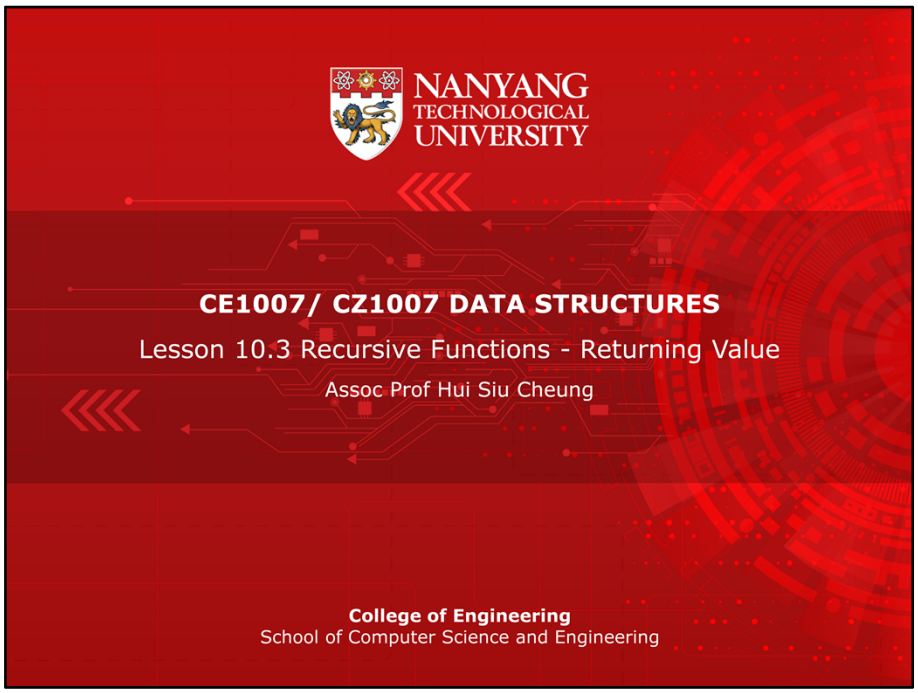
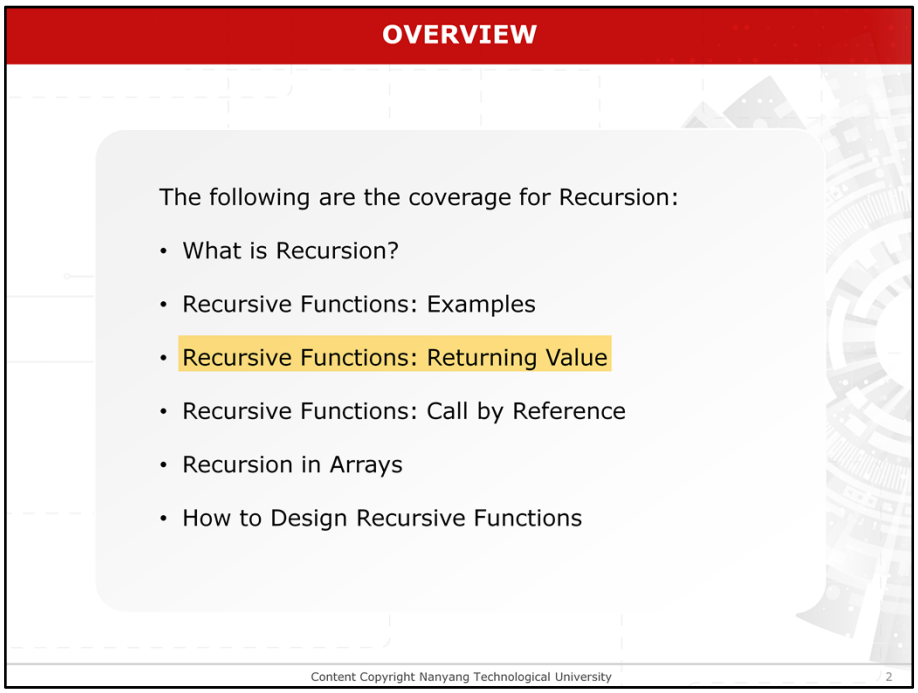
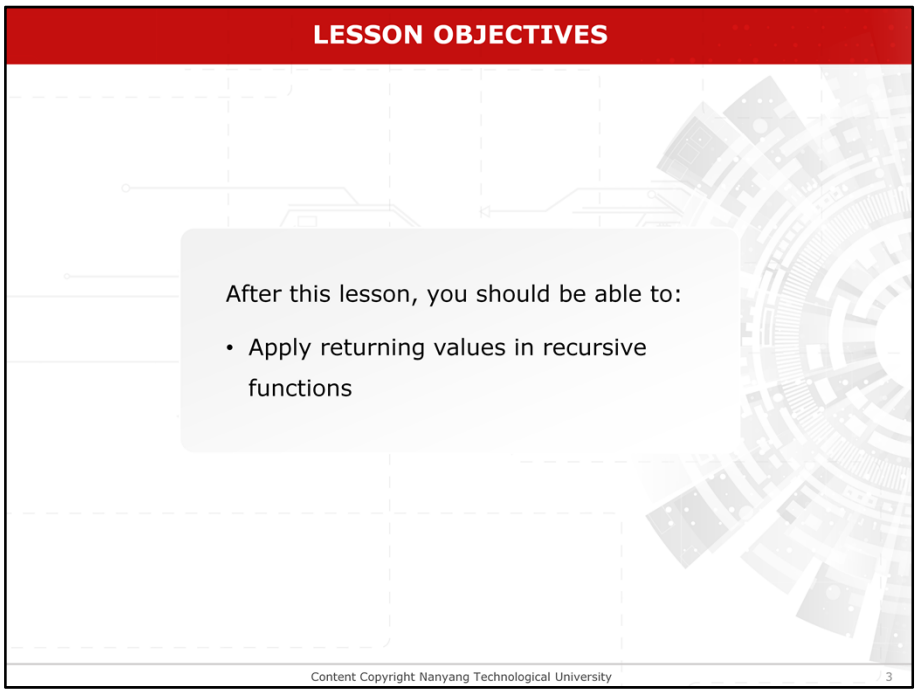
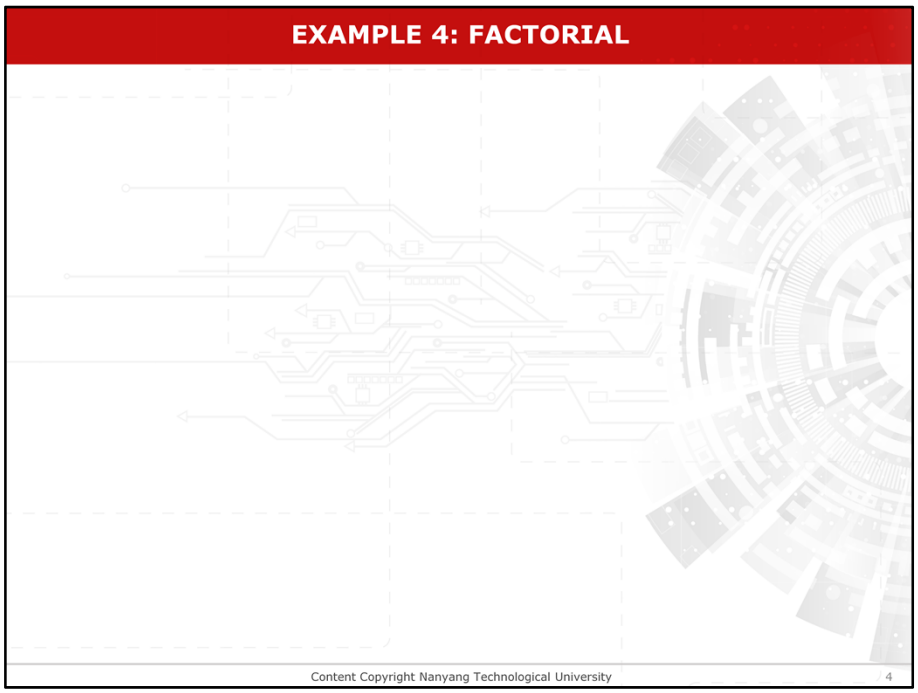

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 10.3 Recursive Functions - Returning Value
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering



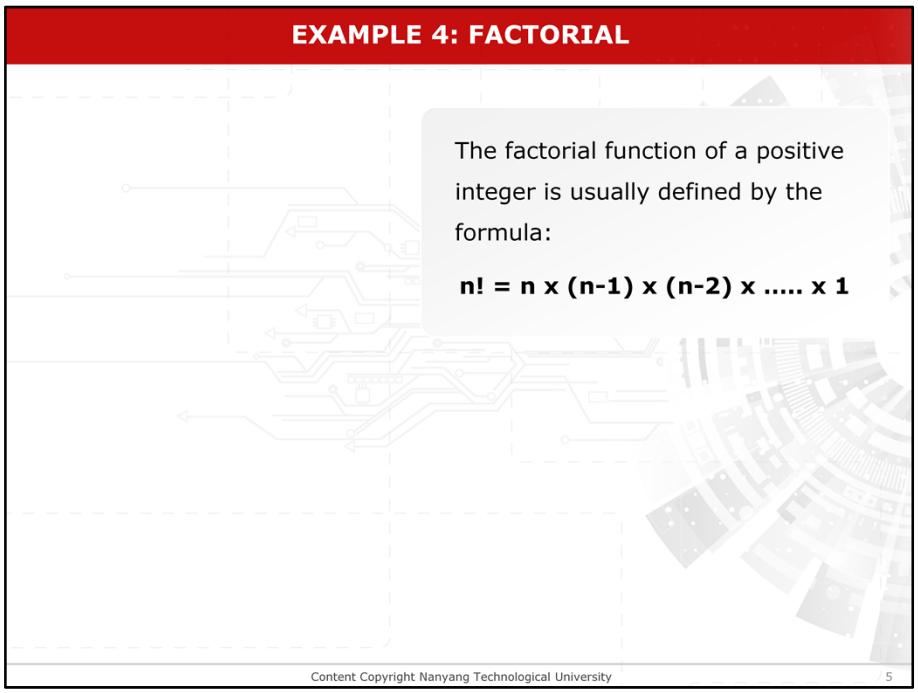
There are 6 main sections to cover for Recursion as shown. This video lesson focuses on the third part where we will look into returning values in recursive functions.



After this lesson, you should be able to apply returning values in recursive functions.



Example 4: Factorial



Consider the problem of finding the factorial of a number **n** where the factorial function of a positive integer is usually defined by the formula

n factorial = n multiply by (n minus 1) multiply by (n minus 2) and so on till 1

EXAMPLE 4: FACTORIAL

```
#include  <stdio.h>
int fact(int n);
int main()
{
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    printf("n! = %d\n", fact(num));
    return 0;
}
int fact(int n)
{
    int i;
    int temp = 1;
    for (i = n; i > 0; i--)
        temp *= i;
    return temp;
}
```

The factorial function of a positive integer is usually defined by the formula:

n! = n x (n-1) x (n-2) x ..... x 1

Non-recursive approach: use for loop

The program shown implements the factorial function using non-recursive approach with a **for** loop iteratively.

EXAMPLE 4: FACTORIAL

```
#include  <stdio.h>
int fact(int n);
int main()
{
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    printf("n! = %d\n", fact(num));
    return 0;
}
int fact(int n)
{
    int i;
    int temp = 1;
    for (i = n; i > 0; i--)
        temp *= i;
    return temp;
}
```

The factorial function of a positive integer is usually defined by the formula:

n! = n x (n-1) x (n-2) x ..... x 1

Non-recursive approach: use for loop

The function **fact()** receives the argument from the calling function. In the **for** loop, the factorial of the input argument is calculated. The result is then passed to the calling function.

EXAMPLE 4: FACTORIAL

```
#include  <stdio.h>
int fact(int n);
int main()
{
     int num;
     printf("Enter an integer: ");
     scanf("%d", &num);
     printf("n! = %d\n", fact(num));
     return 0;
}
int fact(int n)
{
     int i;
     int temp = 1;
     for (i = n; i > 0; i--)
          temp *= i;
     return temp;
}
```

The factorial function of a positive integer is usually defined by the formula:

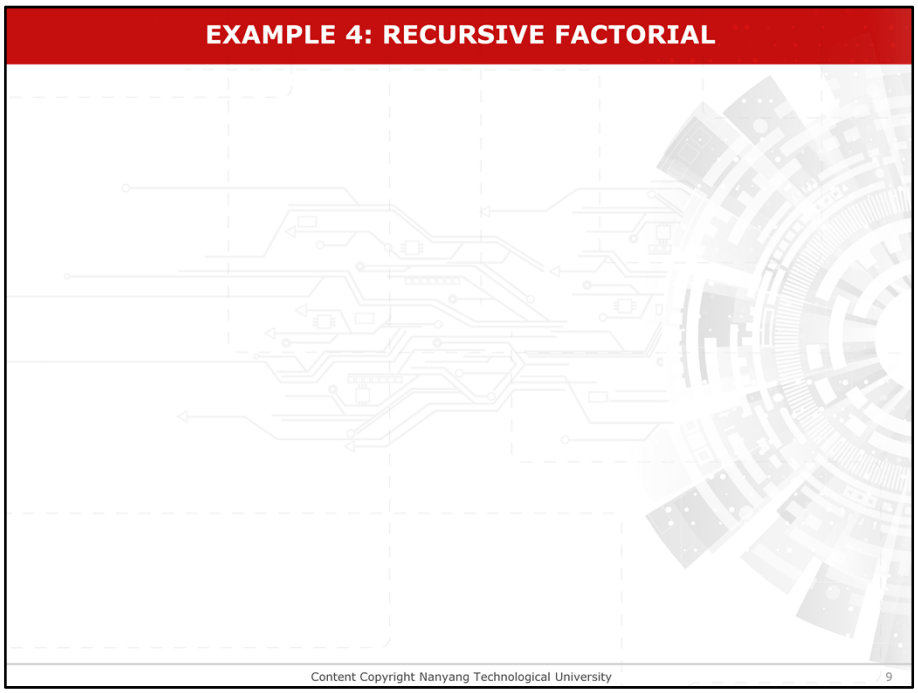
n! = n x (n-1) x (n-2) x ..... x 1

Non-recursive approach: use for loop

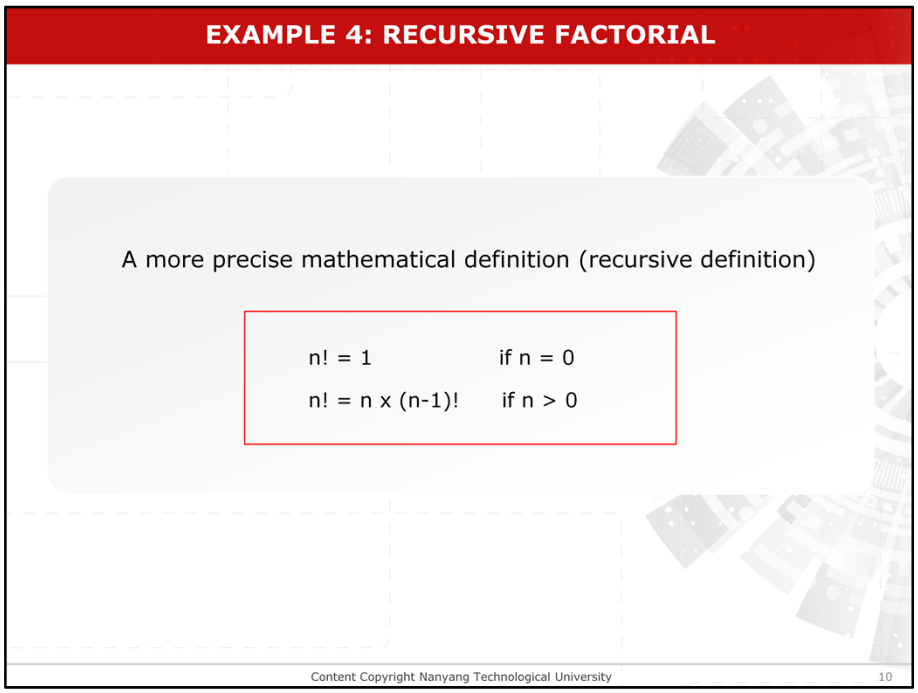
Output Enter an integer: 4

n! = 24

For **fact(4)**, it will return the value of 24 to the calling **main()** function.
It is quite straightforward to implement the factorial function using non-recursive approach.



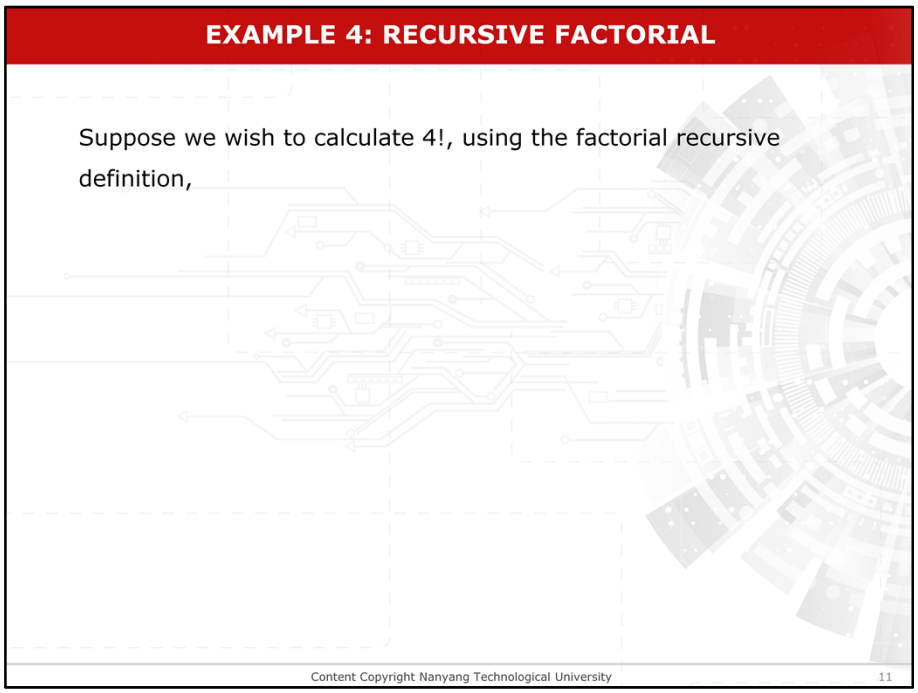
It is possible to rewrite the same factorial function using recursive approach.
Let's see how the recursive definition can be written in the recursive function factorial().



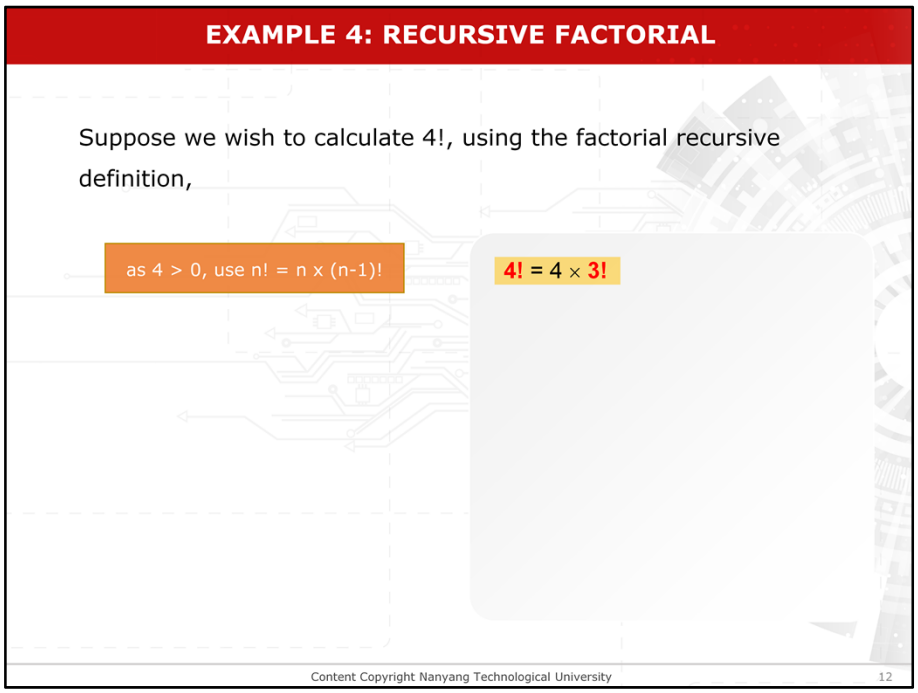
A more precise mathematical definition (recursive definition) is that:

If n = 0, n factorial is equal to 1.

If n greater than 0, n factorial is equal to n multiply by (n minus 1) factorial.

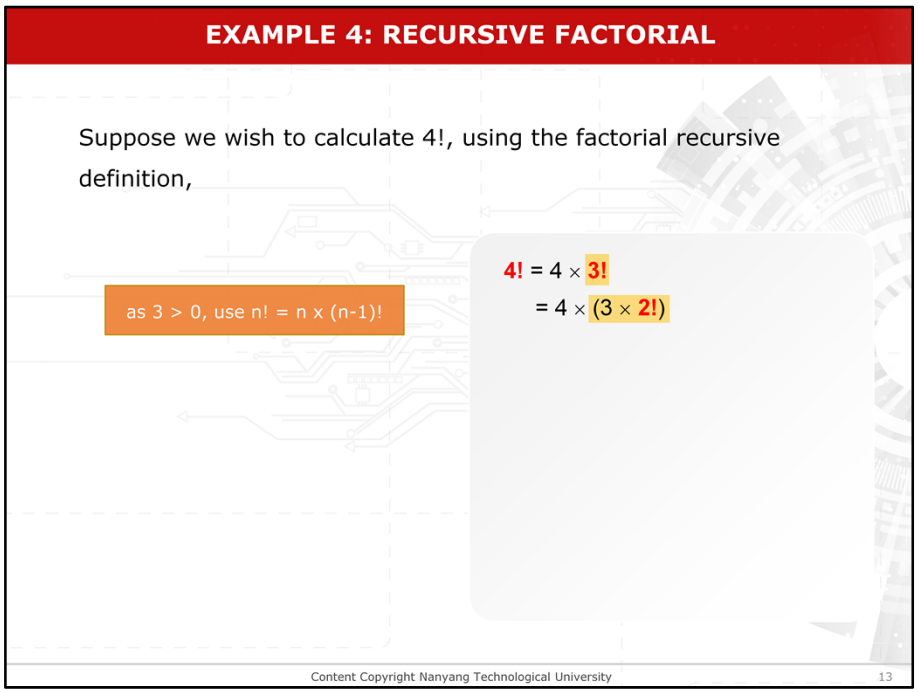


Suppose we wish to calculate 4 factorial, using the factorial recursive definition,



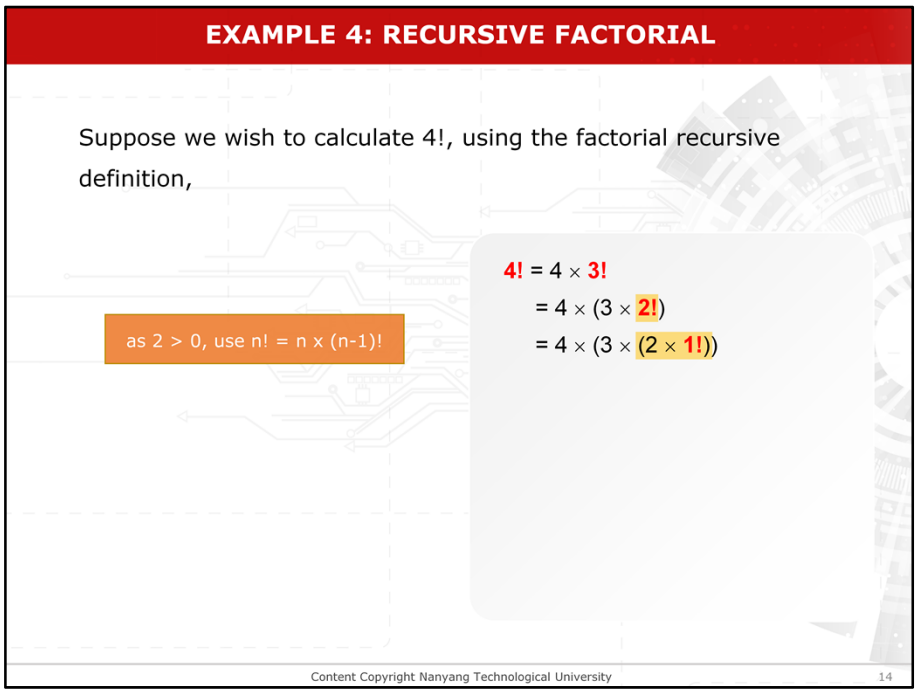
as 4 > 0, we use n factorial = n multiply by (n minus 1) factorial so

4 factorial is equal to 4 mutliply by 3 factorial



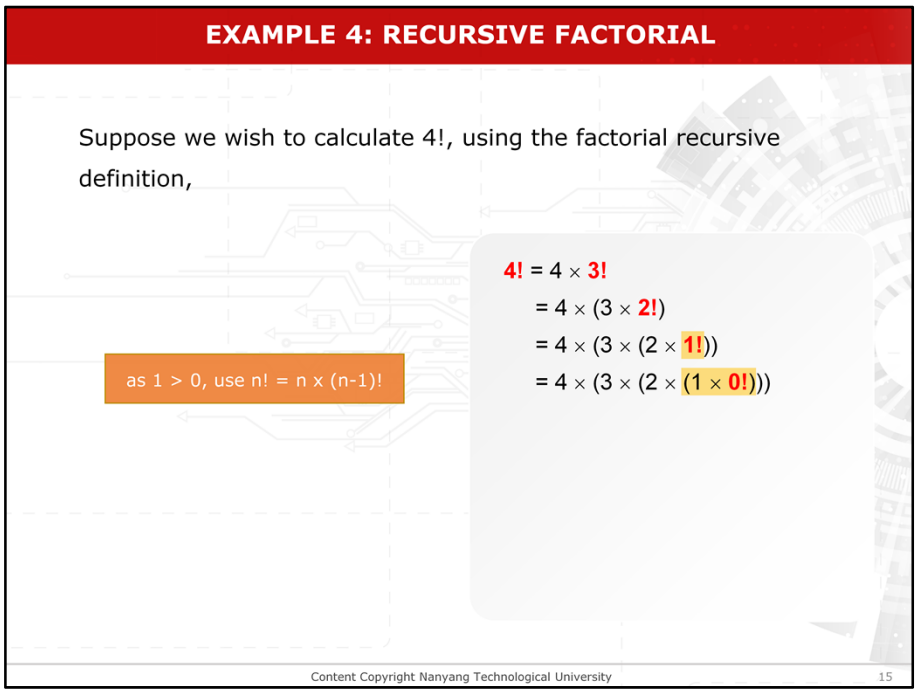
as 3 > 0, we use again n factorial = n multiply by (n minus 1) factorial so

3 factorial is equal to 3 mutliply by 2 factorial



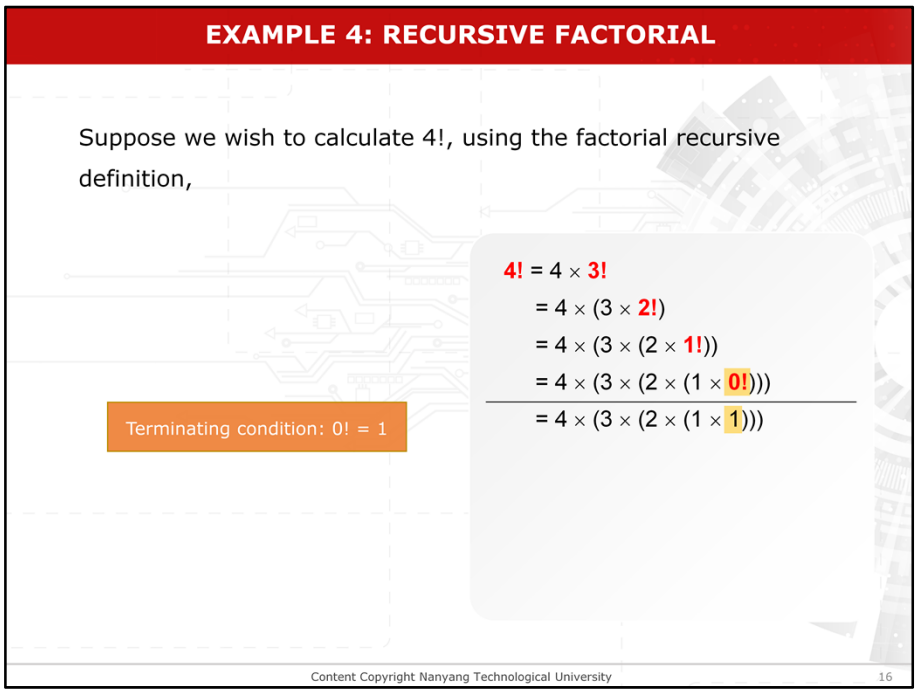
as 2 > 0, we use again n factorial = n multiply by (n minus 1) factorial so

2 factorial is equal to 2 mutliply by 1 factorial



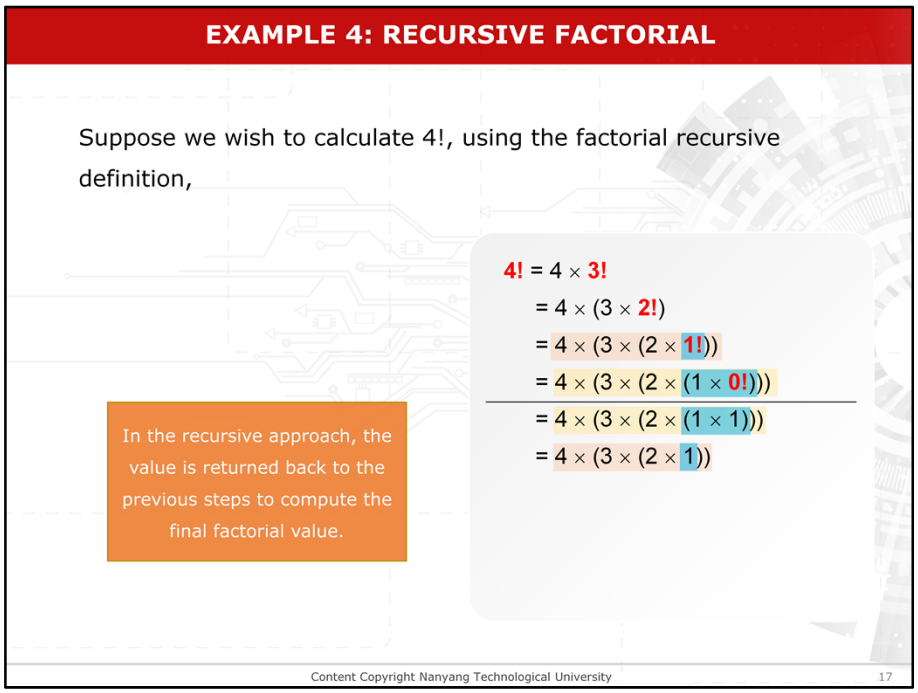
as 1 > 0, we use again n factorial = n multiply by (n minus 1) factorial so

1 factorial is equal to 1 mutliply by 0 factorial

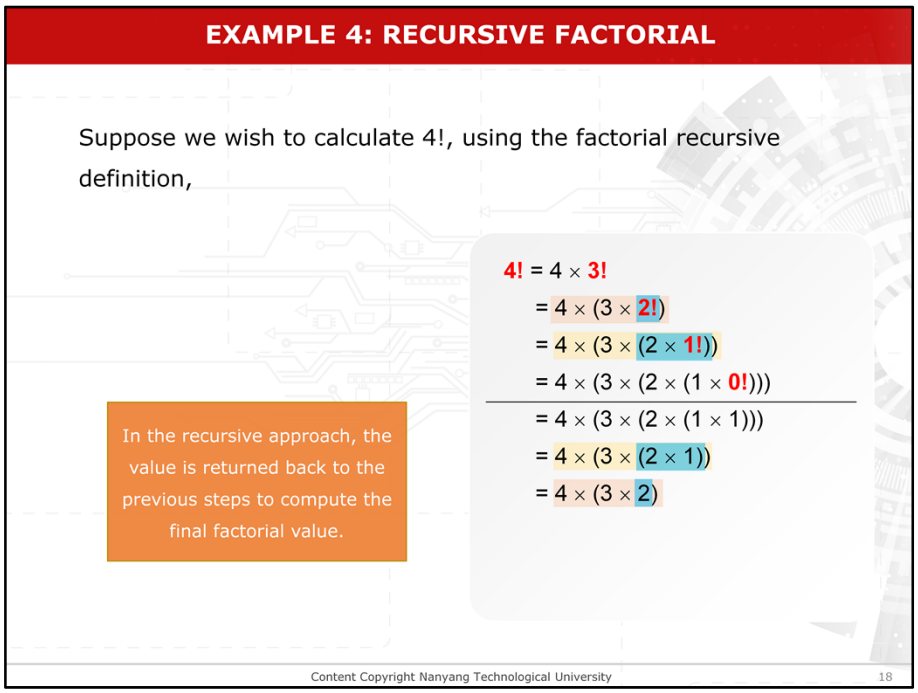


0 factorial is defined as 1.

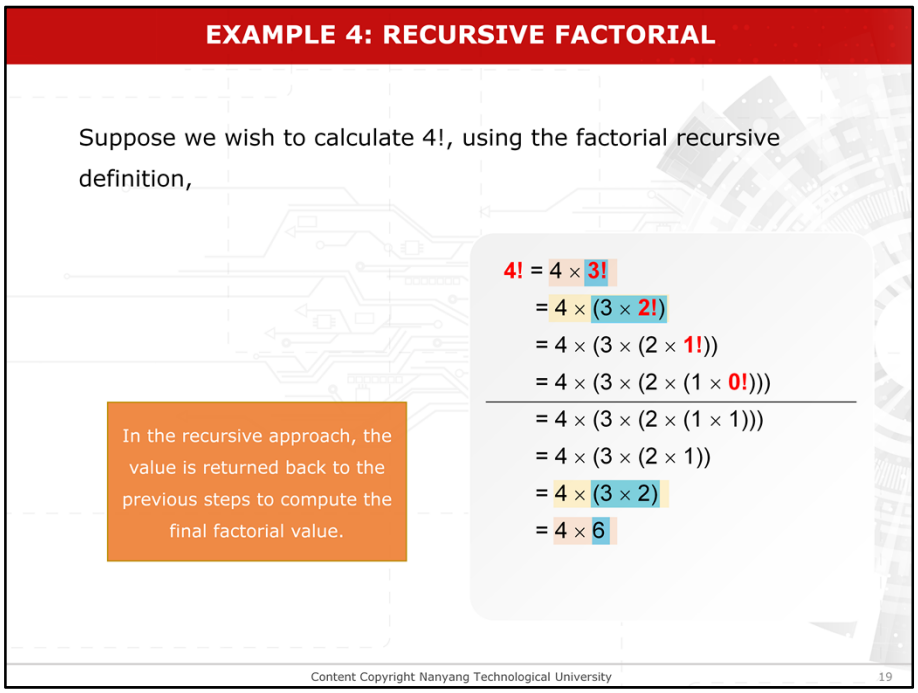
This is the terminating condition.



In the recursive approach, the value is returned back to the previous steps to compute the final factorial value.


EXAMPLE 4: RECURSIVE FACTORIAL
Suppose we wish to calculate 4!, using the factorial recursive definition,
4! = 4 × 3!
= 4 × (3 × 2!)
= 4 × (3 × (2 × 1!))
= 4 × (3 × (2 × (1 × 0!)))
= 4 × (3 × (2 × (1 × 1)))
= 4 × (3 × (2 × 1))
= 4 × (3 × 2)
In the recursive approach, the value is returned back to the previous steps to compute the final factorial value.
Content Copyright Nanyang Technological University
18

[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
Suppose we wish to calculate 4!, using the factorial recursive definition,
4! = 4 × 3!
= 4 × (3 × 2!)
= 4 × (3 × (2 × 1!))
= 4 × (3 × (2 × (1 × 0!)))
= 4 × (3 × (2 × (1 × 1)))
= 4 × (3 × (2 × 1))
= 4 × (3 × 2)
= 4 × 6
In the recursive approach, the value is returned back to the previous steps to compute the final factorial value.
Content Copyright Nanyang Technological University
19

[no audio]

EXAMPLE 4: RECURSIVE FACTORIAL

Suppose we wish to calculate 4!, using the factorial recursive definition,

$4! = 4 \times 3!$
$= 4 \times (3 \times 2!)$
$= 4 \times (3 \times (2 \times 1!))$
$= 4 \times (3 \times (2 \times (1 \times 0!)))$

$= 4 \times (3 \times (2 \times (1 \times 1)))$
$= 4 \times (3 \times (2 \times 1))$
$= 4 \times (3 \times 2)$
$= 4 \times 6$
$= 24$

In the recursive approach, the value is returned back to the previous steps to compute the final factorial value.

So the final factorial value for 4 factorial is 24.

EXAMPLE 4: RECURSIVE FACTORIAL

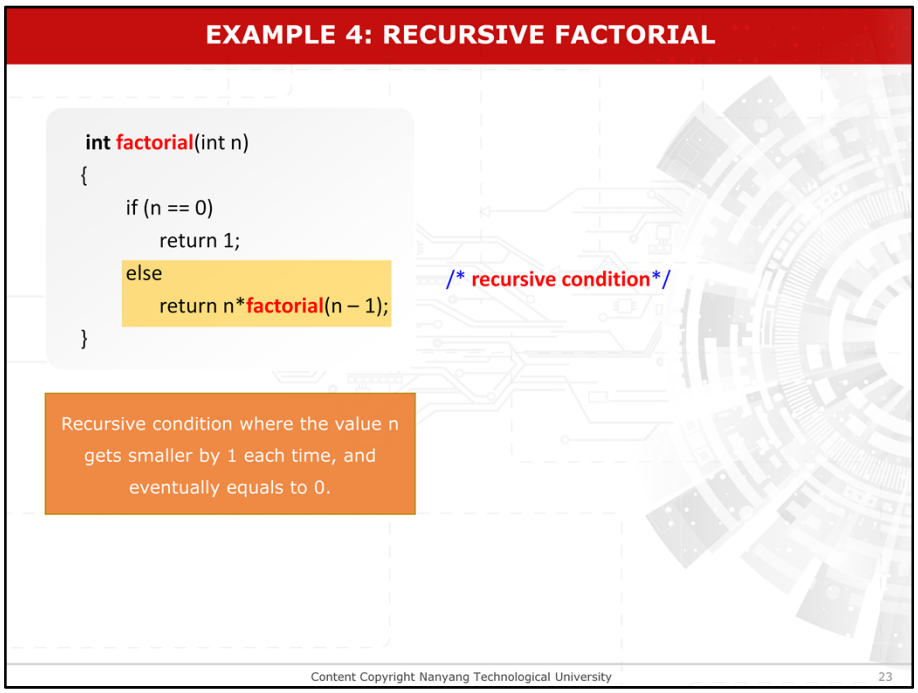
```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n*factorial(n – 1);
}
```


The recursive factorial function can be implemented as shown.

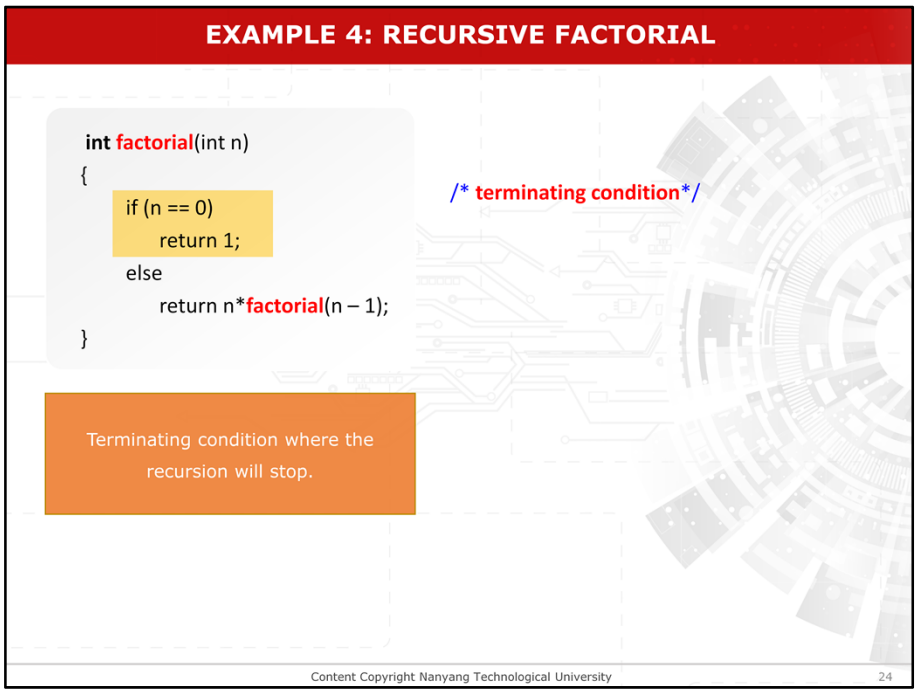
EXAMPLE 4: RECURSIVE FACTORIAL

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n*factorial(n – 1);
}
```


There are two **return** statements in the function.

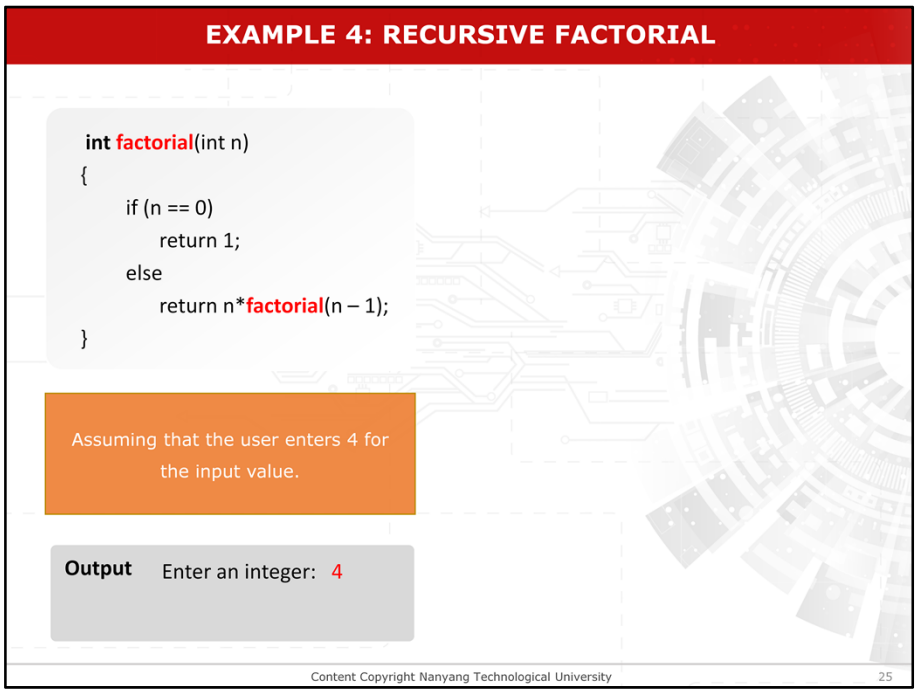


Recursive condition: For each new call to **factorial()** for recursive condition, the value **n** gets smaller by 1 each time, and eventually equals to 0.

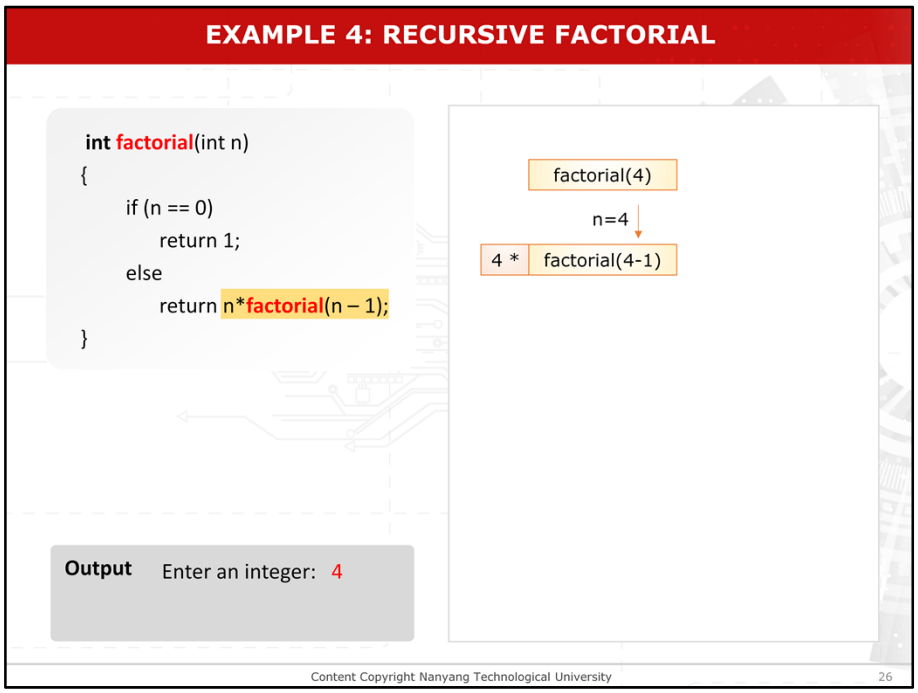


Terminating condition: The first **return 1** statement does not call the function **factorial()**. This is the terminating condition, where the recursion will stop.

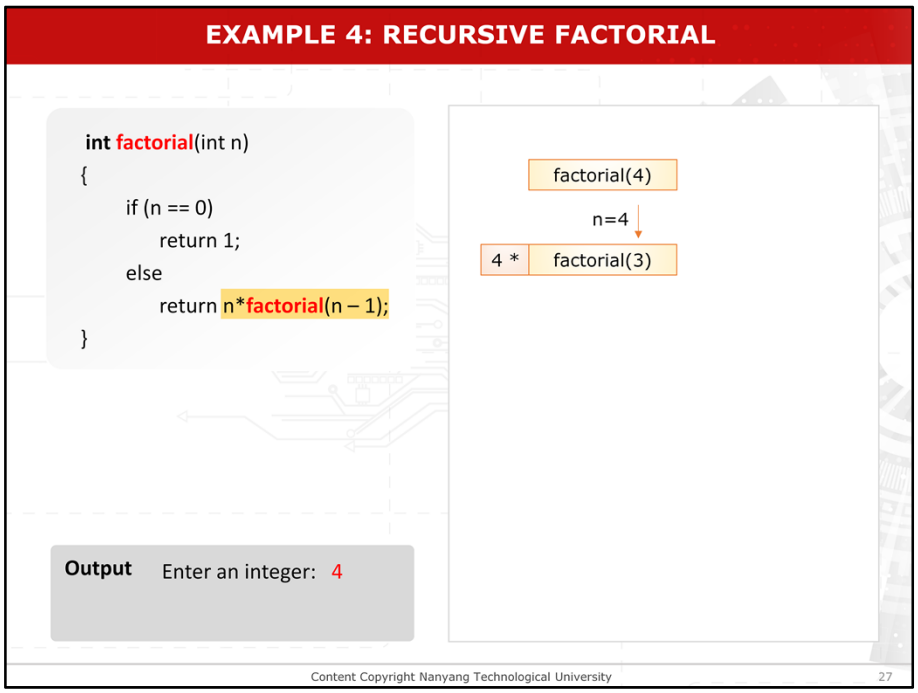
When **n** equals to 0, that is **0 factorial**, the terminating condition is reached, and the function returns the value 1.



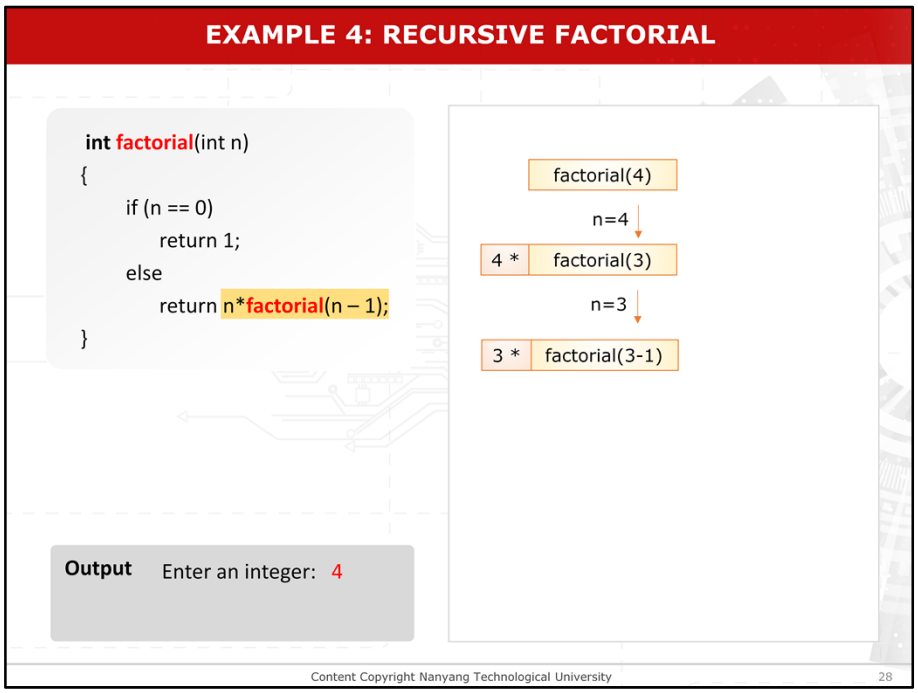
To evaluate the function **factorial()**, we assume that the user enters 4 for the input value.



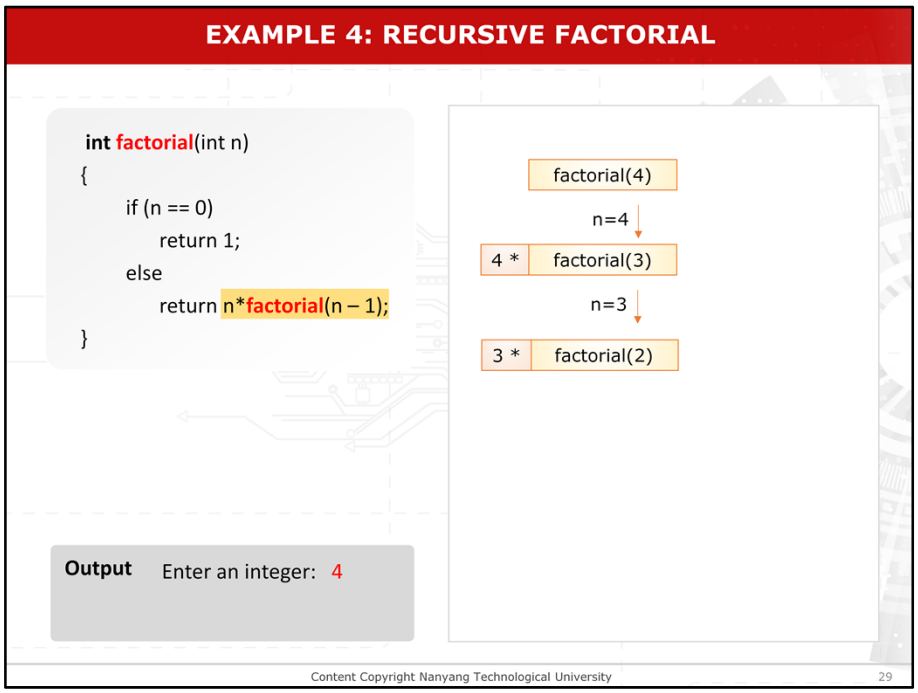
The program tracing for the function call **factorial(4)** is illustrated. The values associated with the parameter **n** for each level of recursion will be shown.


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
factorial(4)
n=4
4 *
factorial(3)
Output
Enter an integer: 4
Content Copyright Nanyang Technological University
27

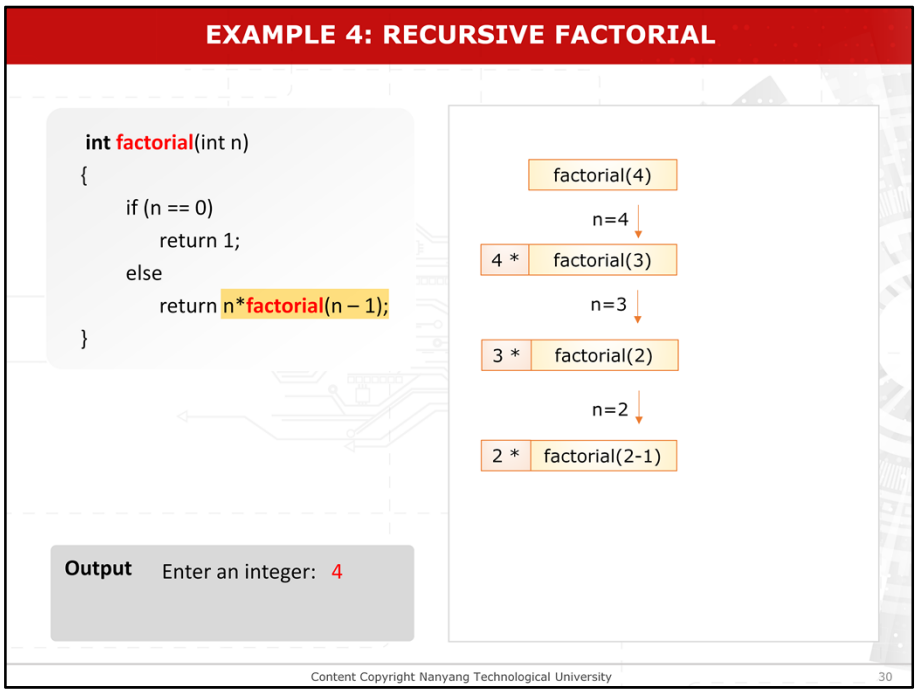
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(3-1)
Output Enter an integer: 4
Content Copyright Nanyang Technological University
28

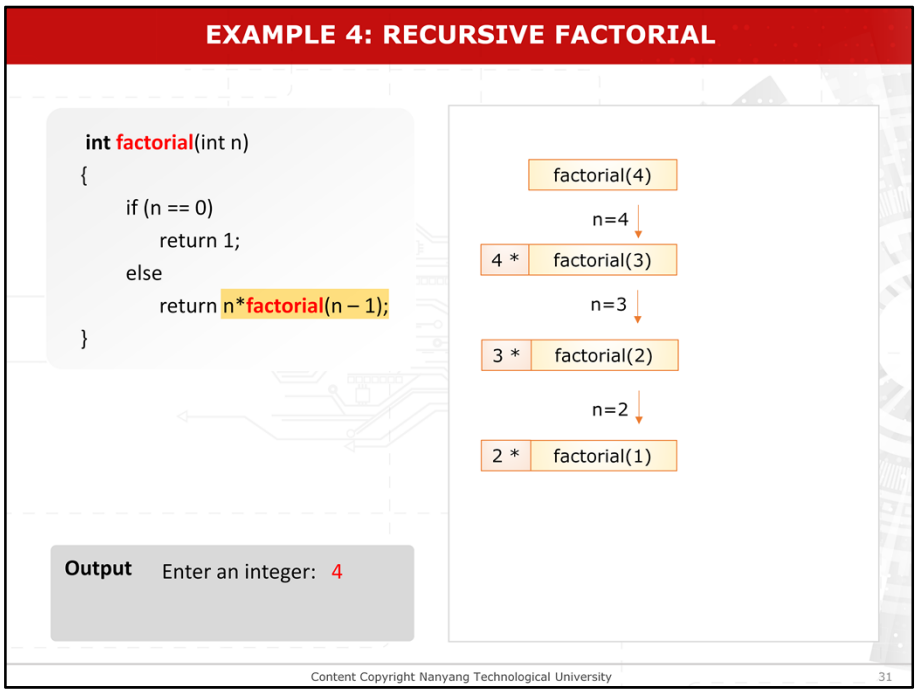
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
Output Enter an integer: 4
Content Copyright Nanyang Technological University
29

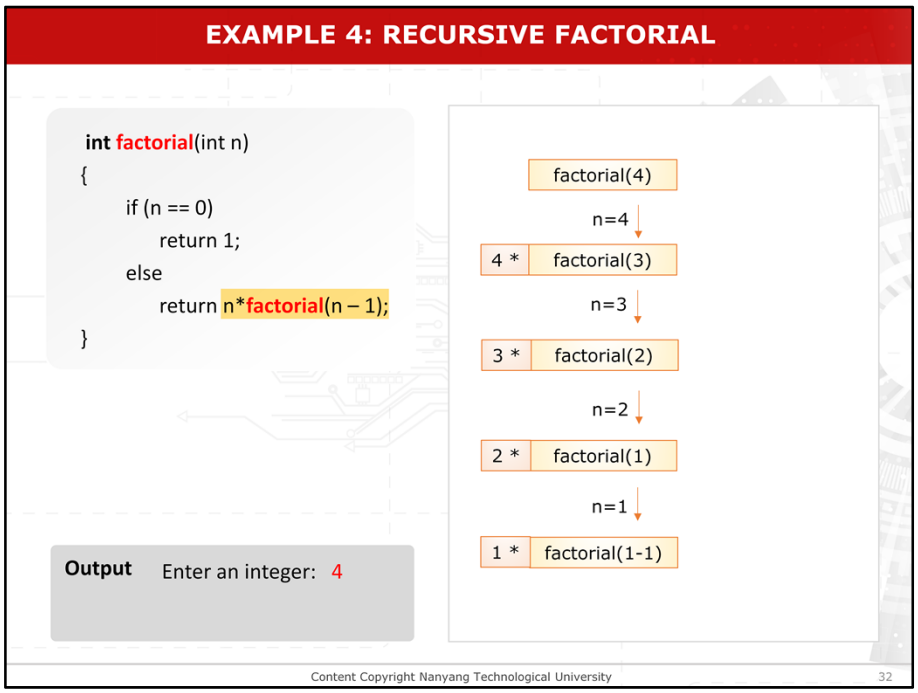
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
n=2
2 * factorial(2-1)
Output Enter an integer: 4
Content Copyright Nanyang Technological University
30

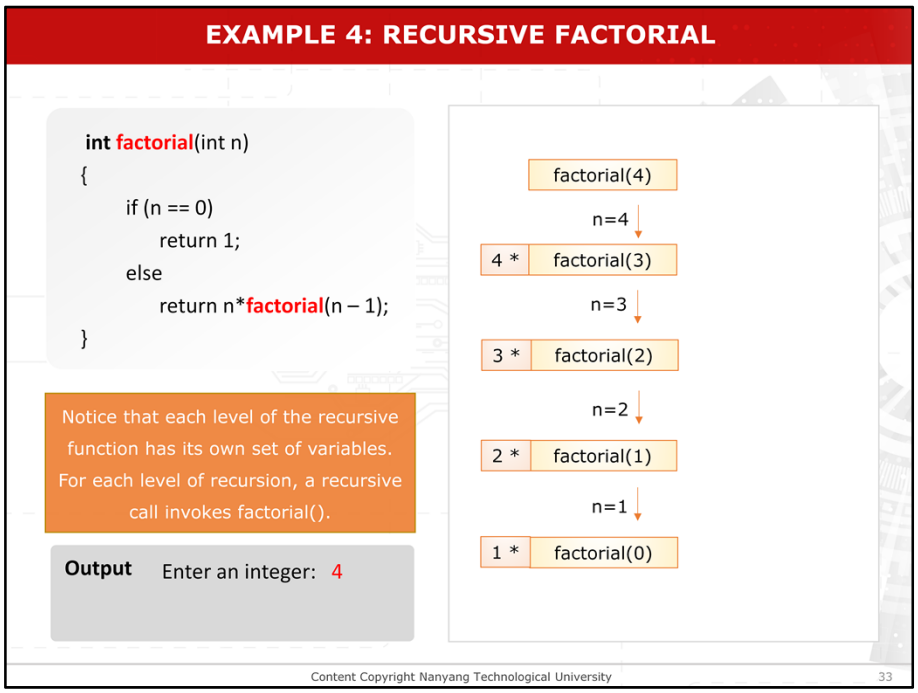
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
n=2
2 * factorial(1)
Output Enter an integer: 4
Content Copyright Nanyang Technological University
31

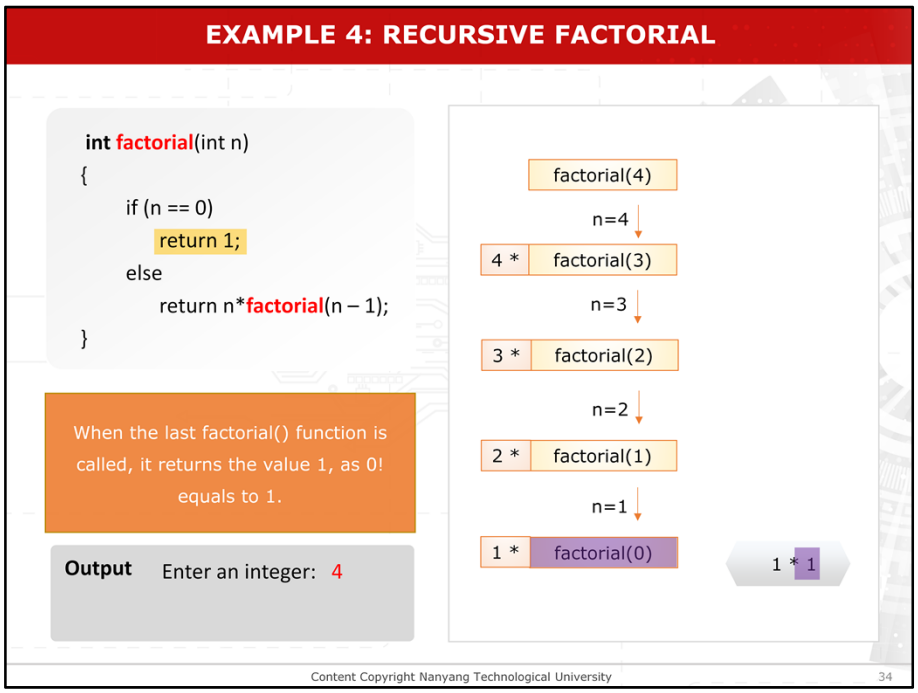
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
Output
Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
n=2
2 * factorial(1)
n=1
1 * factorial(1-1)
Content Copyright Nanyang Technological University
32

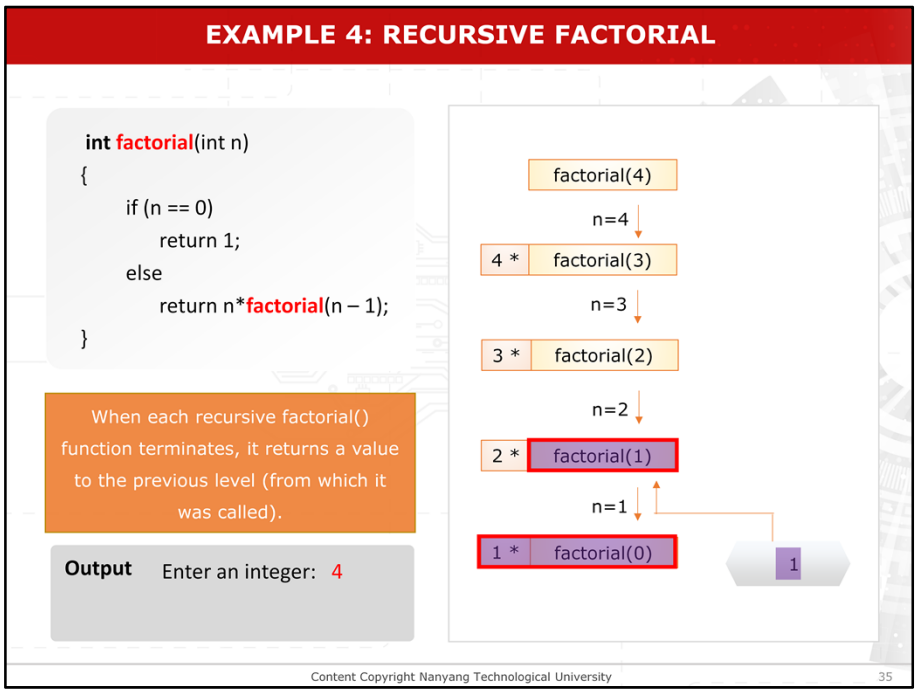
[no audio]



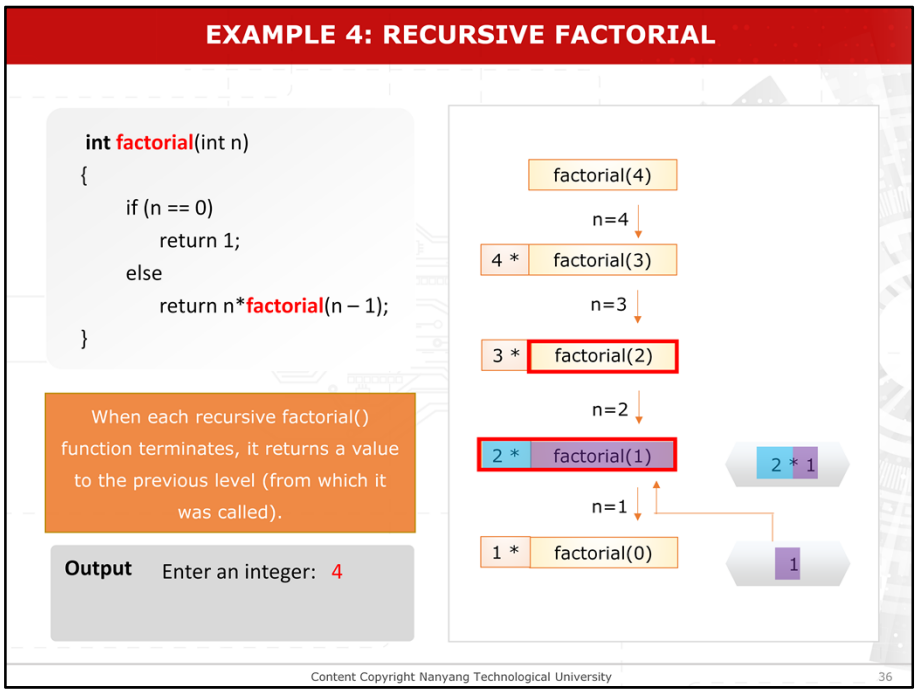
Notice that each level of the recursive function has its own set of variables. For each level of recursion, a recursive call invokes **factorial()**.



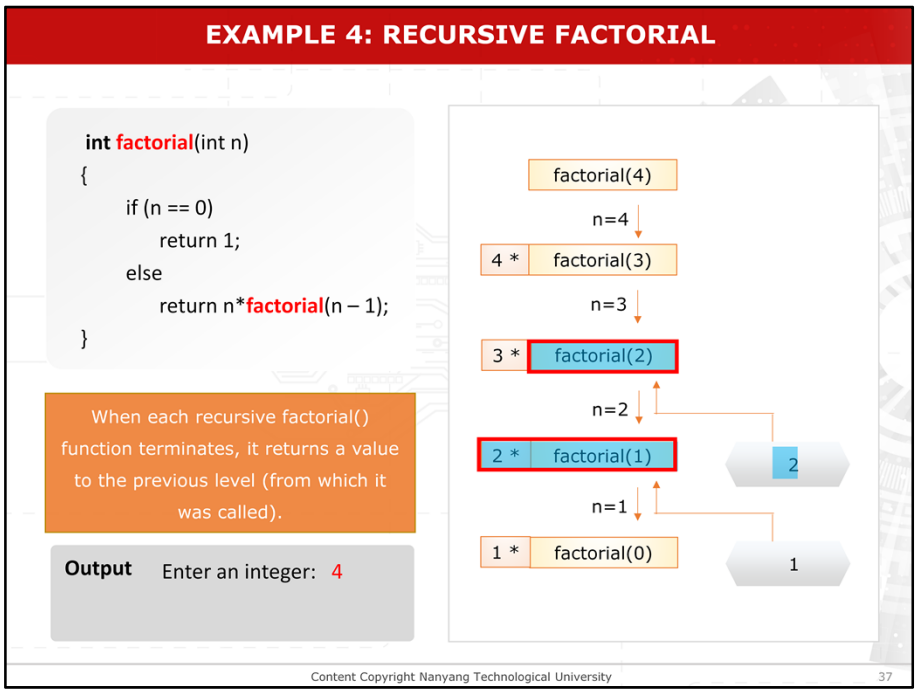
When the last **factorial()** function is called, it returns the value 1, as **0 factorial** equals to 1.



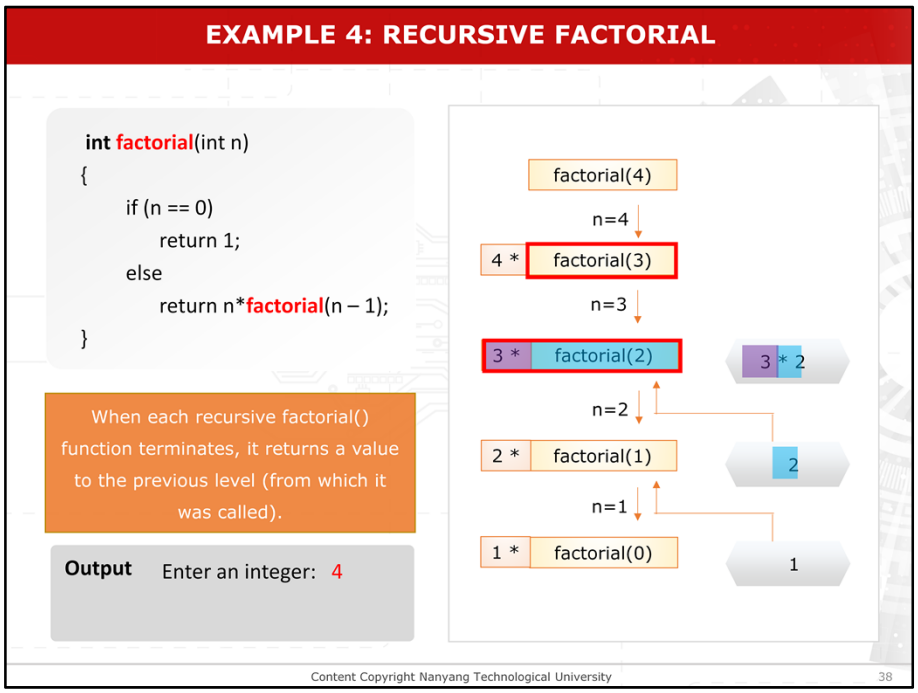
When each recursive **factorial()** function terminates, it returns a value to the previous level (from which it was called).


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
When each recursive factorial() function terminates, it returns a value to the previous level (from which it was called).
Output Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
n=2
2 * factorial(1)
n=1
1 * factorial(0)
2 * 1
1
Content Copyright Nanyang Technological University
36

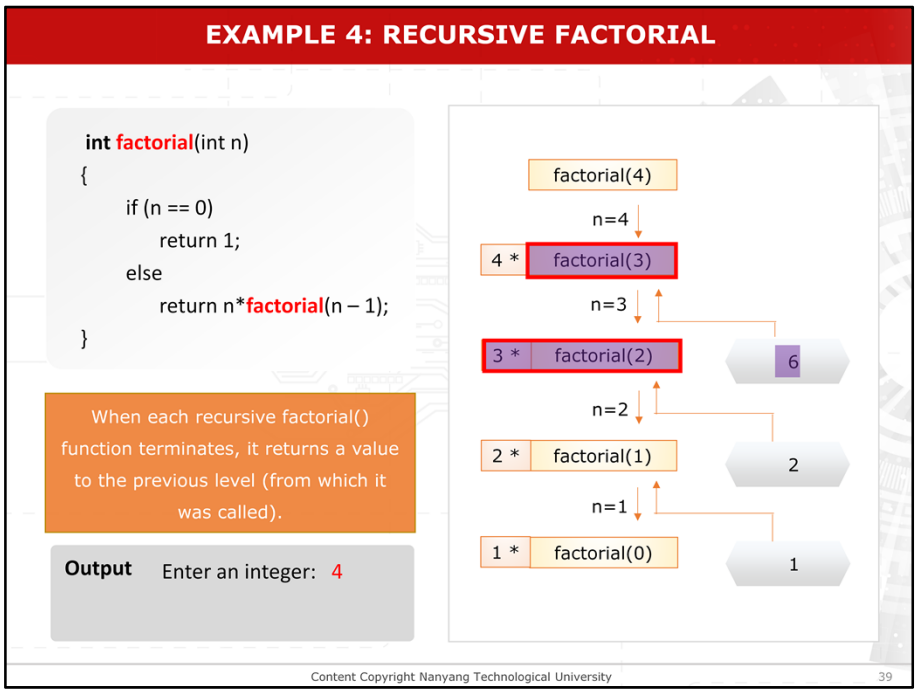
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
When each recursive factorial() function terminates, it returns a value to the previous level (from which it was called).
Output Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
n=2
2 * factorial(1)
2
n=1
1 * factorial(0)
1
Content Copyright Nanyang Technological University
37

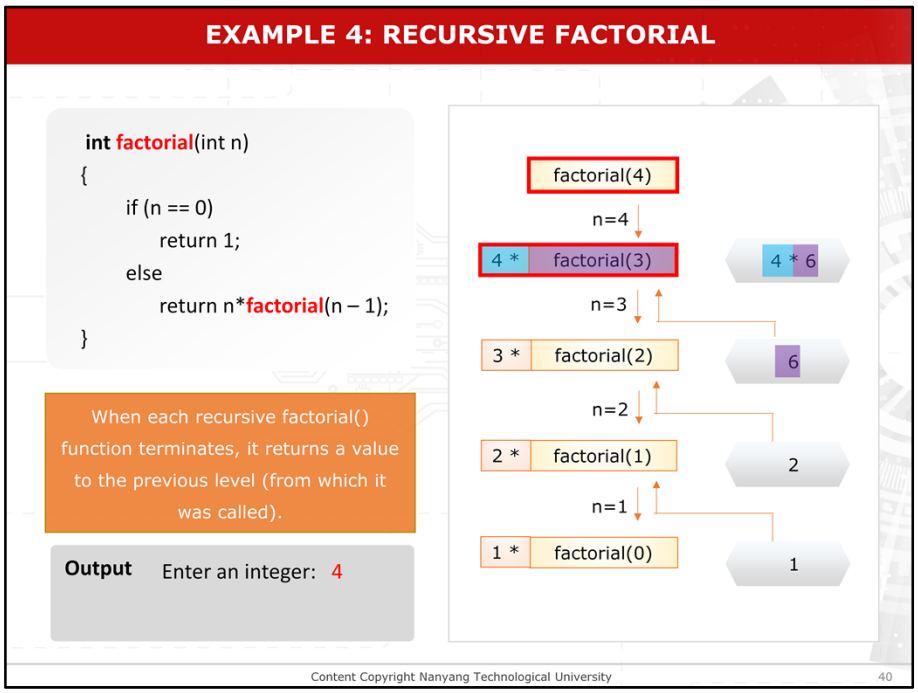
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
When each recursive factorial() function terminates, it returns a value to the previous level (from which it was called).
Output Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
3 * 2
n=2
2 * factorial(1)
2
n=1
1 * factorial(0)
1
Content Copyright Nanyang Technological University
38

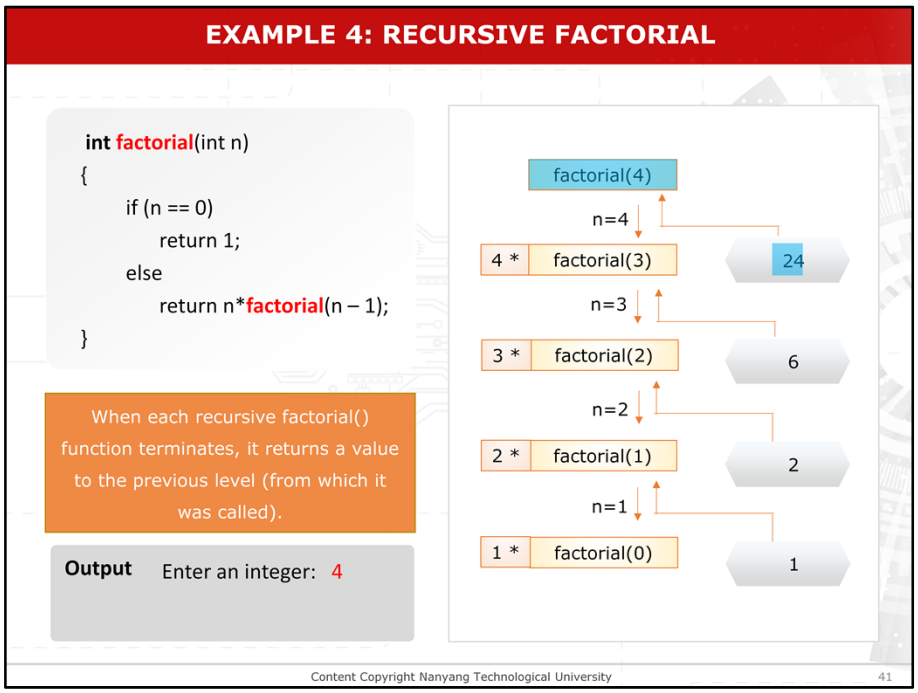
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
When each recursive factorial() function terminates, it returns a value to the previous level (from which it was called).
Output Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
n=3
3 * factorial(2)
6
n=2
2 * factorial(1)
2
n=1
1 * factorial(0)
1
Content Copyright Nanyang Technological University
39

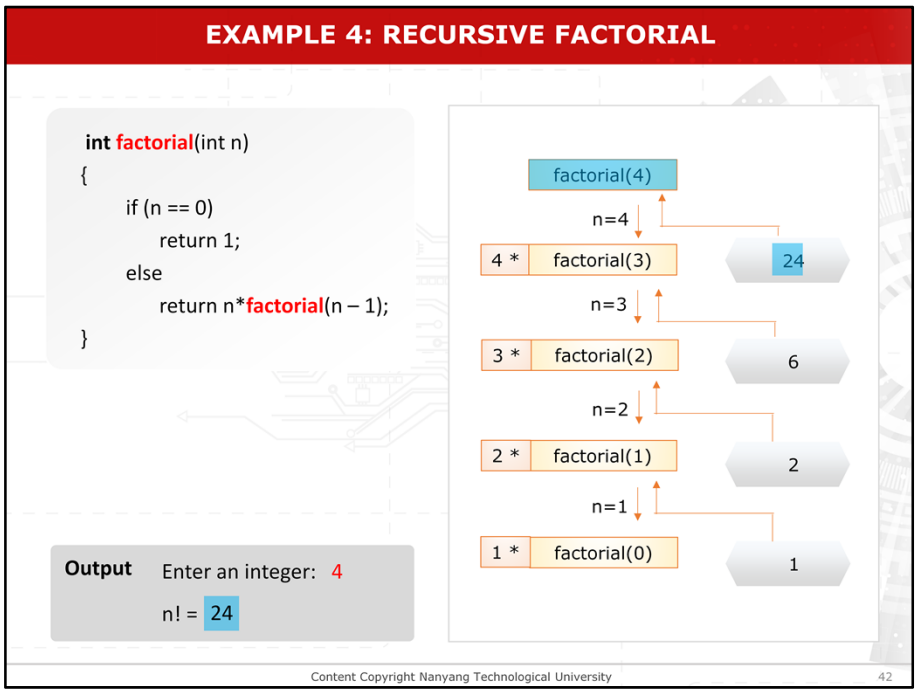
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
When each recursive factorial() function terminates, it returns a value to the previous level (from which it was called).
Output Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
4 * 6
n=3
3 * factorial(2)
6
n=2
2 * factorial(1)
2
n=1
1 * factorial(0)
1
Content Copyright Nanyang Technological University
40

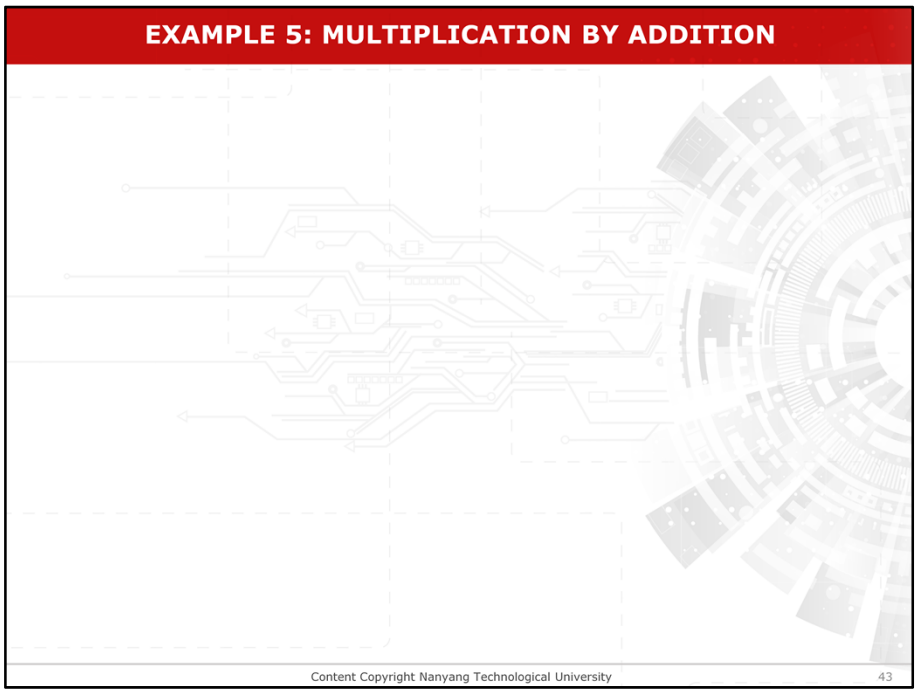
[no audio]


EXAMPLE 4: RECURSIVE FACTORIAL
int factorial(int n)
{
if (n == 0)
return 1;
else
return n*factorial(n – 1);
}
When each recursive factorial() function terminates, it returns a value to the previous level (from which it was called).
Output Enter an integer: 4
factorial(4)
n=4
4 * factorial(3)
24
n=3
3 * factorial(2)
6
n=2
2 * factorial(1)
2
n=1
1 * factorial(0)
1
Content Copyright Nanyang Technological University
41

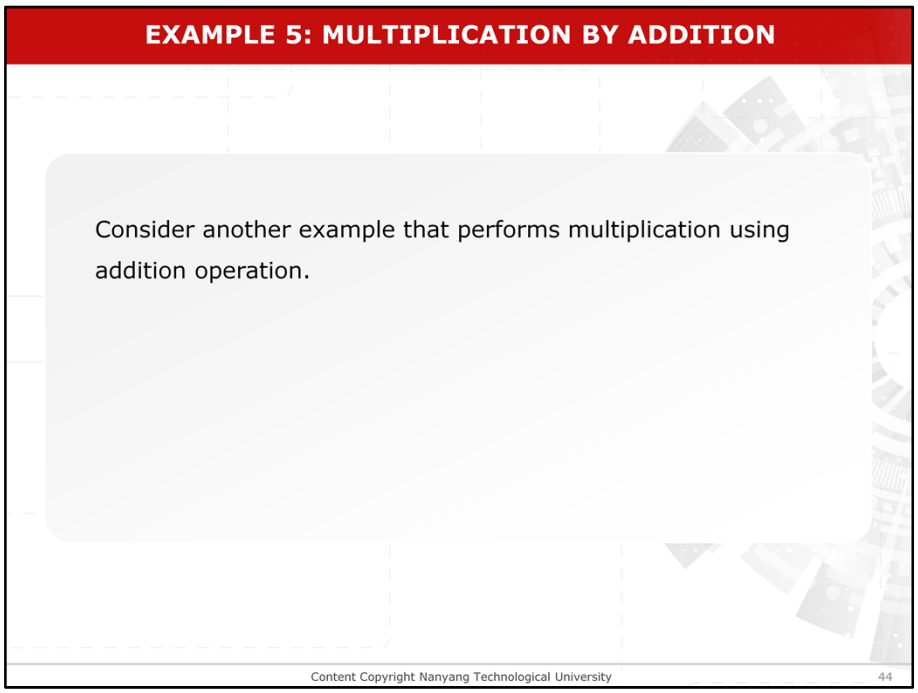
[no audio]



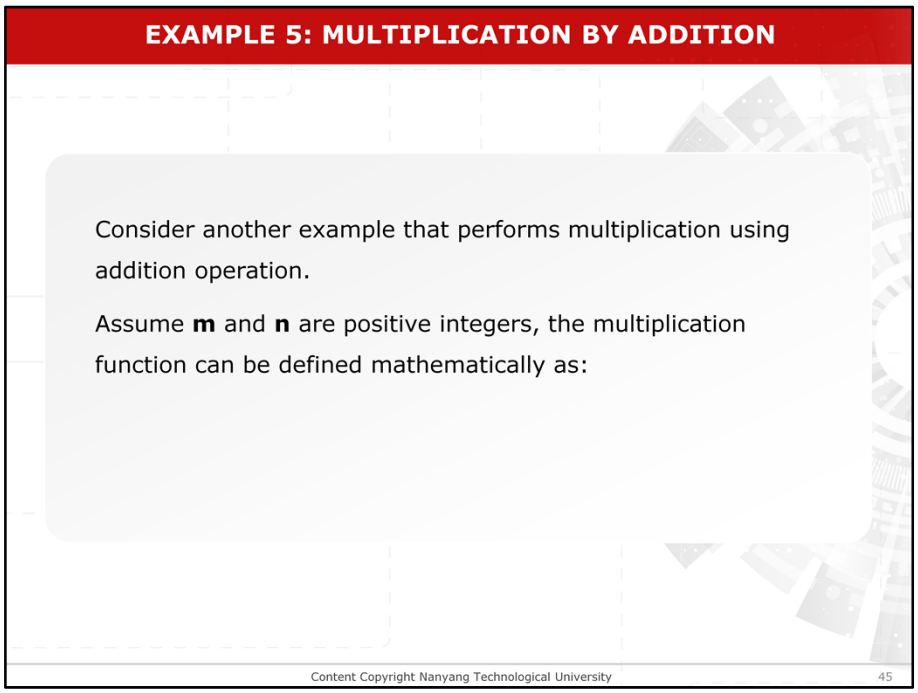
the result for4 factorial is 24.



Example 5: Multiplication by Addition



Consider another example that performs multiplication using addition operation.



Assume **m** and **n** are positive integers, the multiplication function can be defined mathematically as:

EXAMPLE 5: MULTIPLICATION BY ADDITION

Consider another example that performs multiplication using addition operation.

Assume **m** and **n** are positive integers, the multiplication function can be defined mathematically as:

m * n = m **if n=1**

if n is equal to 1, m multiply by n is equal to m

EXAMPLE 5: MULTIPLICATION BY ADDITION

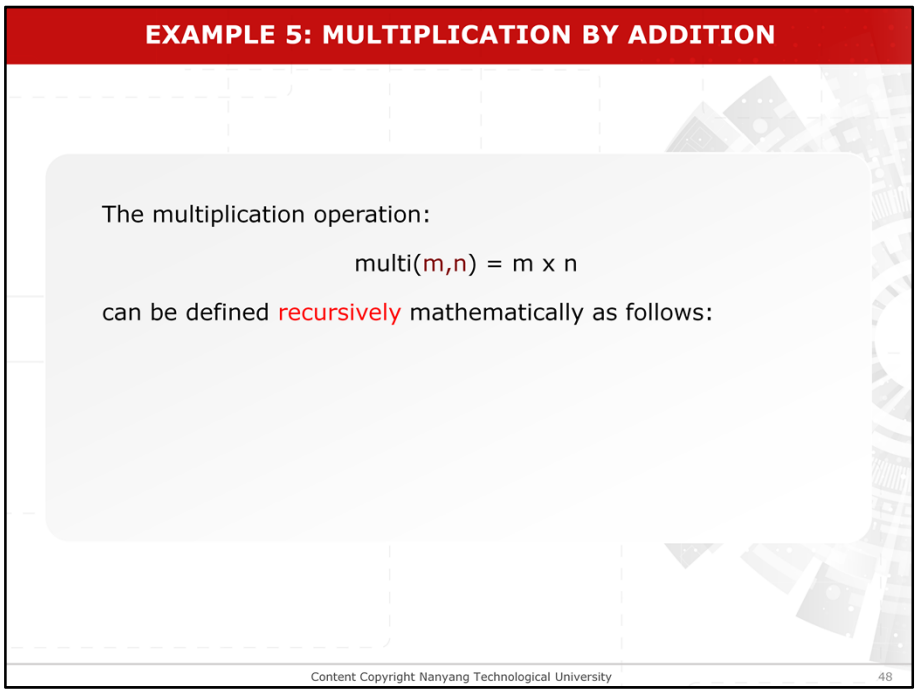
Consider another example that performs multiplication using addition operation.

Assume **m** and **n** are positive integers, the multiplication function can be defined mathematically as:

m * n = m **if n=1**

m * n = m + m*(n-1) **if n>1**

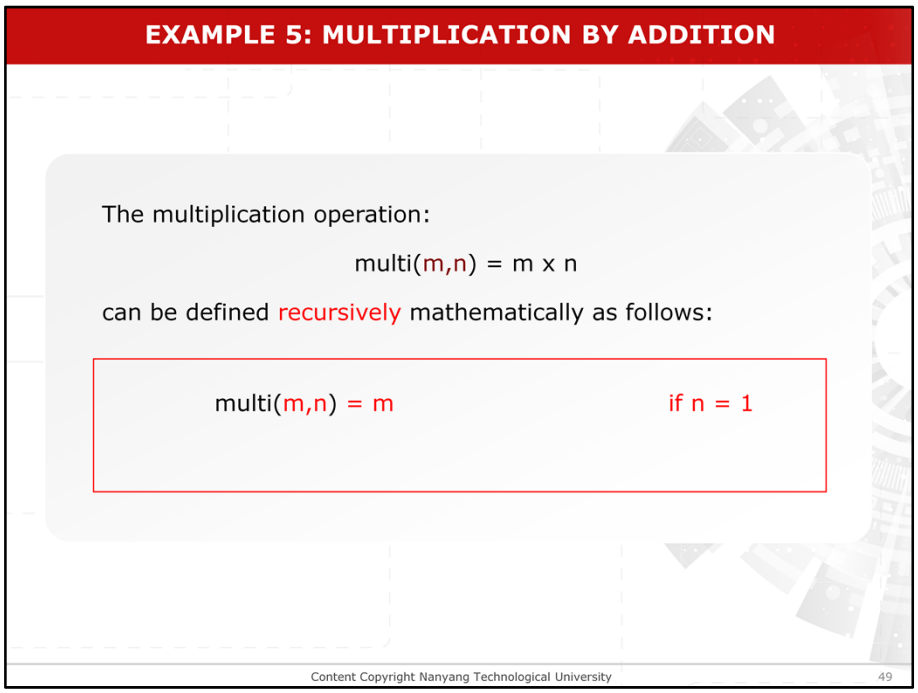
if n is greater than 1, m multiply by n is equal to m plus m multiply by (n minus 1)



The multiplication operation:

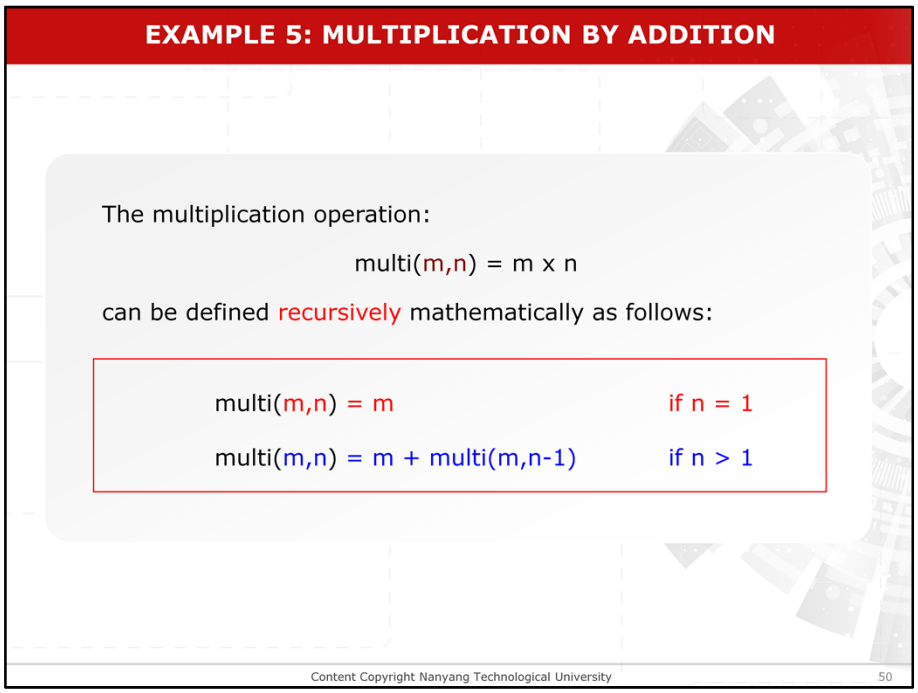
multi(m,n) is equals to m multiply by n

can be defined recursively mathematically as follows:



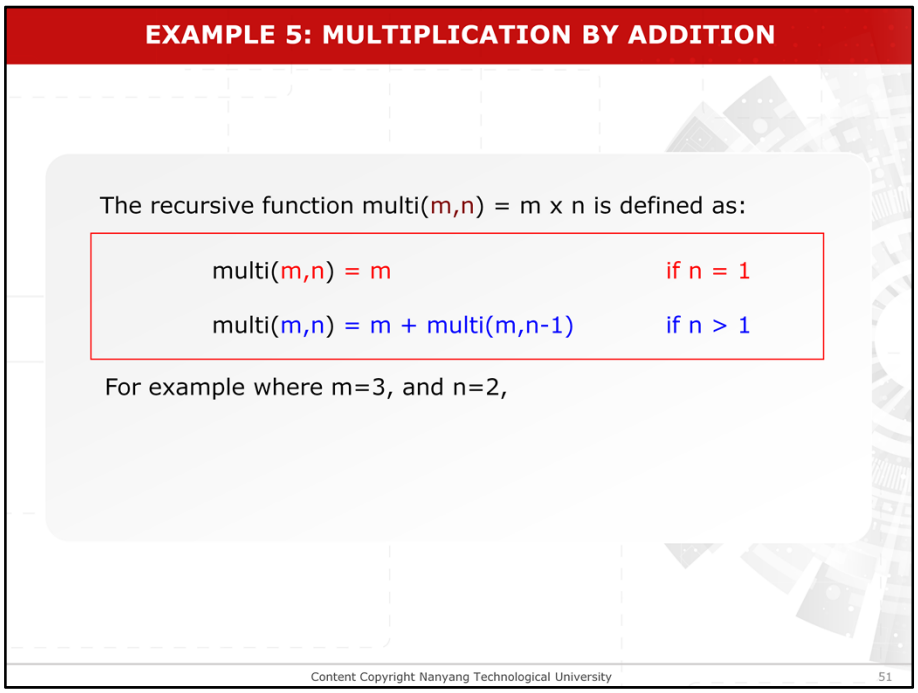
Terminal condition is

If n is equal to 1, multi (m, n) is equal to m.

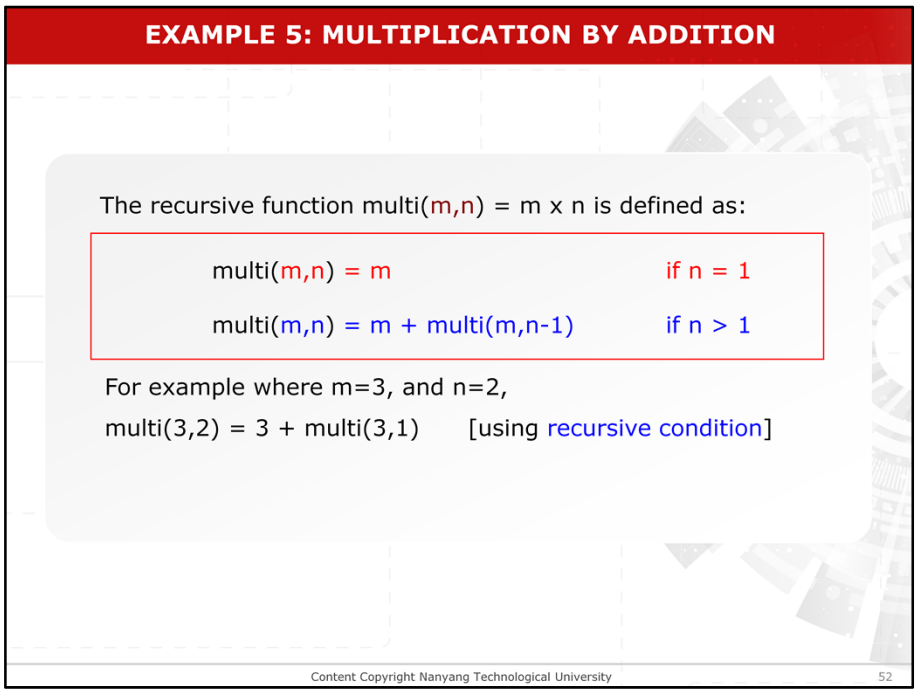


Recursive condition is

If n is greater than 1, multi(m, n) is equal to m plus multi(m, n minus 1)



Let see a simple example where we will apply both the terminal condition and recursive condition.



For m is equal to 3 and n is equal to 2, we first apply the recursive condition since n is greater than 1.

EXAMPLE 5: MULTIPLICATION BY ADDITION

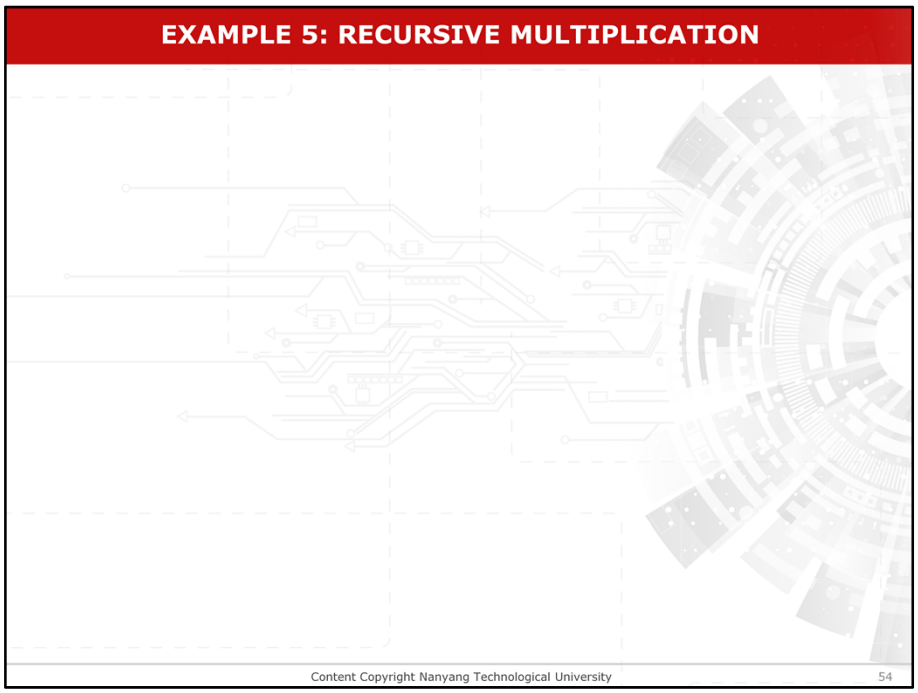
The recursive function multi(m,n) = m x n is defined as:

multi(m,n) = m	if n = 1
multi(m,n) = m + multi(m,n-1)	if n > 1

For example where m=3, and n=2,

multi(3,2) = 3 + multi(3,1) [using recursive condition]
= 3 + (3) [using terminating condition]
= 6

Then since n is reduced to 1, terminating condition is applied.
And the final result is computed as shown.



We will see how to write the Recursive Multiplication program code.

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

In the program, it implements a recursive integer multiplication function using **call by value**.

In the program, it implements a recursive integer multiplication function using call by value.

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

In the function **multi()**, it receives two integer arguments, **m** and **n**, from the calling function.

In the function **multi()**, it receives two integer arguments, **m** and **n**, from the calling function.

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

The function calls itself with the following **return** statement:

return m + multi(m, n - 1);

The function calls itself with the following **return** statement:

return m plus multi(m, n minus 1);

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

For each new function call, a new set of parameters is created and used. The value **n** also gets smaller,

For each new function call, a new set of parameters is created and used. The value **n** also gets smaller,

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

For each new function call, a new set of parameters is created and used. The value **n** also gets smaller, and eventually equals to 1, where the terminating condition is reached.

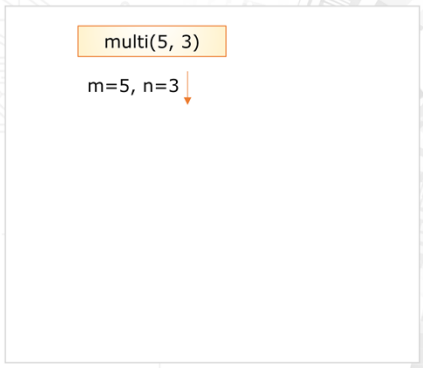
and eventually equal to 1, where the terminating condition is reached.

EXAMPLE 5: RECURSIVE MULTIPLICATION

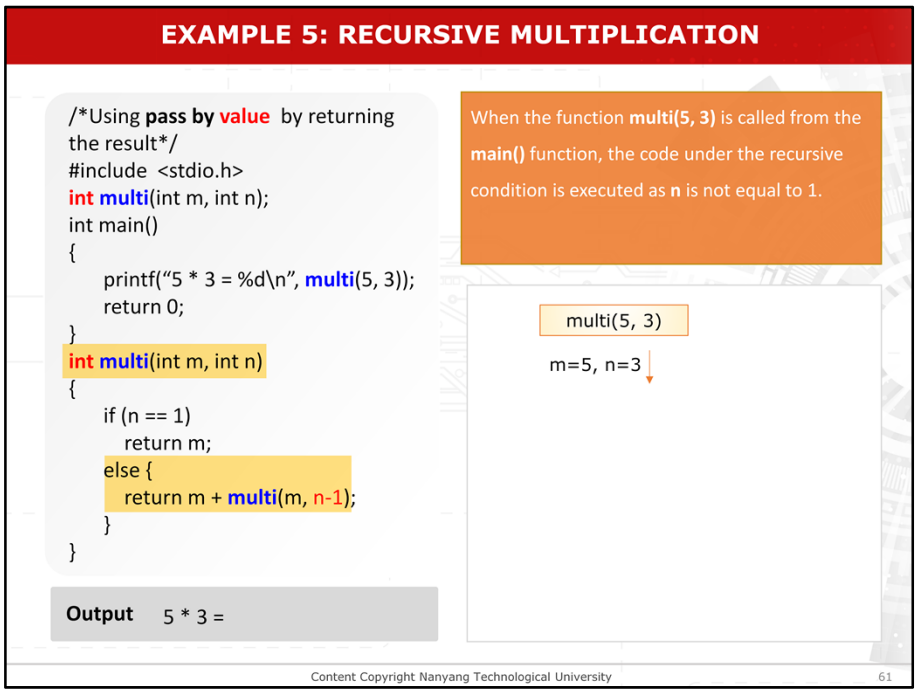
```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

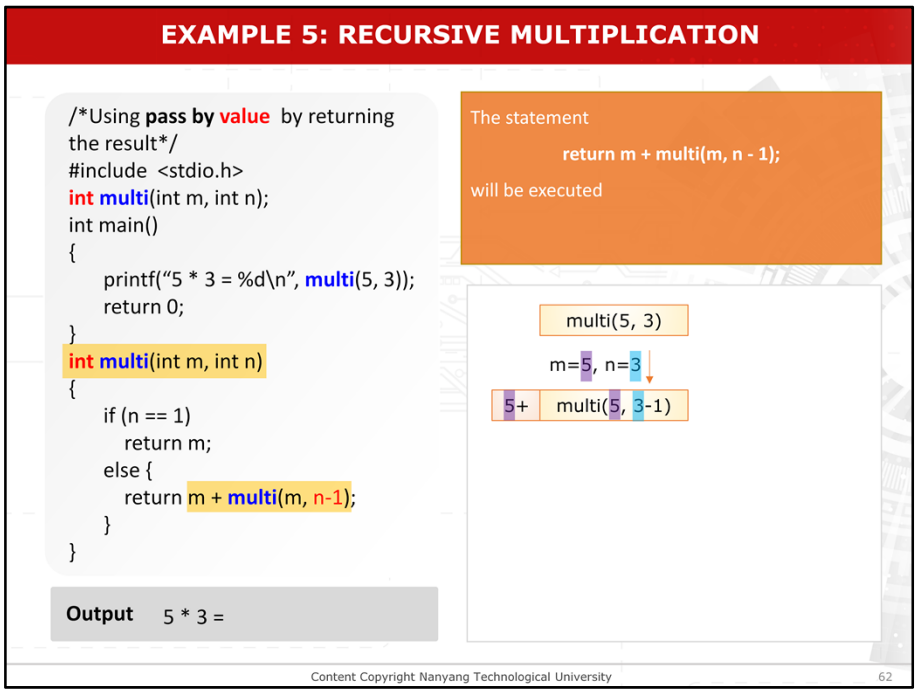
When the function **multi(5, 3)** is called from the **main()** function,



When the function **multi(5, 3)** is called from the **main()** function,



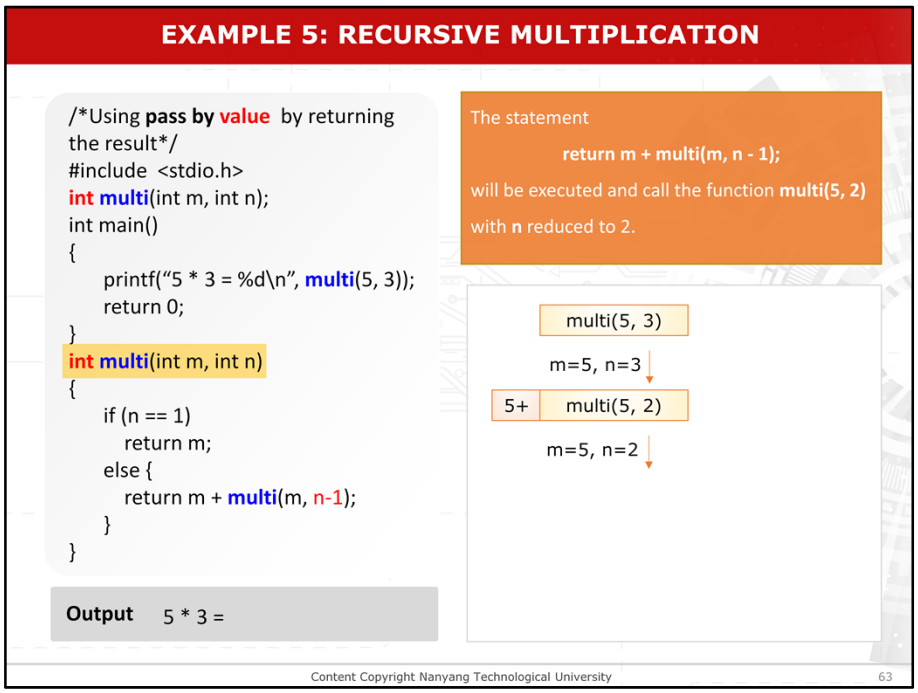
the code under the recursive condition is executed as **n** is not equal to 1.



The statement

return m plus multi(m, n minus 1);

will be executed



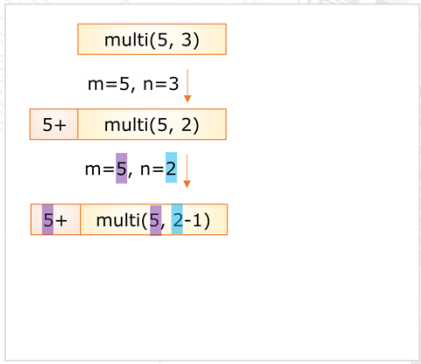
and call the function **multi(5, 2)** with **n** reduced to 2.

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

This will in turn execute the function call **multi(5, 1)**.



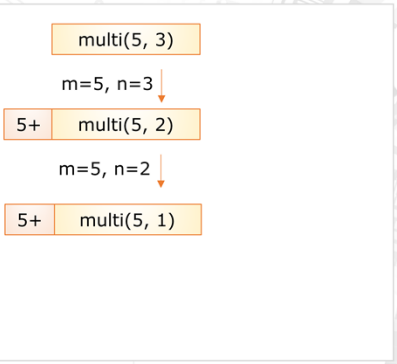
This will in turn execute the function call **multi(5, 1)**.

EXAMPLE 5: RECURSIVE MULTIPLICATION

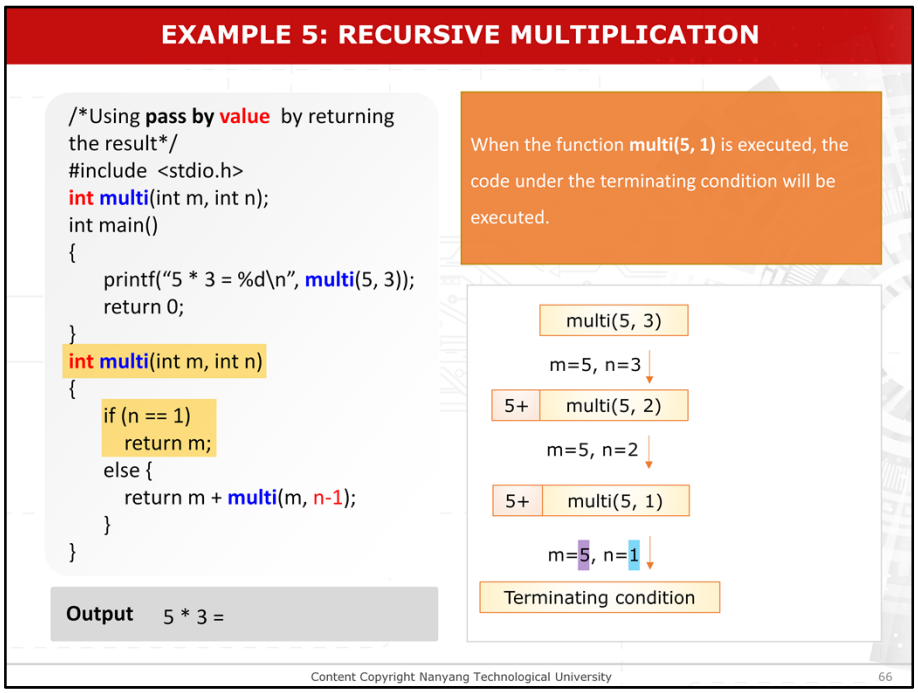
```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

This will in turn execute the function call **multi(5, 1)**.



[no audio]



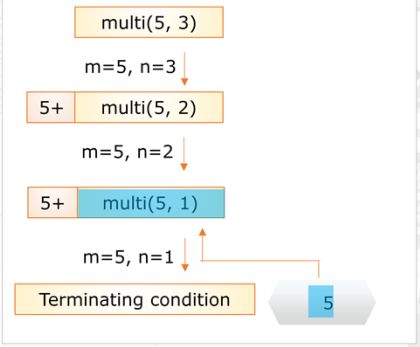
When the function **multi(5, 1)** is executed, the code under the terminating condition will be executed.

EXAMPLE 5: RECURSIVE MULTIPLICATION

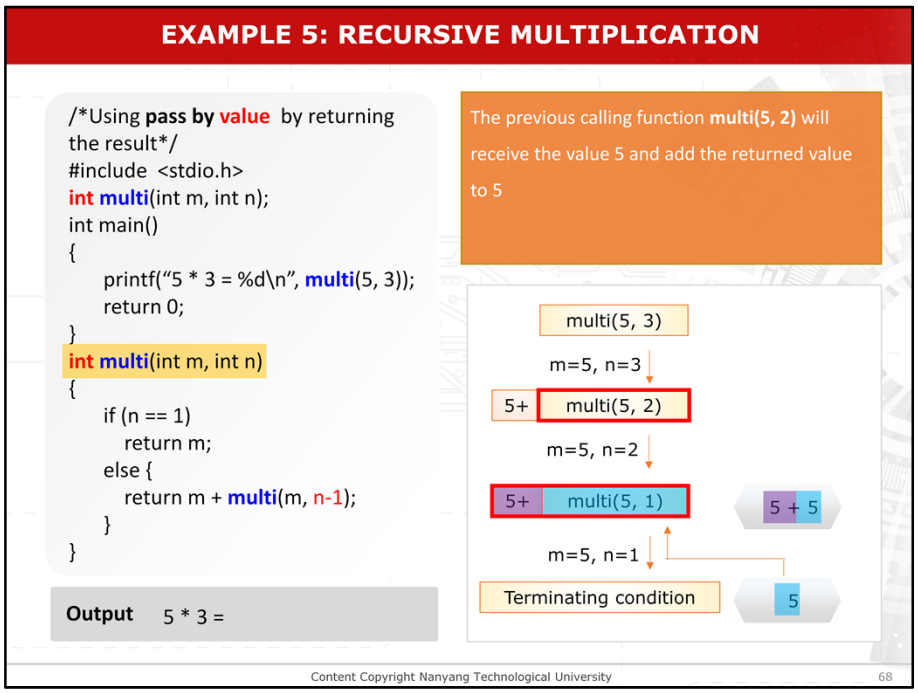
```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

The value 5 will be returned to the calling function.



The value 5 will be returned to the calling function.



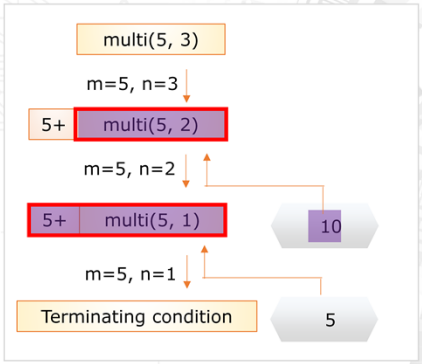
The previous calling function **multi(5, 2)** will receive the value 5 and add the returned value to 5

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

The previous calling function **multi(5, 2)** will receive the value 5 and add the returned value to 5 and return the sum of 10



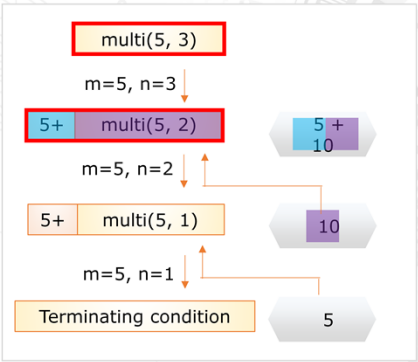
and return the sum of 10

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
       return m;
     else {
       return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

The previous calling function **multi(5, 2)** will receive the value 5 and add the returned value to 5 and return the sum of 10 to the previous calling function **multi(5, 3)**.



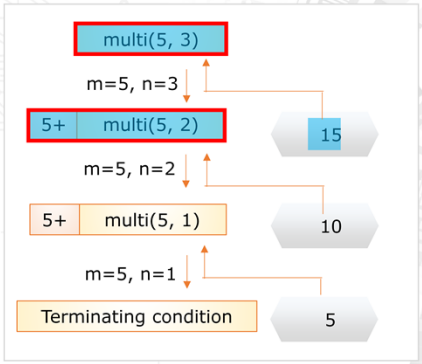
to the previous calling function **multi(5, 3)**.

EXAMPLE 5: RECURSIVE MULTIPLICATION

```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 =

The function **multi(5, 3)** will add the returned value to 5 and return the sum of 15 to the calling function **main()**.



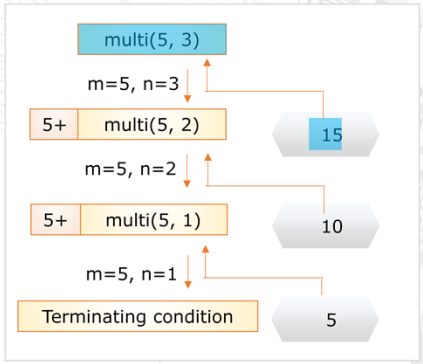
The function **multi(5, 3)** will add the returned value to 5 and return the sum of 15 to the calling function **main()**.

EXAMPLE 5: RECURSIVE MULTIPLICATION

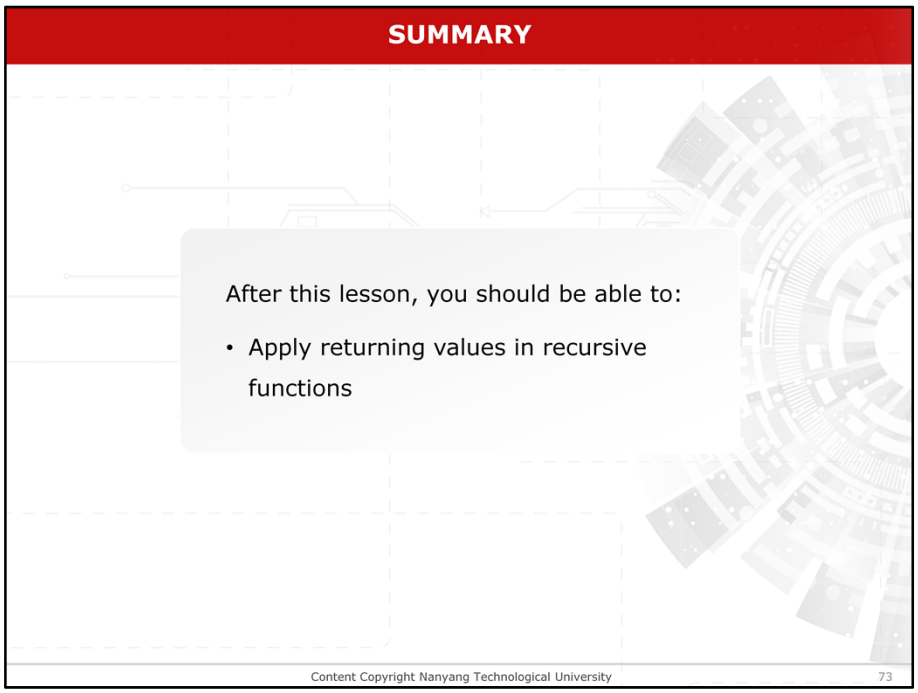
```
/*Using pass by value  by returning
the result*/
#include  <stdio.h>
int multi(int m, int n);
int main()
{
     printf("5 * 3 = %d\n", multi(5, 3));
     return 0;
}
int multi(int m, int n)
{
     if (n == 1)
        return m;
     else {
        return m + multi(m, n-1);
     }
}
```

Output 5 * 3 = 15

In the **main()** function, the returned value of 15 will then be printed to the screen.



In the **main()** function, the returned value of 15 will then be printed to the screen.



In summary, after viewing this video lesson, you should be able to apply returning values in recursive functions