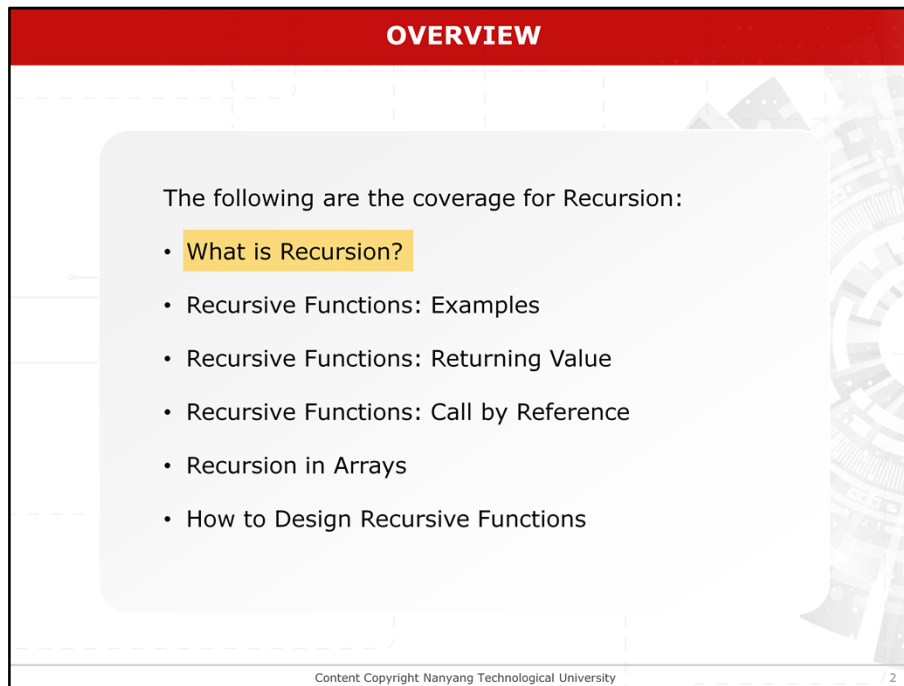


One of the most interesting features of C is the ability of a function to call itself. This is known as recursion.

The use of recursion is particularly convenient for those problems that can be defined in naturally recursive terms, though such problems can also be programmed using non-recursive techniques. In this lecture, we will discuss the concepts of recursion and give a few examples on recursive functions.

The slide features a red header with the word "OVERVIEW" in white. Below the header is a light gray background with a faint architectural pattern. A white rounded rectangle in the center contains the text "The following are the coverage for Recursion:" followed by a bulleted list. The first item, "What is Recursion?", is highlighted with a yellow background. The footer contains the text "Content Copyright Nanyang Technological University" and a small number "2".

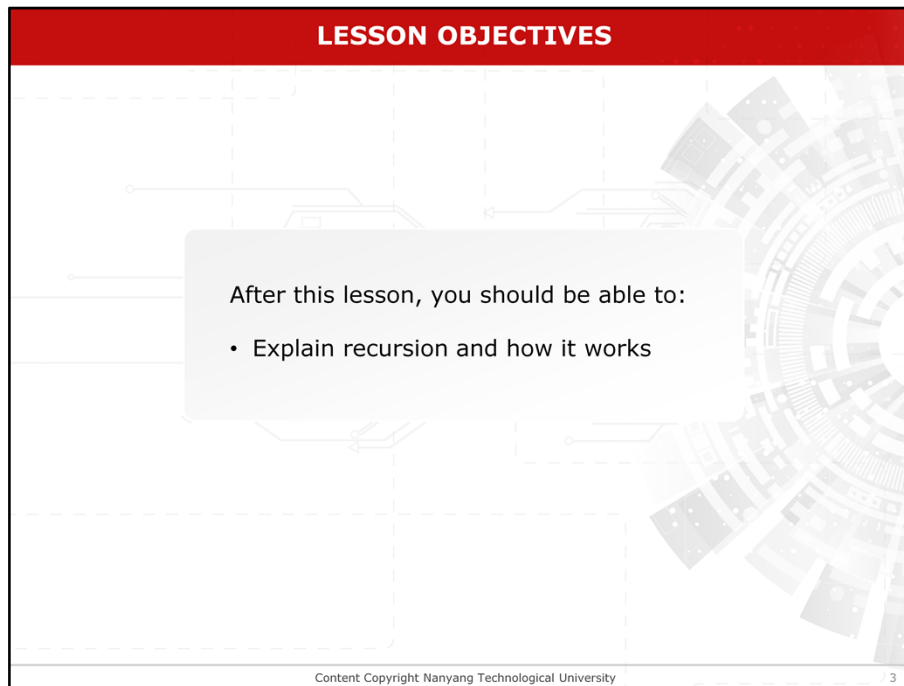
OVERVIEW

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 first part on What is Recursion?



LESSON OBJECTIVES

After this lesson, you should be able to:

- Explain recursion and how it works

Content Copyright Nanyang Technological University 3

After this lesson, you should be able to explain recursion and how it works.

WHAT IS RECURSION?

C functions have the following properties:

- A function, when called, will accomplish a certain job.

Content Copyright Nanyang Technological University 4

In C, functions have the following properties:
A function, when called, will accomplish a certain job.

WHAT IS RECURSION?

C functions have the following properties:

- A function, when called, will accomplish a certain job.
- When a function **F()** is called, control is transferred from the calling point to the first statement in **F()**. After the function finishes execution, the control will return back to the calling point. The next statement after the function call will be executed.

Content Copyright Nanyang Technological University

5

When a function **F()** is called, control is transferred from the calling point to the first statement in **F()**.

After the function finishes execution, the control will return back to the calling point.

The next statement after the function call will be executed.

WHAT IS RECURSION?

C functions have the following properties:

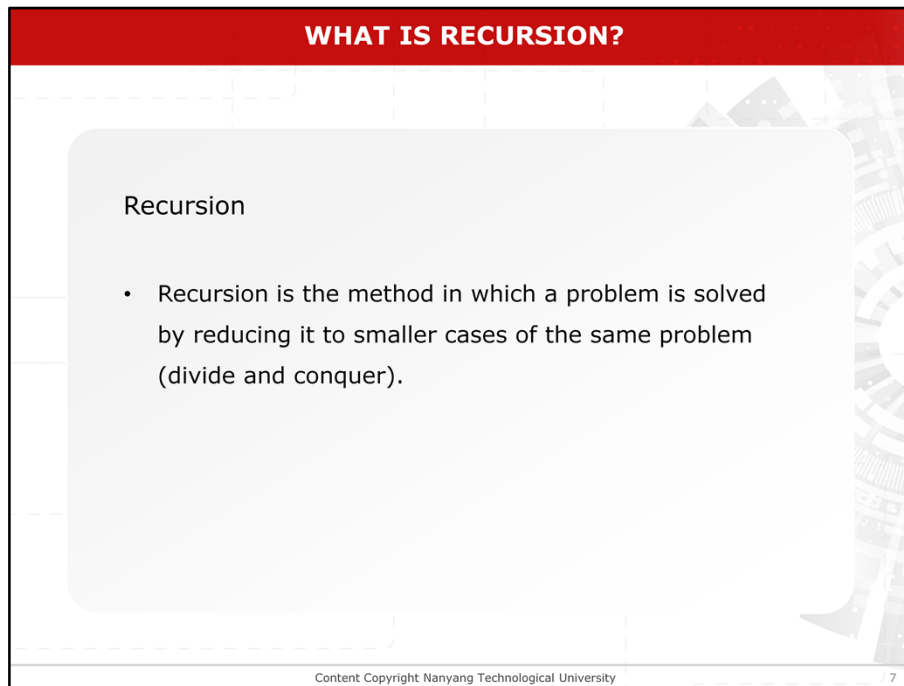
- A function, when called, will accomplish a certain job.
- When a function $F()$ is called, control is transferred from the calling point to the first statement in $F()$. After the function finishes execution, the control will return back to the calling point. The next statement after the function call will be executed.
- Each call to a function has its **own set of values** for the actual arguments and local variables.

Content Copyright Nanyang Technological University

6

Each call to a function has its own set of values for the actual arguments and local variables.

These properties are important for the understanding of *recursive functions*.



WHAT IS RECURSION?

Recursion

- Recursion is the method in which a problem is solved by reducing it to smaller cases of the same problem (divide and conquer).

Content Copyright Nanyang Technological University 7

Recursion is the method in which a problem is solved by reducing it to smaller cases of the same problem (divide and conquer)

WHAT IS RECURSION?

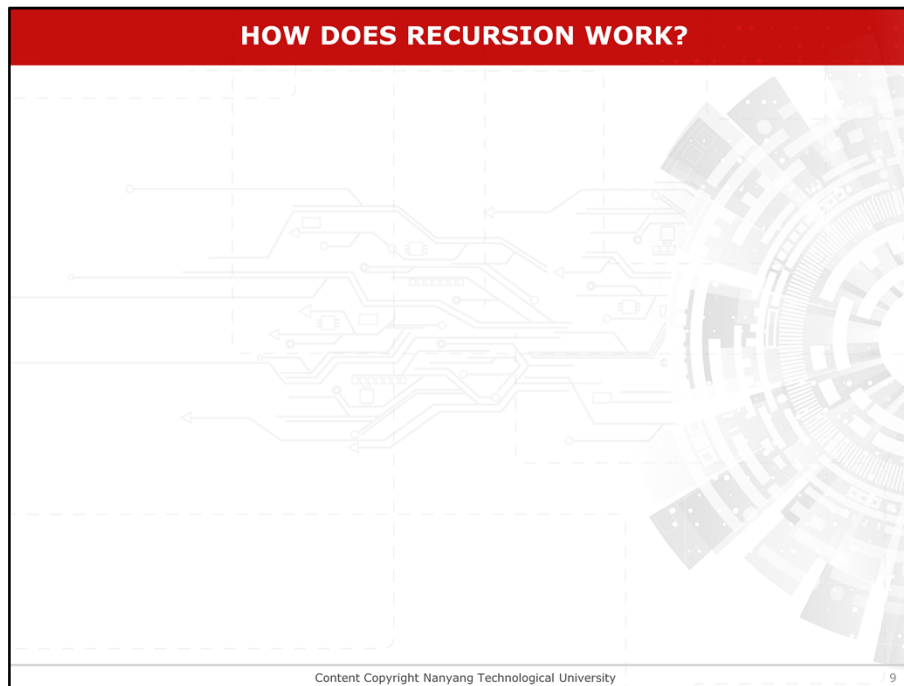
Recursion

- Recursion is the method in which a problem is solved by reducing it to smaller cases of the same problem (divide and conquer).
- In recursion, the function calls itself or calls a sequence of other functions, one of which eventually calls the first function again.

Content Copyright Nanyang Technological University

8

In recursion, the function calls itself or calls a sequence of other functions, one of which eventually calls the first function again.



How does recursion work?

HOW DOES RECURSION WORK?

- Recursive function consists of two parts:
 - Base case (with terminating condition)
 - Recursive case (with recursive condition)

Content Copyright Nanyang Technological University 10

Generally, a recursive function consists of two parts: base case for terminating condition and recursive case for recursive condition.

HOW DOES RECURSION WORK?

- Recursive function consists of two parts:
 - Base case (with terminating condition)
 - Recursive case (with recursive condition)
- A recursive function is a function that calls itself. Each function makes a call to itself with an argument which is closer to the terminating condition.

Content Copyright Nanyang Technological University 11

A recursive function is a function that calls itself. Each function makes a call to itself with an argument which is closer to the terminating condition.

HOW DOES RECURSION WORK?

- Recursive function consists of two parts:
 - Base case (with terminating condition)
 - Recursive case (with recursive condition)
- A recursive function is a function that calls itself. Each function makes a call to itself with an argument which is closer to the terminating condition.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

Here is an example of a recursive function. Notice the difference of the two conditions, where the recursive condition calls itself.

HOW DOES RECURSION WORK?

- When a function calls itself, the execution of the current function is suspended, the information that it needs to continue execution is saved, and the recursive call is then evaluated.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

When a function calls itself, the execution of the current function is suspended, the information that it needs to continue execution is saved, and the recursive call is then evaluated.

HOW DOES RECURSION WORK?

- When a function calls itself, the execution of the current function is suspended, the information that it needs to continue execution is saved, and the recursive call is then evaluated.
- Each call to a function has its own set of values/arguments for the formal arguments and local variables.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

Each call to a function has its own set of values or arguments for the formal arguments and local variables.

HOW DOES RECURSION WORK?

- When the recursive call is evaluated, a new set of variables is created, which is independent from the variables that are created from the previous calls to the function.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

When the recursive call is evaluated, a new set of variables is created, which is independent from the variables that are created from the previous calls to the function.

HOW DOES RECURSION WORK?

- When the recursive call is evaluated, a new set of variables is created, which is independent from the variables that are created from the previous calls to the function.
- When a recursive call is made, control is transferred from the calling point to the first statement of the recursive function.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

When a recursive call is made, control is transferred from the calling point to the first statement of the recursive function.

HOW DOES RECURSION WORK?

- When a call at a certain level is finished, control returns to the calling point one level up.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

When a call at a certain level is finished, control returns to the calling point one level up.

HOW DOES RECURSION WORK?

- When a call at a certain level is finished, control returns to the calling point one level up.
- When the recursive call is completed, the evaluation result is passed back to the previous call, until the process is completed.

Example:

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

Factorial - recursive definition:

- $n! = 1$ if $n = 0$
- $n! = n \times (n-1)!$ if $n > 0$

When the recursive call is completed, the evaluation result is passed back to the previous call, until the process is completed.

SUMMARY

After this lesson, you should be able to:

- Explain recursion and how it works

Content Copyright Nanyang Technological University 19

In summary, after viewing this video lesson, you should be able to explain recursion and how it works.

We will look into examples of recursion and how it works in the next few video lectures.