: using pointers.

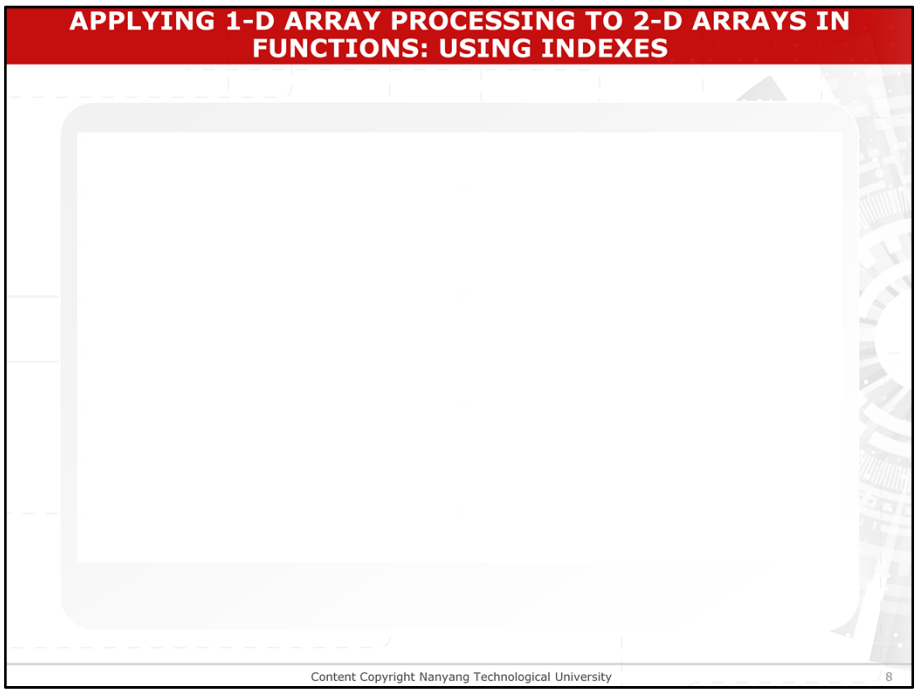


- Explain Array processing to 2 dimensional arrays in functions: using indexes



A function that is written for processing one-dimensional arrays can be used to deal with two-dimensional arrays. In the program, **array** is an array of 2 by 4 integers. The function **display 1** is written to access the elements of the array with the specified **size** and prints the contents to the screen. In **display 1**, it accepts a pointer variable and accesses the elements of the array using the pointer variable.

In the **for** loop of the **main** function, when **i=0**, we pass **element of an array** to **display1**. Element of an **array** corresponds to the address of **array with 0 row and 0 column** (i.e. **&array with 0 row and 0 column**). The function then accesses the array starting from the location **array with 0 row and 0 column**, and prints the 4 elements out to the display as specified in the function. When **i=1**, **array of 1** is passed to **display1**. Now, **array of 1** corresponds to the address of **array with 1 row and 0 column** (i.e. **&array with 1 row and 0 column**). The function then accesses the 4 elements starting from **array with 1 row and 0 column**, and print the contents of the 4 elements.



Applying 1-D Array Processing to 2-D Arrays – using Indexes

In the program, **array** is an array of 2 by 4 integers. The function **display2()** is written to access the elements of the array with the specified **size** and prints the contents to the screen.

In **display2**, it accepts the array pointer and uses array index to access the elements of the array. In the **for** loop of the **main()** function, when **i=0**, we pass **Element of an array** to **display2**. **Element of an array** corresponds to the address of **array with 0 row and 0 column .**The function then accesses the 4 elements of the array starting from the location **array with 0 row and 0 column**, and prints the results to the display as specified in the function. When **i=1**, **array of 1** is passed to **display2**. Now, **array of 1** corresponds to the address of **array with 1 row and 0 column.** The function then accesses the 4 elements starting from **array with 1 row and 0 column**, and prints the results according to the function.

We can also view **array** as an array of 8 integers. When we pass **array** as an argument to the function **display 2** with d**isplay 2(array, 8);**

the array **ar** in the function **display 2** is initialized to the address of **array with 0 row and 0 column**. As a result, **element of an array** corresponds to **array of 0 row and 0 column**, while **array of 1** corresponds to **array with 0 row and 1 column**. Similarly, **array** correspond to **array with 1 row and 0 column**, and so on. Therefore, all the elements of the two-dimensional array can be accessed and printed to the screen.

EXAMPLE: MINMAX()

Write a C function minMax() that takes a 5x5 two-dimensional array of integers *a* as a parameter. The function returns the minimum and maximum numbers of the array back to the caller through the two parameters *min* and *max* respectively.
[using call by reference]

```
#include <stdio.h>
void minMax(int a[5][5], int *min, int *max);
int main()
{
    int A[5][5];
    int i, j;
    int min, max;
    printf("Enter your matrix data (5x5): \n");
    // nested loop
    for (i=0; i<5; i++)
      for (j=0; j<5; j++)
        scanf("%d", &A[i][j]);
    minMax(A, &min, &max);
    printf("min = %d; max = %d", min, max);
    return 0;
}
```


In this application example, you are required to write a C function **minMax** that takes a 5x5 two-dimensional array of integers *a* as a parameter.

EXAMPLE: MINMAX()

Write a C function minMax() that takes a 5x5 two-dimensional array of integers *a* as a parameter. The function returns the minimum and maximum numbers of the array back to the caller through the two parameters *min* and *max* respectively.
[using call by reference]

```
#include <stdio.h>
void minMax(int a[5][5], int *min, int *max);
int main()
{
    int A[5][5];
    int i, j;
    int min, max;
    printf("Enter your matrix data (5x5): \n");
    // nested loop
    for (i=0; i<5; i++)
      for (j=0; j<5; j++)
        scanf("%d", &A[i][j]);
    minMax(A, &min, &max);
    printf("min = %d; max = %d", min, max);
    return 0;
}
```


The function returns the minimum and maximum numbers of the array back to the caller through the two parameters *min* and *max* respectively. Call by reference is used for passing the results on maximum and minimum numbers to the calling function.

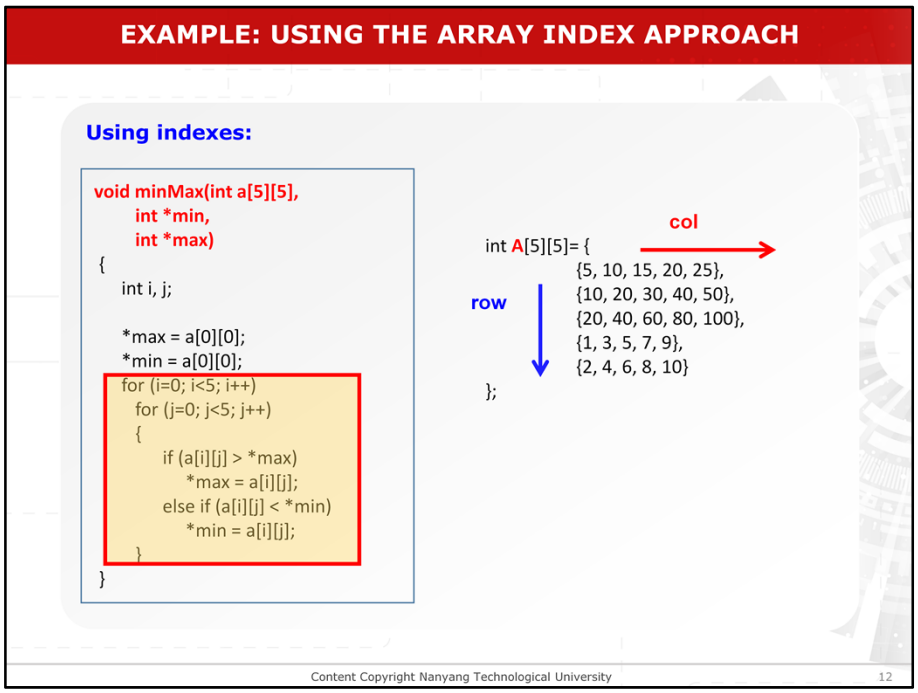
EXAMPLE: MINMAX()

Write a C function minMax() that takes a 5x5 two-dimensional array of integers *a* as a parameter. The function returns the minimum and maximum numbers of the array back to the caller through the two parameters *min* and *max* respectively.
[using call by reference]

```
#include <stdio.h>
void minMax(int a[5][5], int *min, int *max);
int main()
{
    int A[5][5];
    int i, j;
    int min, max;
    printf("Enter your matrix data (5x5): \n");
    // nested loop
    for (i=0; i<5; i++)
      for (j=0; j<5; j++)
        scanf("%d", &A[i][j]);
    minMax(A, &min, &max);
    printf("min = %d; max = %d", min, max);
    return 0;
}
```

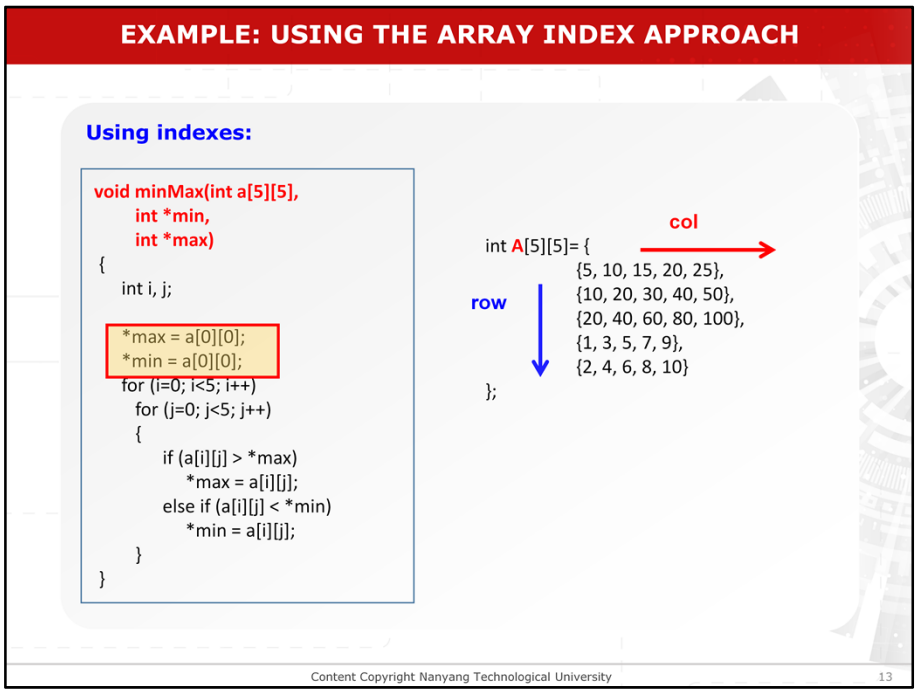
```
void minMax(int a[5][5], int *min,
            int *max)
{
   int i, j;

   /* add your code here */

      Q: Using index?

      Q: Using pointer ?
}
```


You may use the array index approach or pointer variable approach for processing the 2 array.



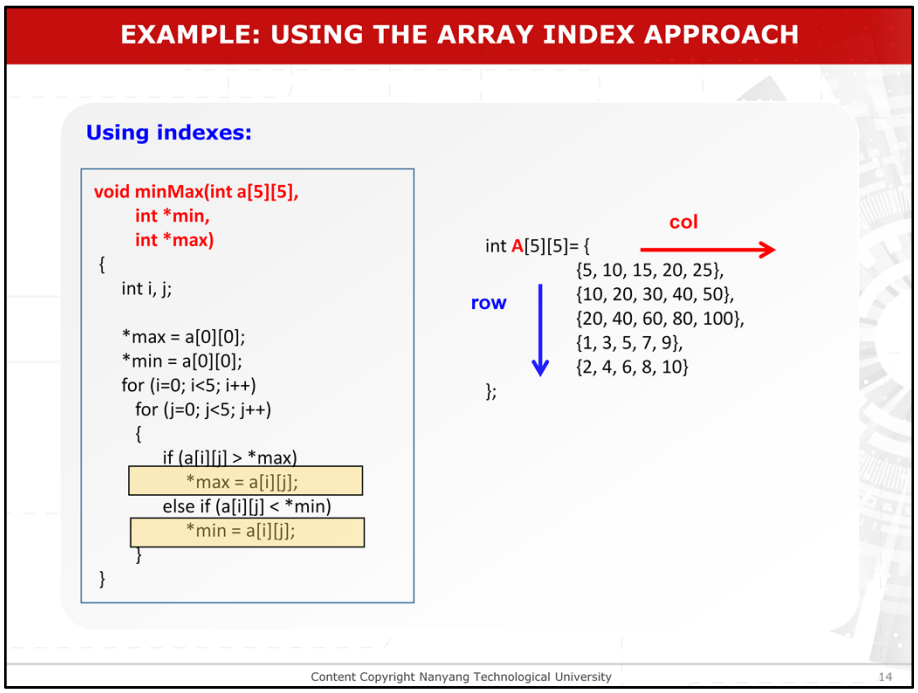
Example: Using the Array Index Approach

In this implementation using the array index approach, a nested **for** loop is used to process the 2 array in the function.

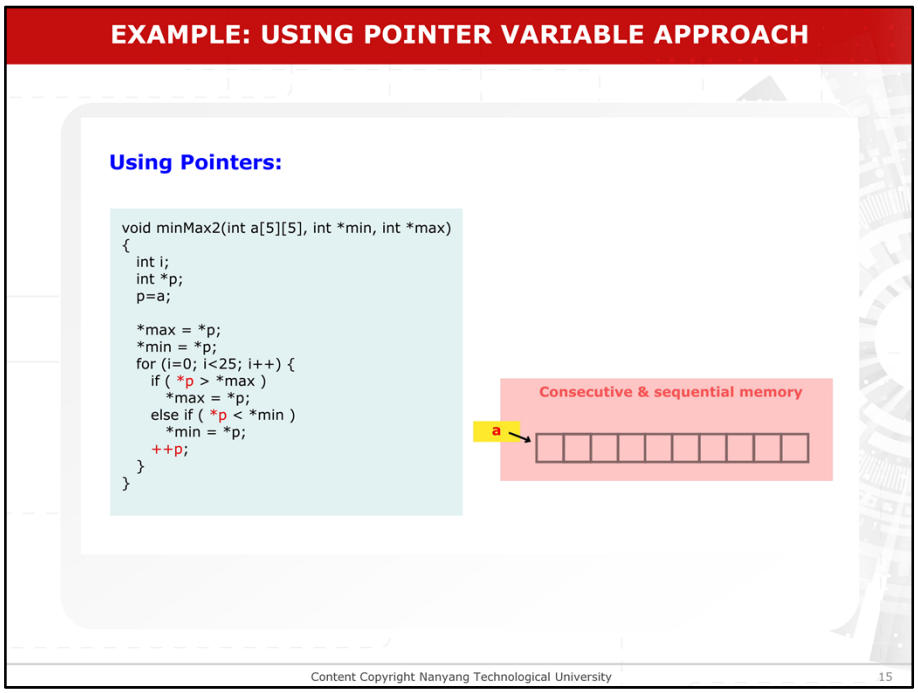


Example: Using the Array Index Approach

In this implementation using the array index approach, a nested **for** loop is used to process the 2 array in the function. It first initializes the **max** and **min** to contain the first array element number. The 2 array **eh** is processed using indexes to access and compare all the elements stored in the array with **max** and **min**.



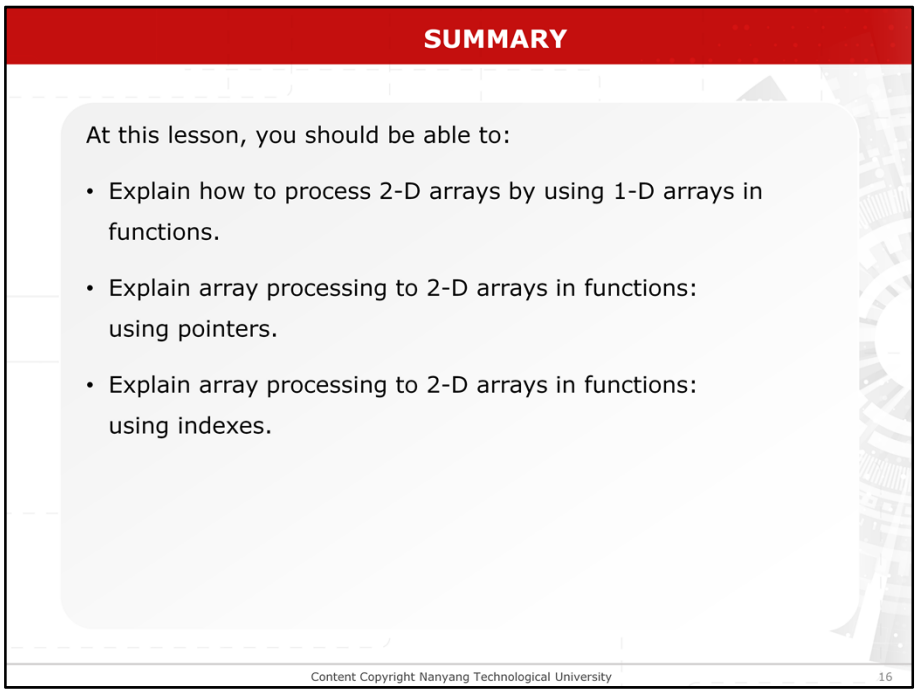
After the processing of the 2 array, the maximum and minimum numbers are determined and stored at **max** and **min** respectively. The implementation using indexes is quite straightforward.



Example: Using the Pointer Variable Approach

In this implementation, it uses the pointer variable approach. It first initializes the **max** and **min** to contain the first array element number. It is implemented using the pointer variable **p.** Then, assign the array **eh** to the pointer variable **p**, and initializes the values for **max** and **min** by the first element of the array:

A **for** loop is used to traverse and process the 2-D array by treating it as a 1-D array. In this approach, we use the **base address** as the reference. The index variable **i** is used to update the pointer variable to the corresponding array memory location, and retrieves the array element content via **the memory location**. Each array element content will be compared with the **max** and **min** to determine the maximum and minimum numbers respectively. At the end of the processing, the maximum and minimum numbers are determined and stored at **max** and **min** respectively. The values are returned to the calling function via call by reference.



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