

## OVERVIEW

The following are the coverage for Recursion:

- What is Recursion?
- Recursive Functions: Examples
- **Recursive Functions: Returning Value**
- Recursive Functions: Call by Reference
- Recursion in Arrays
- How to Design Recursive Functions

Content Copyright Nanyang Technological University

2

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.

## LESSON OBJECTIVES

After this lesson, you should be able to:

- Apply returning values in recursive functions

Content Copyright Nanyang Technological University

3

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

## EXAMPLE 4: FACTORIAL

Content Copyright Nanyang Technological University

4

### Example 4: Factorial

## EXAMPLE 4: FACTORIAL

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

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Content Copyright Nanyang Technological University

5

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:

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Non-recursive approach: use for loop

Content Copyright Nanyang Technological University

6

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:

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Non-recursive approach: use for loop

Content Copyright Nanyang Technological University

7

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:

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Non-recursive approach: use for loop

Output Enter an integer: 4

n! = 24

Content Copyright Nanyang Technological University

8

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.

## EXAMPLE 4: RECURSIVE FACTORIAL

Content Copyright Nanyang Technological University

9

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().

#### EXAMPLE 4: RECURSIVE FACTORIAL

A more precise mathematical definition (recursive definition)

$$\begin{aligned} n! &= 1 && \text{if } n = 0 \\ n! &= n \times (n-1)! && \text{if } n > 0 \end{aligned}$$

Content Copyright Nanyang Technological University

10

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.

#### EXAMPLE 4: RECURSIVE FACTORIAL

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

Content Copyright Nanyang Technological University

11

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

## EXAMPLE 4: RECURSIVE FACTORIAL

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

as  $4 > 0$ , use  $n! = n \times (n-1)!$

$$4! = 4 \times 3!$$

Content Copyright Nanyang Technological University

12

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

## EXAMPLE 4: RECURSIVE FACTORIAL

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

as  $3 > 0$ , use  $n! = n \times (n-1)!$

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \end{aligned}$$

Content Copyright Nanyang Technological University

13

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

## EXAMPLE 4: RECURSIVE FACTORIAL

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

as  $2 > 0$ , use  $n! = n \times (n-1)!$

$$\begin{aligned}4! &= 4 \times 3! \\&= 4 \times (3 \times 2!) \\&= 4 \times (3 \times (2 \times 1!))\end{aligned}$$

Content Copyright Nanyang Technological University

14

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

#### EXAMPLE 4: RECURSIVE FACTORIAL

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

as  $1 > 0$ , use  $n! = n \times (n-1)!$

$$\begin{aligned}4! &= 4 \times 3! \\&= 4 \times (3 \times 2!) \\&= 4 \times (3 \times (2 \times 1!)) \\&= 4 \times (3 \times (2 \times (1 \times 0!)))\end{aligned}$$

Content Copyright Nanyang Technological University

15

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

## EXAMPLE 4: RECURSIVE FACTORIAL

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

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \\ &= 4 \times (3 \times (2 \times 1!)) \\ &= 4 \times (3 \times (2 \times (1 \times 0!))) \\ &\quad \hline \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \end{aligned}$$

Terminating condition:  $0! = 1$

Content Copyright Nanyang Technological University

16

$0$  factorial is defined as  $1$ .

This is the terminating condition.

## EXAMPLE 4: RECURSIVE FACTORIAL

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

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

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \\ &= 4 \times (3 \times (2 \times 1!)) \\ &= 4 \times (3 \times (2 \times (1 \times 0!))) \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \\ &= 4 \times (3 \times (2 \times 1)) \end{aligned}$$

Content Copyright Nanyang Technological University

17

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,

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

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \\ &= 4 \times (3 \times (2 \times 1!)) \\ &= 4 \times (3 \times (2 \times (1 \times 0!))) \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \\ &= 4 \times (3 \times (2 \times 1)) \\ &= 4 \times (3 \times 2) \end{aligned}$$

Content Copyright Nanyang Technological University

18

[no audio]

## EXAMPLE 4: RECURSIVE FACTORIAL

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

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

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \\ &= 4 \times (3 \times (2 \times 1!)) \\ &= 4 \times (3 \times (2 \times (1 \times 0!))) \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \\ &= 4 \times (3 \times (2 \times 1)) \\ &= 4 \times (3 \times 2) \\ &= 4 \times 6 \end{aligned}$$

Content Copyright Nanyang Technological University

19

[no audio]

## EXAMPLE 4: RECURSIVE FACTORIAL

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

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

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \\ &= 4 \times (3 \times (2 \times 1!)) \\ &= 4 \times (3 \times (2 \times (1 \times 0!))) \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \\ &= 4 \times (3 \times (2 \times 1)) \\ &= 4 \times (3 \times 2) \\ &= 4 \times 6 \\ &= 24 \end{aligned}$$

Content Copyright Nanyang Technological University

20

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);
}
```

Content Copyright Nanyang Technological University

21

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);
}
```

Content Copyright Nanyang Technological University

22

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

## EXAMPLE 4: RECURSIVE FACTORIAL

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

Recursive condition where the value n gets smaller by 1 each time, and eventually equals to 0.

/\* recursive condition \*/

Content Copyright Nanyang Technological University

23

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.

#### EXAMPLE 4: RECURSIVE FACTORIAL

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

Terminating condition where the recursion will stop.

/\* terminating condition \*/

Content Copyright Nanyang Technological University

24

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.

## EXAMPLE 4: RECURSIVE FACTORIAL

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

Assuming that the user enters 4 for the input value.

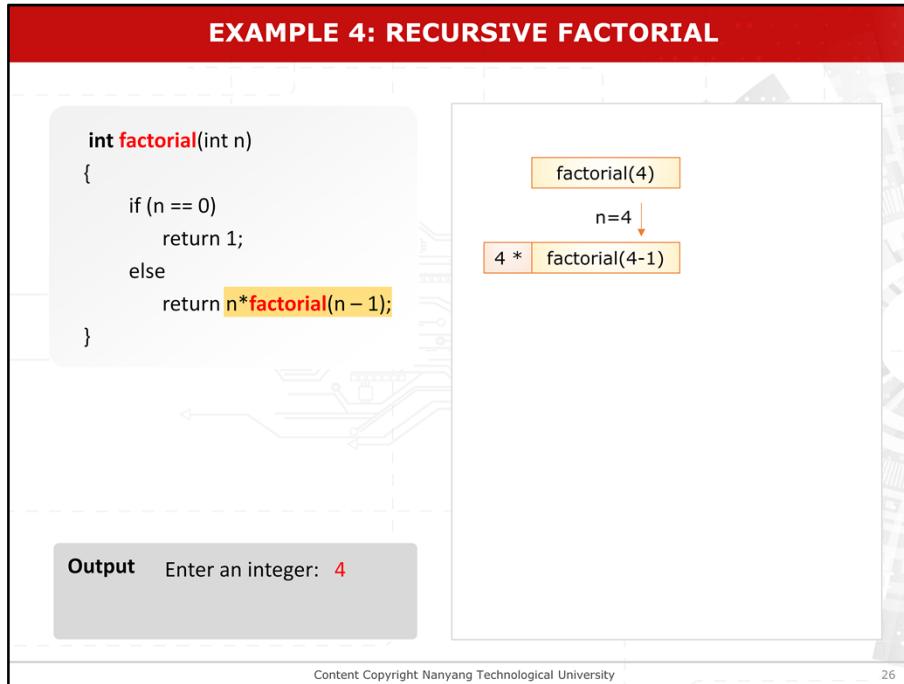
**Output** Enter an integer: 4

Content Copyright Nanyang Technological University

25

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

## EXAMPLE 4: RECURSIVE FACTORIAL



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

The diagram illustrates the execution of a recursive factorial function. On the left, the code for `factorial` is shown:

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

On the right, the recursive call `factorial(4)` is expanded into `4 * factorial(3)`. A red arrow points from the value 4 to the parameter `n` in the second term. Below the code, there is a small illustration of a computer monitor and keyboard.

**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);
}
```

Output Enter an integer: 4

factorial(4)

n=4 ↓

4 \* factorial(3)

n=3 ↓

3 \* factorial(3-1)

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);
}
```

Output Enter an integer: 4

The diagram shows the recursive calls for calculating factorial(4). It starts with a call to factorial(4) with n=4. This leads to 4 \* factorial(3) with n=3. Finally, it leads to 3 \* factorial(2) with n=2. Arrows indicate the flow from the current call to the next recursive step.

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(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);
}
```

Output Enter an integer: 4

factorial(4)

n=4 ↓

4 \* factorial(3)

n=3 ↓

3 \* factorial(2)

n=2 ↓

2 \* factorial(1)

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

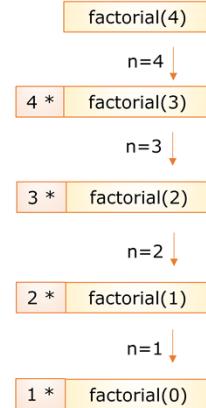
[no audio]

## EXAMPLE 4: RECURSIVE FACTORIAL

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

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

**Output** Enter an integer: 4



Content Copyright Nanyang Technological University

33

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

## EXAMPLE 4: RECURSIVE FACTORIAL

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

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

Content Copyright Nanyang Technological University

34

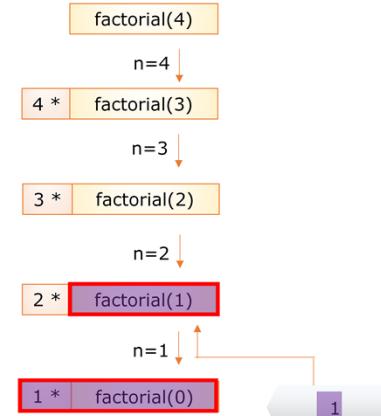
When the last **factorial()** function is called, it returns the value 1, as **0 factorial** equals to 1.

## 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



Content Copyright Nanyang Technological University

35

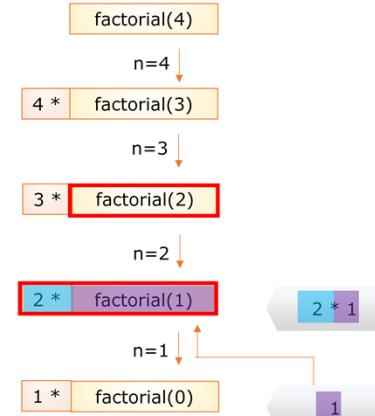
When each recursive **factorial()** function terminates, it returns a value to the previous level (from which it was called).

## 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



Content Copyright Nanyang Technological University

36

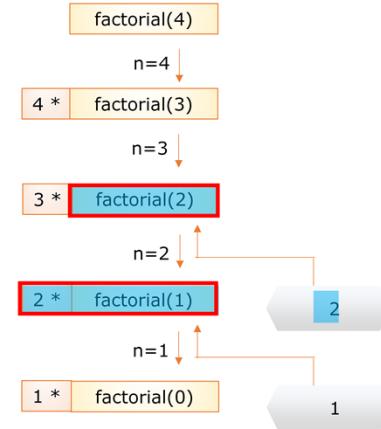
[no audio]

## 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



Content Copyright Nanyang Technological University

37

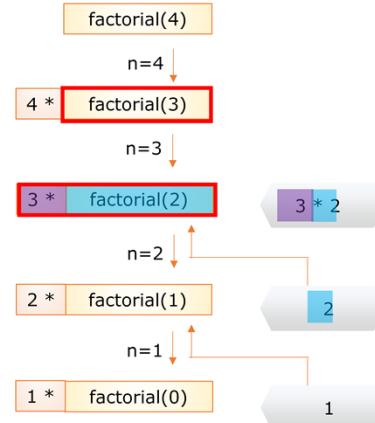
[no audio]

## 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



Content Copyright Nanyang Technological University

38

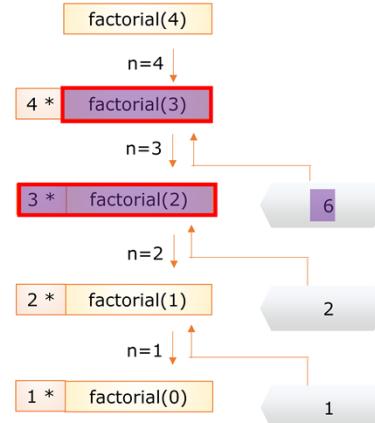
[no audio]

## 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



Content Copyright Nanyang Technological University

39

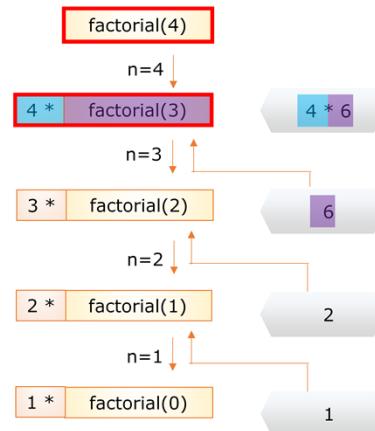
[no audio]

## 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



Content Copyright Nanyang Technological University

40

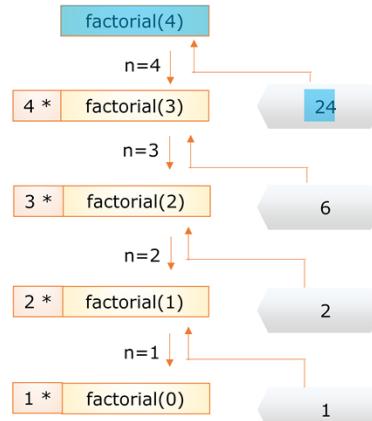
[no audio]

## 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



Content Copyright Nanyang Technological University

41

[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**  
n! = **24**

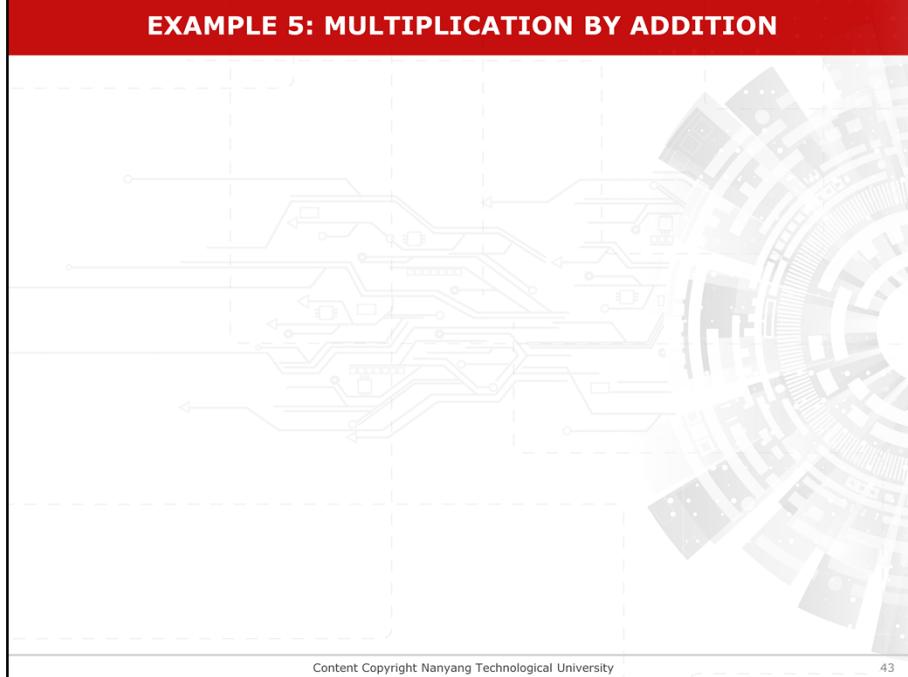
```
graph TD
    A[factorial(4)] --> B[4 * factorial(3)]
    B --> C[3 * factorial(2)]
    C --> D[2 * factorial(1)]
    D --> E[1 * factorial(0)]
    E --> F[24]
    E --> G[6]
    E --> H[2]
    E --> I[1]
```

Content Copyright Nanyang Technological University

42

the result for 4 factorial is 24.

### EXAMPLE 5: MULTIPLICATION BY ADDITION



Content Copyright Nanyang Technological University

43

Example 5: Multiplication by Addition

### EXAMPLE 5: MULTIPLICATION BY ADDITION

Consider another example that performs multiplication using addition operation.

Consider another example that performs multiplication using addition operation.

### 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:

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:

$$m * n = m \quad \text{if } n=1$$

Content Copyright Nanyang Technological University

46

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:

$$m * n = m \quad \text{if } n=1$$

$$m * n = m + m*(n-1) \quad \text{if } n>1$$

Content Copyright Nanyang Technological University

47

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

## EXAMPLE 5: MULTIPLICATION BY ADDITION

The multiplication operation:

$$\text{multi}(m,n) = m \times n$$

can be defined **recursively** mathematically as follows:

Content Copyright Nanyang Technological University

48

The multiplication operation:

$\text{multi}(m,n)$  is equals to m multiply by n

can be defined **recursively** mathematically as follows:

## EXAMPLE 5: MULTIPLICATION BY ADDITION

The multiplication operation:

$$\text{multi}(m,n) = m \times n$$

can be defined **recursively** mathematically as follows:

$$\boxed{\text{multi}(m,n) = m \quad \text{if } n = 1}$$

Content Copyright Nanyang Technological University

49

Terminal condition is

If  $n$  is equal to 1,  $\text{multi}(m, n)$  is equal to  $m$ .

## EXAMPLE 5: MULTIPLICATION BY ADDITION

The multiplication operation:

$$\text{multi}(m,n) = m \times n$$

can be defined **recursively** mathematically as follows:

$$\text{multi}(m,n) = m \quad \text{if } n = 1$$

$$\text{multi}(m,n) = m + \text{multi}(m,n-1) \quad \text{if } n > 1$$

Content Copyright Nanyang Technological University

50

Recursive condition is

If  $n$  is greater than 1,  $\text{multi}(m, n)$  is equal to  $m$  plus  $\text{multi}(m, n minus 1)$

## EXAMPLE 5: MULTIPLICATION BY ADDITION

The recursive function  $\text{multi}(m,n) = m \times n$  is defined as:

$$\text{multi}(m,n) = m \quad \text{if } n = 1$$

$$\text{multi}(m,n) = m + \text{multi}(m,n-1) \quad \text{if } n > 1$$

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

Content Copyright Nanyang Technological University

51

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

## EXAMPLE 5: MULTIPLICATION BY ADDITION

The recursive function  $\text{multi}(m,n) = m \times n$  is defined as:

$$\text{multi}(m,n) = m \quad \text{if } n = 1$$

$$\text{multi}(m,n) = m + \text{multi}(m,n-1) \quad \text{if } n > 1$$

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

$\text{multi}(3,2) = 3 + \text{multi}(3,1)$  [using recursive condition]

Content Copyright Nanyang Technological University

52

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  $\text{multi}(m,n) = m \times n$  is defined as:

$$\text{multi}(m,n) = m \quad \text{if } n = 1$$

$$\text{multi}(m,n) = m + \text{multi}(m,n-1) \quad \text{if } n > 1$$

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

$$\begin{aligned} \text{multi}(3,2) &= 3 + \text{multi}(3,1) && [\text{using recursive condition}] \\ &= 3 + (3) && [\text{using terminating condition}] \\ &= 6 \end{aligned}$$

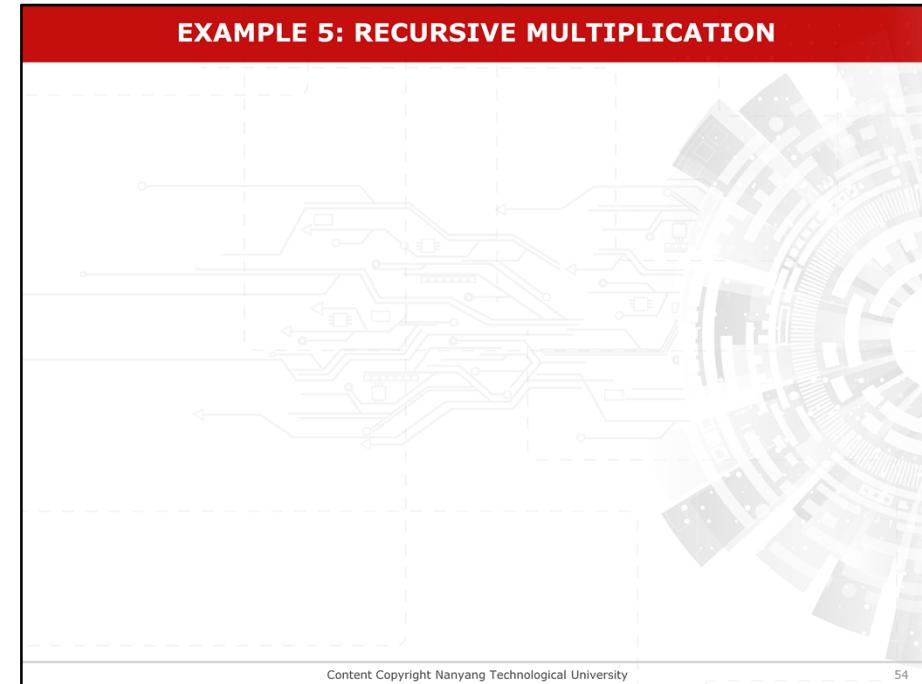
Content Copyright Nanyang Technological University

53

Then since  $n$  is reduced to 1, terminating condition is applied.

And the final result is computed as shown.

## EXAMPLE 5: RECURSIVE MULTIPLICATION



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**.

Content Copyright Nanyang Technological University

55

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.

Content Copyright Nanyang Technological University

56

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);
```

Content Copyright Nanyang Technological University

57

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,

Content Copyright Nanyang Technological University

58

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.

Content Copyright Nanyang Technological University

59

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,

multi(5, 3)

m=5, n=3 ↓

Content Copyright Nanyang Technological University

60

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

Output 5 \* 3 =

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.

multi(5, 3)

m=5, n=3 ↓

Content Copyright Nanyang Technological University

61

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

The statement  
return m + multi(m, n - 1);  
will be executed

multi(5, 3)

m=5, n=3

5+ multi(5, 3-1)

Content Copyright Nanyang Technological University

62

The statement

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

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 statement

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

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

multi(5, 3)

m=5, n=3 ↓

5+ multi(5, 2)

m=5, n=2 ↓

Content Copyright Nanyang Technological University

63

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).**

multi(5, 3)

m=5, n=3 ↓

5+ multi(5, 2)

m=5, n=2 ↓

5+ multi(5, 2-1)

Content Copyright Nanyang Technological University

64

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).**

multi(5, 3)

m=5, n=3 ↓

5+ multi(5, 2)

m=5, n=2 ↓

5+ multi(5, 1)

Content Copyright Nanyang Technological University

65

[no audio]

## 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, 1)** is executed, the code under the terminating condition will be executed.

multi(5, 3)

m=5, n=3 ↓

5+ multi(5, 2)

m=5, n=2 ↓

5+ multi(5, 1)

m=5, n=1 ↓

Terminating condition

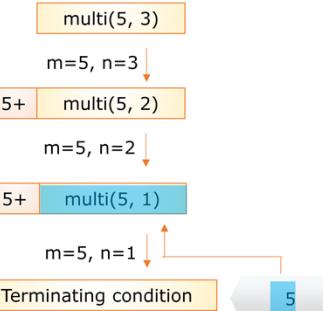
When the function **multi(5, 1)** is executed, the code under the terminating condition will be executed.

## 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.



Content Copyright Nanyang Technological University

67

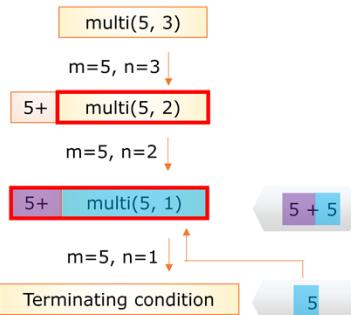
The value 5 will be returned to the calling function.

## 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);  
    }  
}
```

Output 5 \* 3 =

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



Content Copyright Nanyang Technological University

68

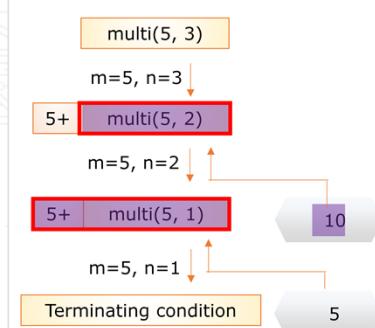
The previous calling function **multi(5, 2)** will receive the value 5 and add the returned value to 5

## 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



Content Copyright Nanyang Technological University

69

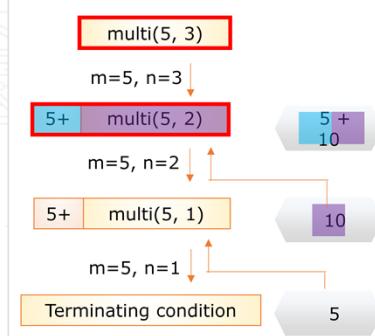
and return the sum of 10

## 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)**.



Content Copyright Nanyang Technological University

70

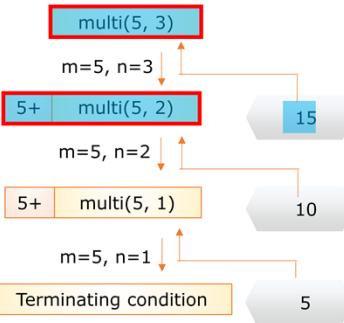
to the previous calling function **multi(5, 3)**.

## 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()**.



Content Copyright Nanyang Technological University

71

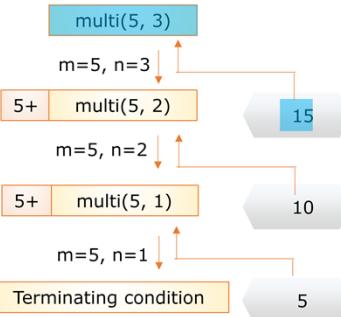
The function **multi(5, 3)** will add the returned value to 5 and return the sum of 15 to the calling function **main()**.

## 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.



Content Copyright Nanyang Technological University

72

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

## SUMMARY

After this lesson, you should be able to:

- Apply returning values in recursive functions

Content Copyright Nanyang Technological University

73

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