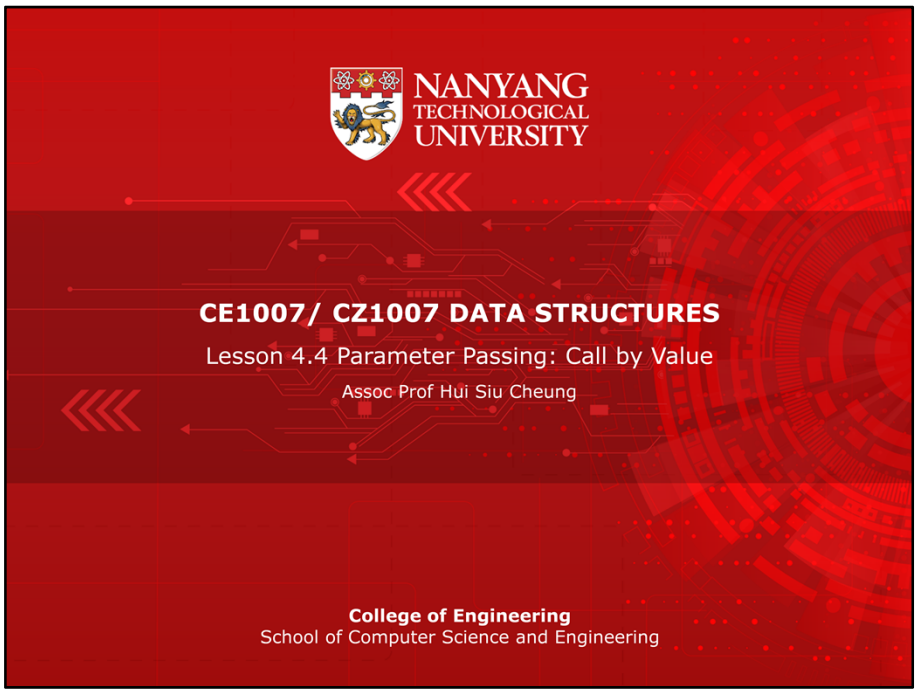
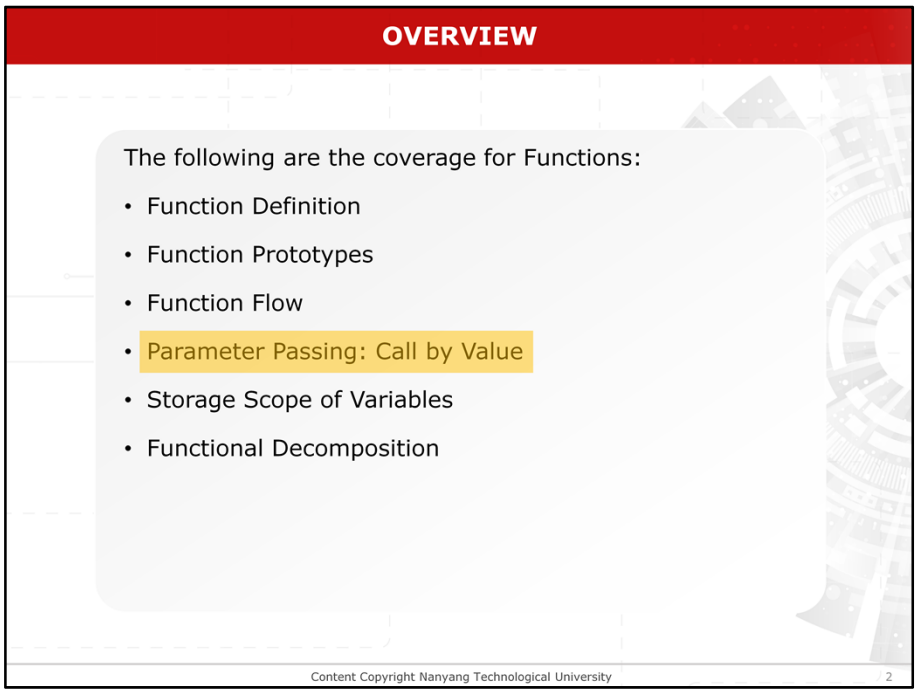
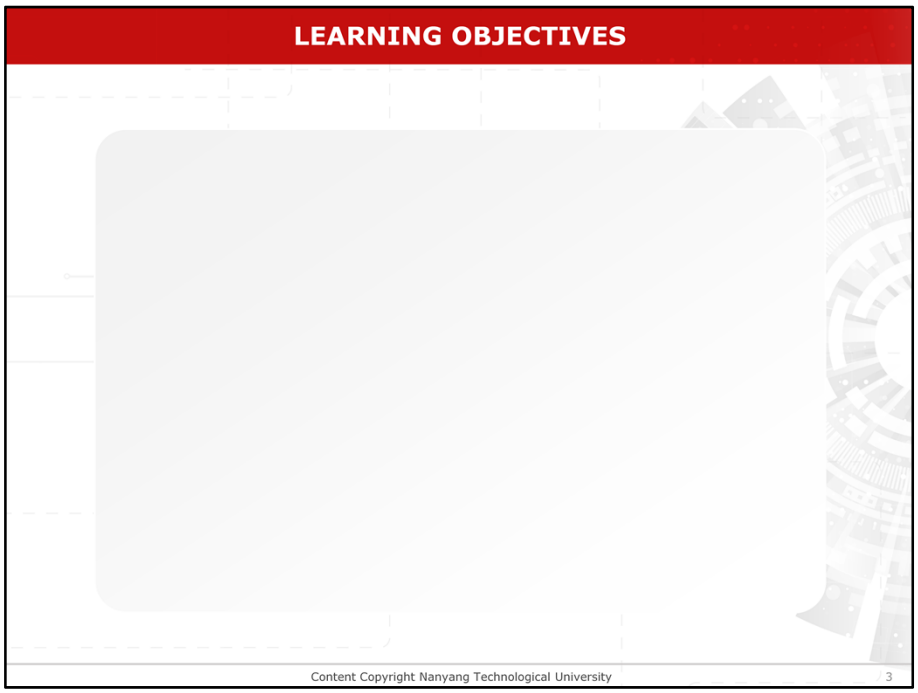


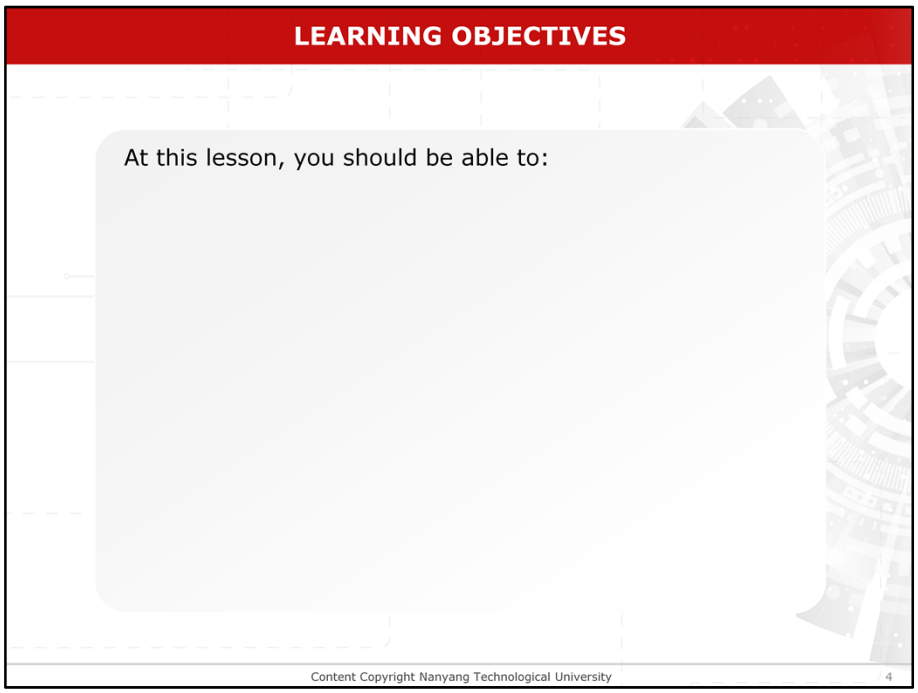
This lesson is on Functions.



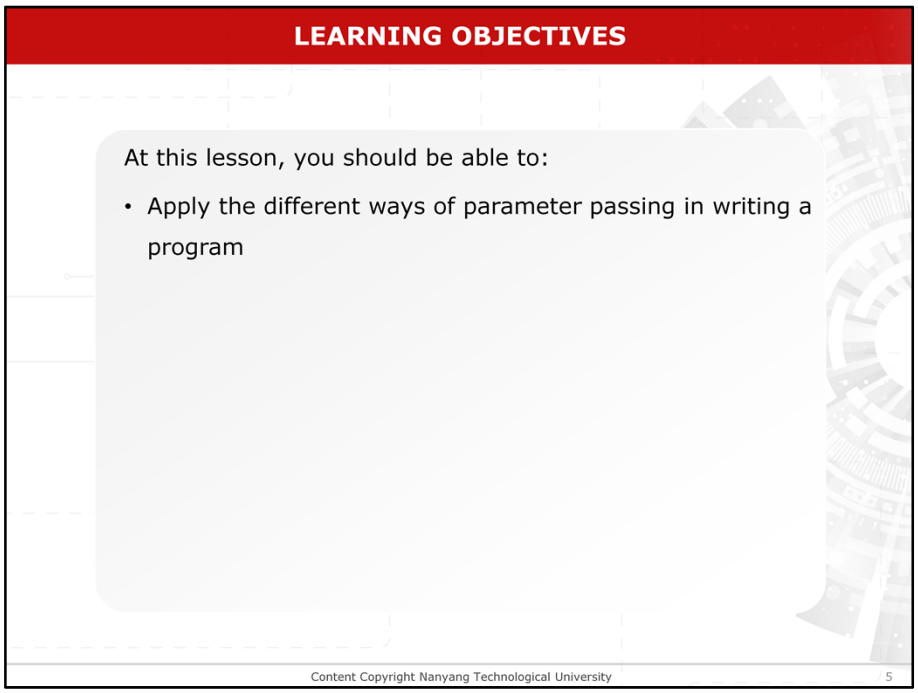
There are 6 main sections to cover for Functions. This video focused on the 4th topic: Parameter passing: Call by value



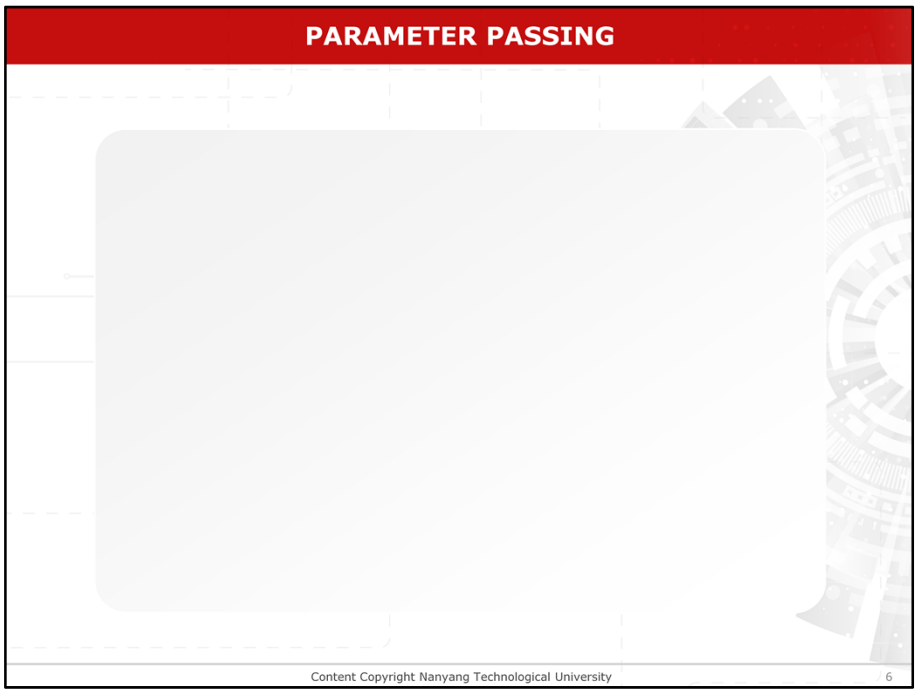
Learning objectives



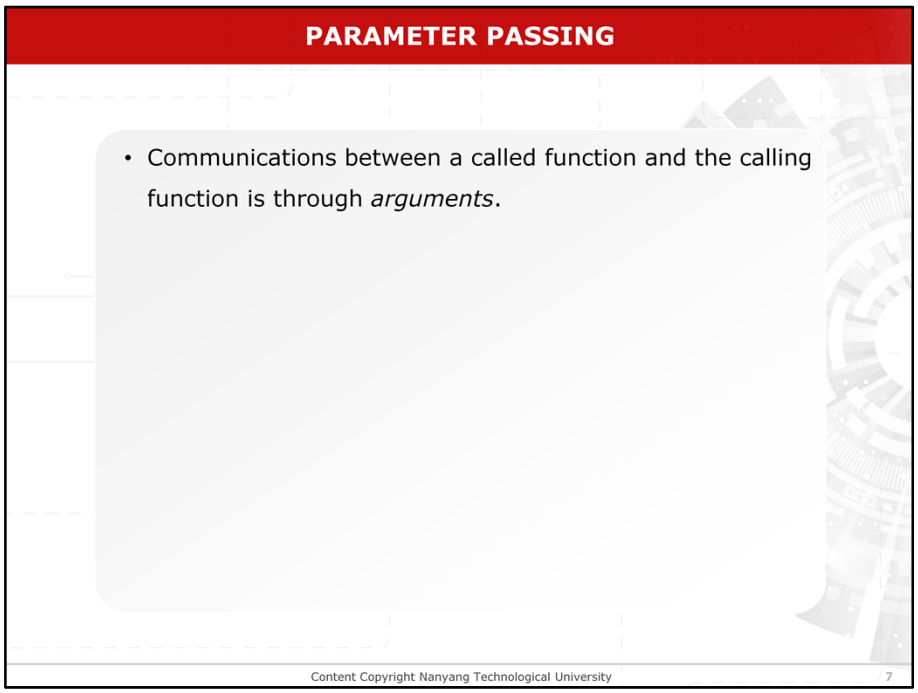
At this lesson, you should be able to:



- Apply the different ways of parameter passing in writing a program



Parameter Passing



Parameter Passing

Communications between a called function and the calling function is through *arguments*.

PARAMETER PASSING

- Communications between a called function and the calling function is through *arguments*.
- The called function then performs the task based on the received argument values.

The called function then performs the task based on the received argument values.

PARAMETER PASSING

- Communications between a called function and the calling function is through *arguments*.
- The called function then performs the task based on the received argument values.
- The called function can also return a value back to the calling function.

 9

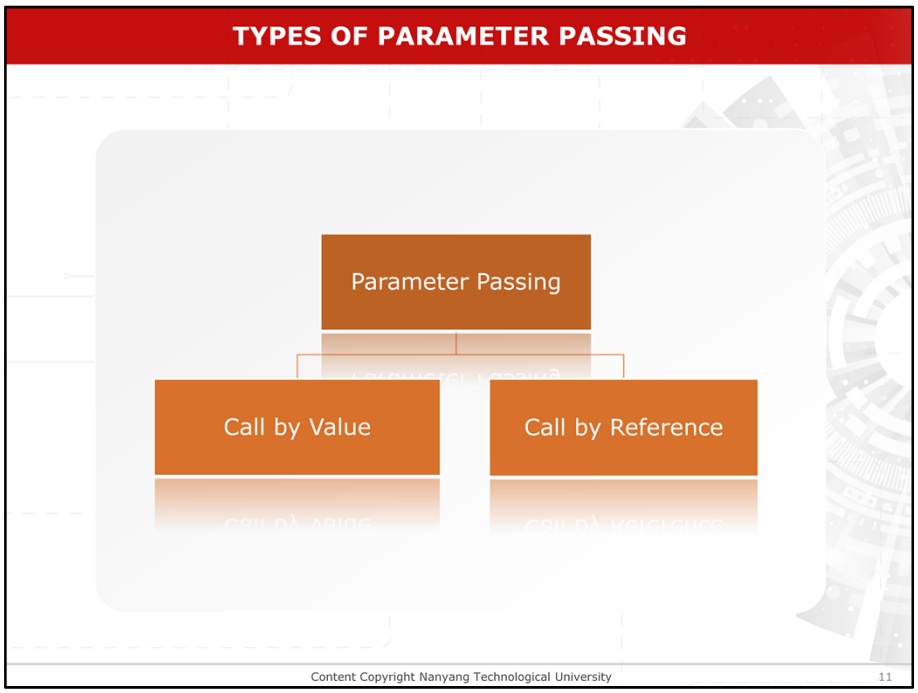
The called function can also return a value back to the calling function.

PARAMETER PASSING

- Communications between a called function and the calling function is through *arguments*.
- The called function then performs the task based on the received argument values.
- The called function can also return a value back to the calling function.
- The types and the number of arguments must match to those of the parameters in the function definition.

 10

The types and the number of arguments must match to those of the parameters in the function definition.



Parameter passing between functions may be performed in two ways: *call by value* and *call by reference*.

PARAMETER PASSING: CALL BY VALUE

- In call by value, the parameters must be declared in the function definition as regular variables.

In call by value, the parameters must be declared in the function definition as regular variables.

PARAMETER PASSING: CALL BY VALUE

- In call by value, the parameters must be declared in the function definition as regular variables.
- The arguments in function calls can be constants, variables or expressions.

 13

The arguments in function calls can be constants, variables or expressions.

PARAMETER PASSING: CALL BY VALUE

- In call by value, the parameters must be declared in the function definition as regular variables.
- The arguments in function calls can be constants, variables or expressions.
- When the function is called, the parameters hold a *copy* of the arguments locally.

When the function is called, the parameters hold a *copy* of the arguments locally.

PARAMETER PASSING: CALL BY VALUE

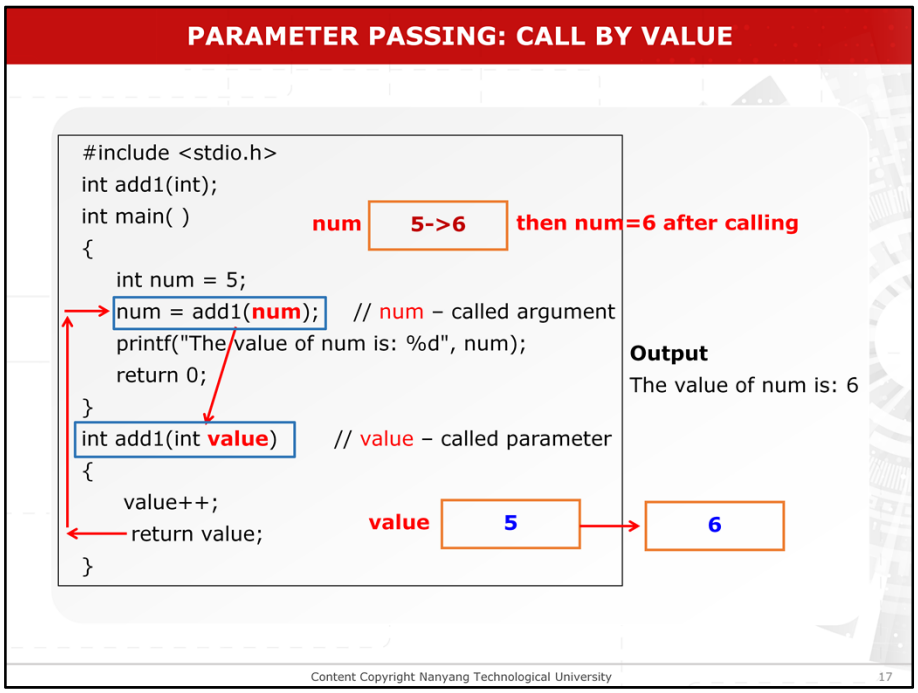
- In call by value, the parameters must be declared in the function definition as regular variables.
- The arguments in function calls can be constants, variables or expressions.
- When the function is called, the parameters hold a *copy* of the arguments locally.
- Therefore, any changes to the parameters in a function are done on the copy of the arguments.

 15

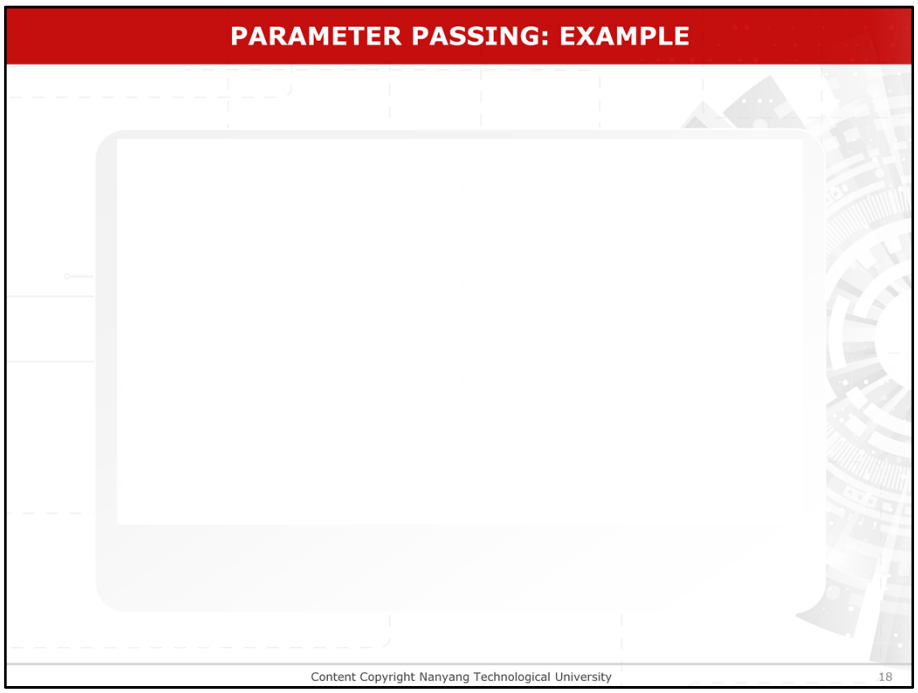
Therefore, any changes to the parameters in a function are done on the copy of the arguments.

PARAMETER PASSING: CALL BY VALUE

In any programs, there are two ways for a called function to return values back to the calling function. The first way is to use the **return** statement as shown in the function **add1()**. However, this can only be used when only **a single value** needs to be returned back from a function. If **two or more values** need to be passed back from a called function, we need to use another approach called call by reference via pointers.



In any programs, there are two ways for a called function to return values back to the calling function. The first way is to use the **return** statement as shown in the function **add1()**. However, this can only be used when only **a single value** needs to be returned back from a function. If **two or more values** need to be passed back from a called function, we need to use another approach called call by reference via pointers.



Parameter Passing: Example

In the program, it calls the function **distance()**. When the statement **dist = distance(2.0, 4.5);** is executed, it calls the function **distance()** in **main()**. Control is then transferred to the function **distance()**. Information is passed between the calling function and the called function through arguments. In this case, the function receives two arguments with values of 2.0 and 4.5. They are assigned to the corresponding parameters in the function definition. In addition, we can also use expression as an argument in the function as shown in the statement: **dist = distance(x*y, a*b)** When the execution of statements in the function body encounters the **return** statement, the control is then transferred back to the **main()** function, and the statement just after the function call in **main()**will continue to execute. The names for parameters need not be the same as function arguments. However, the number of arguments and the data type of the arguments must be the same as parameters defined in function definition. In the program, the arguments **2.0** and **4.5** correspond respectively to the parameters **x** and **y** in the first function call. Similarly, the arguments **x*y** and **a*b** in the **main()** function also correspond respectively to the parameters **x** and **y** in the second function call.

FUNCTION CALLING ANOTHER FUNCTION

```
#include  <stdio.h>
int max3(int, int, int);  /* function prototypes */
int max2(int, int);

int main()
{
     int x, y, z;
     printf("Input three integers => ");
     scanf("%d %d %d", &x, &y, &z);
     printf("Maximum of the 3 is %d\n", max3(x, y, z) );
     return 0;
}

int max3(int i, int j, int k)
{
     printf("Find the max in %d, %d and \%d\n", i, j, k);
     return max2(max2(i, j), max2(j,k));
}

int max2(int h, int k)
{
     printf("Find the max of %d and %d\n", h, k);
     return h > k ? h : k;
}
```

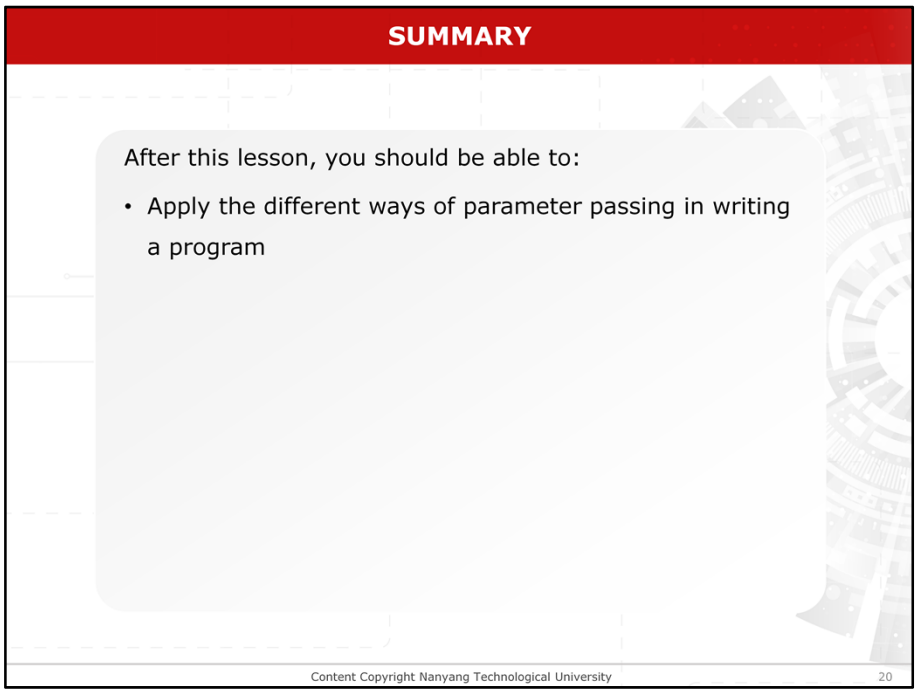
OUTPUT

 19

Function Calling Another Function

A function may be called by **main()** or another function through call by value. In the program, the function **max2()** specifies two parameters, **h** and **k**, of type **int**, and receives two function arguments from the calling function. The values of the arguments are then stored in the memory locations of the two parameters, **h** and **k**. The function then compares their values, and returns the larger value back to the calling function. The function **max3()** specifies three parameters, **i**, **j** and **k**, and receives the function arguments from the calling function, and compares their values to determine the largest value. The **max3()** function calls the **max2()** function to compare two values at a time and returns the maximum value:

Here, the function **max2()** is specified in the function **max2()** itself. The returned value from the called function **max2()** will be used again as arguments in the same function **max2()**. The maximum value is then returned back to the calling function.



In summary, after viewing this video lesson, your should be able to do the listed.