

This lesson is on Functions.

OVERVIEW

The following are the coverage for Functions:

- Function Definition
- Function Prototypes
- Function Flow
- Parameter Passing: Call by Value
- Storage Scope of Variables
- Functional Decomposition

Content Copyright Nanyang Technological University

2

There are 6 main sections to cover for Functions. This video focused on the 1st topic:
Function definition



Learning objectives

LEARNING OBJECTIVES

At this lesson, you should be able to:

At this lesson, you should be able to:

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define a function

Define a function

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define a function
- Design a program involving functions

Design a program involving functions

OVERVIEW

- With Sequential, Branching and Looping, you will be able to build programs for simple applications.

Content Copyright Nanyang Technological University

7

With Sequential, Branching and Looping, you will be able to build programs for simple applications.

OVERVIEW

- With Sequential, Branching and Looping, you will be able to build programs for simple applications.
- However, for more complex applications, your programs may be long and certain code may be repeated in the programnot so good!

Content Copyright Nanyang Technological University

8

However, for more complex applications, your programs may be long and certain code may be repeated in the programnot so good!

OVERVIEW

- With Sequential, Branching and Looping, you will be able to build programs for simple applications.
- However, for more complex applications, your programs may be long and certain code may be repeated in the programnot so good!
- Functions are used to group specific tasks into functions, so that code will not be repeated. It also helps to improve your program readability and efficiency.

Content Copyright Nanyang Technological University

9

Functions are used to group specific tasks into functions, so that code will not be repeated. It also helps to improve your program readability and efficiency.

FUNCTIONS

- To solve real-world problems, we need larger programs than the ones presented in the previous lectures.

Content Copyright Nanyang Technological University

10

To solve real-world problems, we need larger programs than the ones presented in the previous lectures.

FUNCTIONS

- To solve real-world problems, we need larger programs than the ones presented in the previous lectures.
- When developing large and complex programs, the programming task will be much simplified if we use functions.

Content Copyright Nanyang Technological University

11

When developing large and complex programs, the programming task will be much simplified if we use functions.

FUNCTIONS

- To solve real-world problems, we need larger programs than the ones presented in the previous lectures.
- When developing large and complex programs, the programming task will be much simplified if we use functions.
- Every C program is made up of one or more functions.

Content Copyright Nanyang Technological University

12

Every C program is made up of one or more functions.

FUNCTIONS

- To solve real-world problems, we need larger programs than the ones presented in the previous lectures.
- When developing large and complex programs, the programming task will be much simplified if we use functions.
- Every C program is made up of one or more functions.
- A function is a self-contained unit of code to carry out a specific task.

Content Copyright Nanyang Technological University

13

A function is a self-contained unit of code to carry out a specific task.

FUNCTIONS

- Any C functions can call any of the other functions in a program.

Any C functions can call any of the other functions in a program.

FUNCTIONS

- Any C functions can call any of the other functions in a program.
- One of the functions must be named as main(). Program execution begins with main() and terminates when the last statement of the main() function has been executed.

Content Copyright Nanyang Technological University

15

- One of the functions must be named as main(). Program execution begins with main() and terminates when the last statement of the main() function has been executed.

FUNCTIONS

- Any C functions can call any of the other functions in a program.
- One of the functions must be named as main(). Program execution begins with main() and terminates when the last statement of the main() function has been executed.
- In this lecture, we discuss the concepts of functions.

Content Copyright Nanyang Technological University

16

In this lecture, we discuss the concepts of functions.

FUNCTIONS DEFINITION

- A **function** is a self-contained unit of code to carry out a specific task, e.g. `printf()`, `sqrt()`.

Content Copyright Nanyang Technological University 17

Function Definition

A function is a piece of code, which specifies the actions of the function, and the local data used by the function.

The diagram is titled "FUNCTIONS DEFINITION" in a red header bar. Below the title, it says "A **function** consists of:" followed by a bulleted list: "• A header". To the right of the list is a yellow box labeled "Function header". The background features a faint watermark of a building.

A function consists of:

- A header

Function header

Content Copyright Nanyang Technological University 18

A function definition has the following structure: **function_header**

FUNCTIONS DEFINITION

A **function** consists of:

- A header
- An opening curly brace

The diagram illustrates the components of a function definition. It features a red header bar with the text "FUNCTIONS DEFINITION". Below this, a white area contains the explanatory text. To the right, a yellow box labeled "Function header" is positioned above an orange rectangular area. Within the orange area, a blue callout box labeled "Open Brace" points to an opening curly brace character ("{"). The background of the slide has a faint watermark of a gear.

Content Copyright Nanyang Technological University

19

an opening curly brace

FUNCTIONS DEFINITION

A **function** consists of:

- A header
- An opening curly brace
- A function body

The diagram illustrates the structure of a function definition. It features a yellow rectangular box at the top labeled "Function header". Below it is an orange rectangular box labeled "Function body". A blue rounded rectangle labeled "Open Brace" contains a black curly brace character. An arrow points from the text "Open Brace" to the brace character. The entire diagram is set against a background with faint architectural drawings.

Content Copyright Nanyang Technological University 20

a function body

FUNCTIONS DEFINITION

A **function** consists of:

- A header
- An opening curly brace
- A function body

The diagram illustrates the structure of a function definition. It features a large orange rectangular box representing the function body. At the top of this box, a yellow header box contains the text "Function header". Inside the orange box, there is an "Open Brace" symbol (an opening curly brace) positioned above the text "Function body". Below the "Function body" text is a "Close brace" symbol (a closing curly brace). Arrows point from the labels "Function header", "Open Brace", "Function body", and "Close brace" to their respective parts in the diagram.

Content Copyright Nanyang Technological University 21

And a closely curly brace

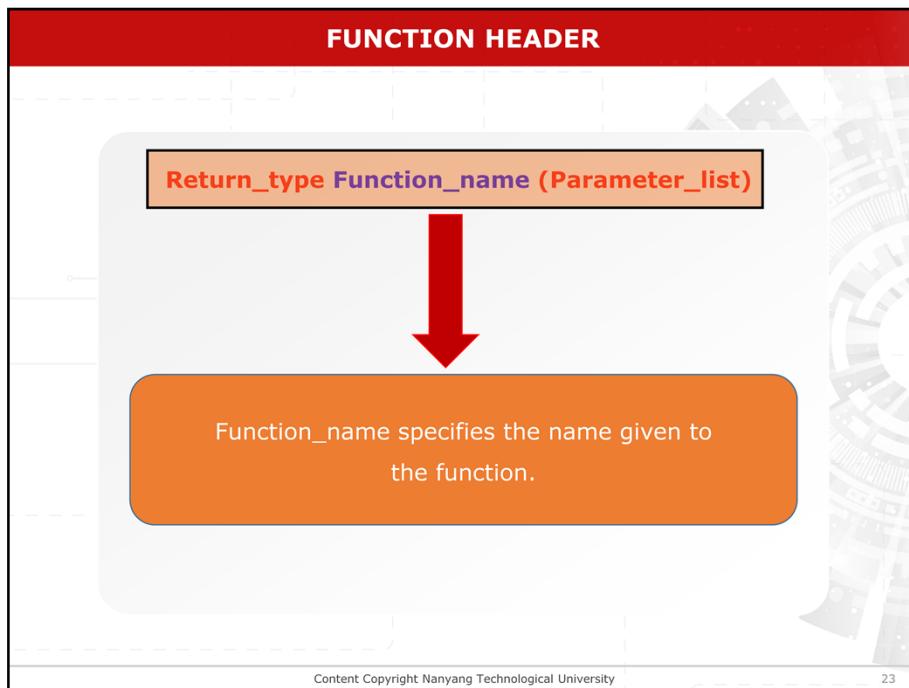
The slide has a red header bar with the word "EXAMPLE" in white capital letters. Below the header is a code block with a light orange background. The code defines a function named `findMax` that takes two float parameters, `x` and `y`. It initializes a variable `maxnum` to the value of `x`, then compares `x` and `y` using an if-else statement. If `x` is greater than or equal to `y`, it assigns `x` to `maxnum`; otherwise, it assigns `y` to `maxnum`. Finally, it returns the value of `maxnum`. The footer of the slide contains the text "Content Copyright Nanyang Technological University" and the number "22".

```
float findMax(float x, float y) // header
{
    // function body
    float maxnum;

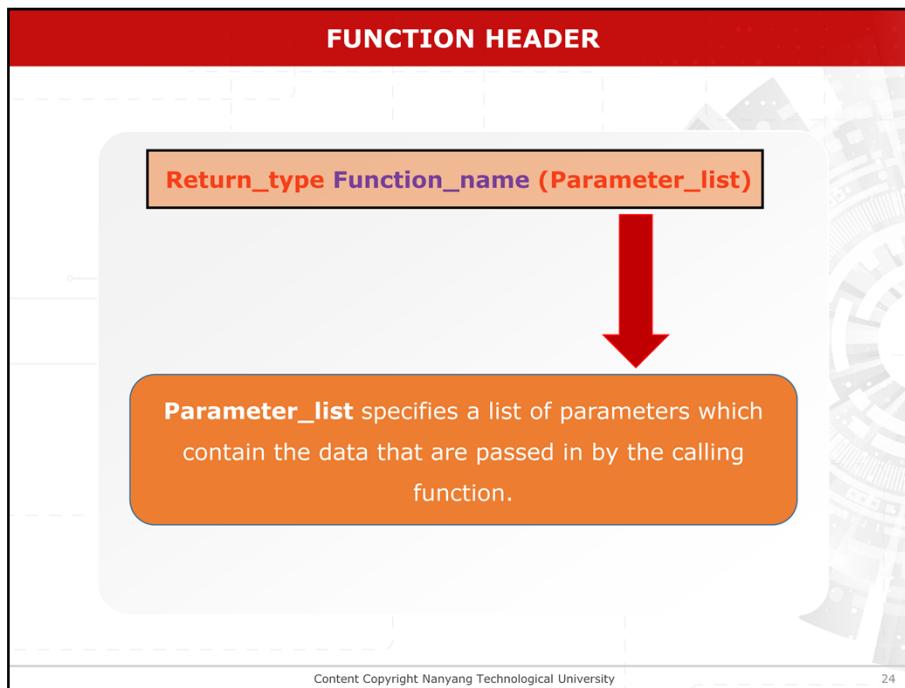
    if (x >= y)
        maxnum = x;
    else
        maxnum = y;

    return maxnum;
}
```

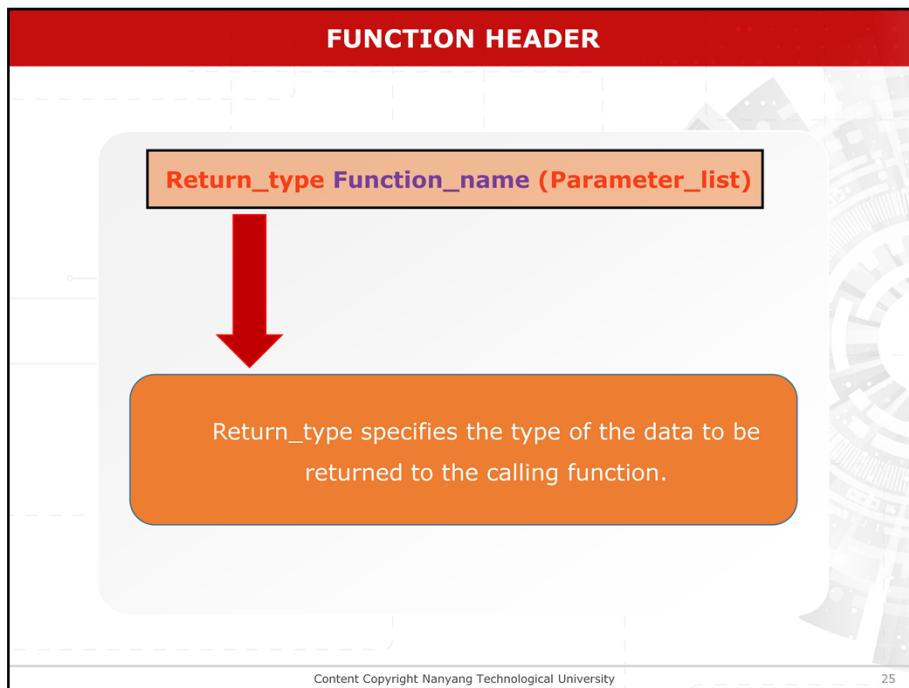
An example is illustrated in the `findMax()` function, which has the function header and function body.



The *function header* has the format as shown. **Function name** is the name given to the function.



Parameter list specifies a list of parameters which contain the data that are passed in by the calling function.



Return type specifies the type of the data to be returned to the calling function.

FUNCTION HEADER: PARAMETER LIST

- Parameters define the data passed into the function.

Content Copyright Nanyang Technological University

26

Parameters define the **data passed into** the function.

FUNCTION HEADER: PARAMETER LIST

- Parameters define the data passed into the function.
- A function can have no parameter, one parameter or many parameters.

Content Copyright Nanyang Technological University

27

A function can have no parameter, one parameter or many parameters.

FUNCTION HEADER: PARAMETER LIST

- Parameters define the data passed into the function.
- A function can have no parameter, one parameter or many parameters.

type parameterName[, type parameterName]

Example: float **findMaximum**(float x, float y)

Content Copyright Nanyang Technological University 28

The format is shown here, where parenthesis can be repeated zero or more times.

FUNCTION HEADER: PARAMETER LIST

- Parameters define the data passed into the function.
- A function can have no parameter, one parameter or many parameters.

type parameterName[, type parameterName]

Example: float **findMaximum**(float x, float y)

Each parameter has:

- **Parameter name**
- **Data type** (e.g. int, char, etc.) of the parameter

Content Copyright Nanyang Technological University 29

The parameters are known only inside the function body. Each parameter has: **parameter name** and **data type** (eg int, car, etc)of the parameter

FUNCTION HEADER: RETURN_TYPE

Return_type is the data type returned from the function, it can be int, float, char, void, or nothing.

Content Copyright Nanyang Technological University 30

Function Header: Return_type

Return_type is the data type of the value returned by the function, it can be int, float, char, void, or nothing.

FUNCTION HEADER: RETURN_TYPE

Return_type is the data type returned from the function, it can be int, float, char, void, or nothing.

- int -- the function will return a value of the type int.

Content Copyright Nanyang Technological University 31

Int , the function will return a value of the type int

FUNCTION HEADER: RETURN_TYPE

Return_type is the data type returned from the function, it can be int, float, char, void, or nothing.

- int** -- the function will return a value of the type int.
- float** -- the function will return a value of the type float.

Content Copyright Nanyang Technological University 32

Float, the function will return a value of the type float

FUNCTION HEADER: RETURN_TYPE

Return_type is the data type returned from the function, it can be int, float, char, void, or nothing.

- int** -- the function will return a value of the type int.
- float** -- the function will return a value of the type float.
- void** -- the function will not return any value.

Content Copyright Nanyang Technological University

33

Void, the function will not return any value

FUNCTION HEADER: RETURN_TYPE

Return_type is the data type returned from the function, it can be int, float, char, void, or nothing.

- int** -- the function will return a value of the type int.
- float** -- the function will return a value of the type float.
- void** -- the function will not return any value.

```
return (expression);
```

Content Copyright Nanyang Technological University 34

The syntax for the **return** statement is
return (expression);

FUNCTION HEADER: RETURN_TYPE

```
int successor(int num) /* function header */  
{    /* function body begins here */  
    return num + 1;  
}    /* function body ends here */
```

The function **successor()** will return a value of type **int**.

Content Copyright Nanyang Technological University 35

In the example shown, the function **successor()** will return a value of type **int**.

FUNCTION HEADER: RETURN_TYPE

```

int successor(int num) /* function header */
{
    /* function body begins here */
    return num + 1;
} /* function body ends here */

return (expression);

```

Content Copyright Nanyang Technological University 36

In the example shown, the function **successor()** will return a value of type **int**. For the return type **void**, the function will not return any value. The function **hello_n_times()**

```

void hello_n_times(int n)
{
    int count;
    for (count = 0; count < n; count++)
        printf("Hello\n");
    /* no return statement here */
}

```

prints a string "Hello" to the screen the number of times specified by the parameter **n**, which is defined to be of type **int**.

If nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.

FUNCTION HEADER: RETURN_TYPE

```
void hello_n_times(int n)
{
    int count;
    for (count = 0; count < n; count++)
        printf("Hello\n");
    /* no return statement */
}
```

For the return type **void**, the function will not return any value.

Content Copyright Nanyang Technological University 37

For the return type **void**, the function will not return any value.

FUNCTION HEADER: RETURN_TYPE

```
void hello_n_times(int n)
{
    int count;
    for (count = 0; count < n; count++)
        printf("Hello\n");
    /* no return statement */
}
```

The function **hello_n_times()** prints a string "Hello" to the screen the number of times specified by the parameter **n**, which is defined to be of type **int**.

Content Copyright Nanyang Technological University 38

The function **hello_n_times()** prints a string "Hello" to the screen the number of times specified by the parameter **n**, which is defined to be of type **int**.

If nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.

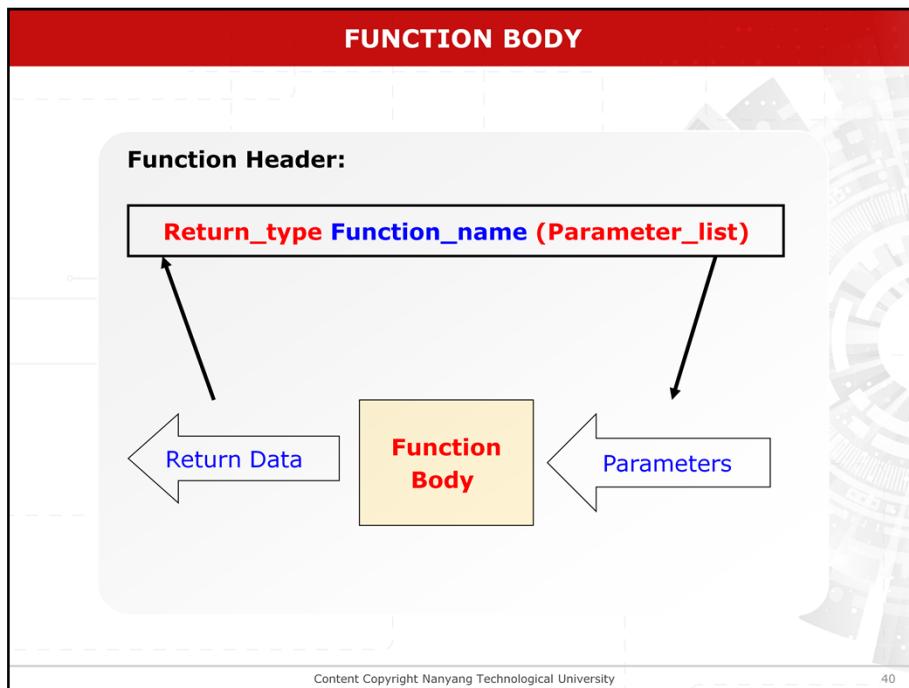
FUNCTION HEADER: RETURN_TYPE

```
void hello_n_times(int n)
{
    int count;
    for (count = 0; count < n; count++)
        printf("Hello\n");
    /* no return statement */
}
```

Note that if nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.

Content Copyright Nanyang Technological University 39

If nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.



The function header and function body are shown here

FUNCTION BODY

- The function body consists of a group of *statements*, it can be declaration statements, simple statements or compound statements.

Content Copyright Nanyang Technological University 41

Function Body

The function body consists of a group of *statements*, it can be declaration statements, simple statements or compound statements.

FUNCTION BODY

- The function body consists of a group of *statements*, it can be declaration statements, simple statements or compound statements.
- The statements are executed when the function is called. The variables declared inside the function body are called *local* variables and are only known within the function.

Content Copyright Nanyang Technological University 42

The statements are executed when the function is called. The variables declared inside the function body are called *local* variables and are only known within the function.

FUNCTION BODY

- The function body consists of a group of *statements*, it can be declaration statements, simple statements or compound statements.
- The statements are executed when the function is called. The variables declared inside the function body are called *local* variables and are only known within the function.
- It's main purpose is to perform the logic of the function.

Content Copyright Nanyang Technological University 43

It's main purpose is to perform the logic of the function.

The diagram illustrates a function definition. At the top, a red bar labeled "FUNCTION BODY" is visible. Below it, a code block shows the implementation of a function:

```
double distance(double x, double y)
{
    return sqrt(x * x + y * y);
}
```

A callout box, highlighted with an orange border, contains the following text:

The function is defined with two parameters, **x** and **y**, of data type **double**. It returns a value of type **double**.

At the bottom of the slide, there is a footer with the text "Content Copyright Nanyang Technological University" and the number "44".

For example, in the **distance()** function shown here, the function is defined with two parameters, **x** and **y**, of data type **double**. It returns a value of type **double**.

The diagram illustrates a function body with a red header bar labeled "FUNCTION BODY". Below it is a code block containing:

```
double distance(double x, double y)
{
    return sqrt(x * x + y * y);
}
```

A callout box below the code states: "The function body consists of a statement to compute the distance of the two values."

Content Copyright Nanyang Technological University 45

The function body consists of a statement to compute the distance of the two values.

The diagram illustrates a function body. At the top, a red bar contains the text "FUNCTION BODY". Below it, a white area contains the C code for a function named "hello". The parameter list "void" is highlighted with a red border. The code is as follows:

```
void hello(void)
{
    printf("Hello\n");
}
```

Below the code, an orange box contains a note: "The parameter_list is void, which means that no data will be passed into this function." At the bottom of the slide, there is a footer with the text "Content Copyright Nanyang Technological University" and the number "46".

In the `hello()` function shown here, the **parameter_list** is **void**, which means that no data will be passed into this function.

The diagram illustrates a function body. At the top, a red bar contains the text "FUNCTION BODY". Below it, a code block shows:

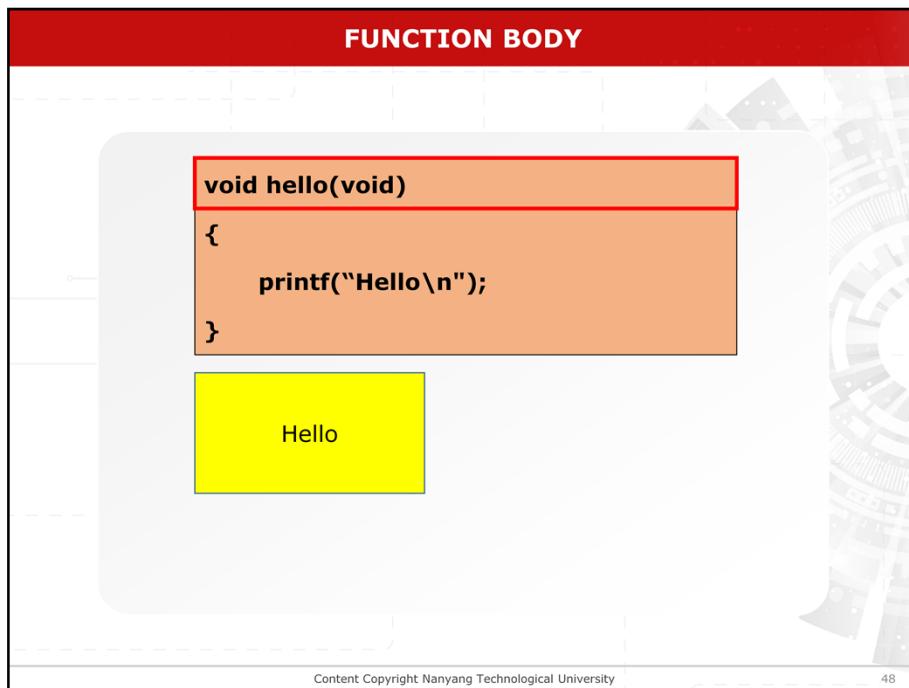
```
void hello(void)
{
    printf("Hello\n");
}
```

A callout box points to the word "hello" in the code, which is highlighted with a red border. Below the code, an orange box contains the following text:

In addition, the function will not return any value to the calling function.

At the bottom of the slide, there is a footer with the text "Content Copyright Nanyang Technological University" and the number "47".

In addition, the function will not return any value to the calling function. The function body just prints out the message **hello** on the screen.



The function body just prints out the message **hello** on the screen.

MULTIPLE RETURN STATEMENTS

- The return statement may appear in any place and in more than one place inside the function body.

Content Copyright Nanyang Technological University

49

Multiple Return Statements

The **return** statement may appear in any place and in more than one place inside the function body.

MULTIPLE RETURN STATEMENTS

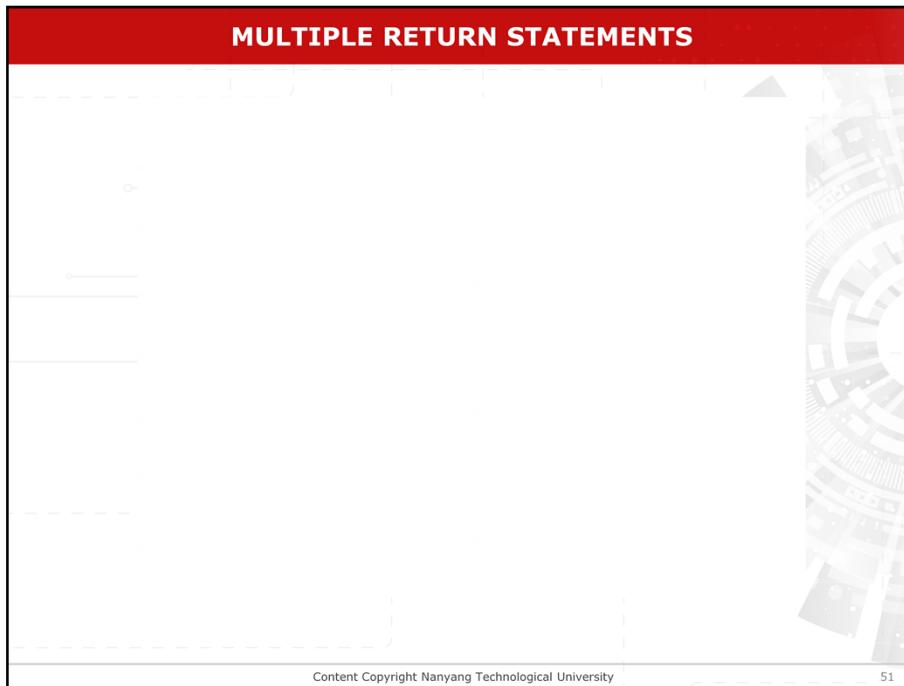
- The return statement may appear in any place and in more than one place inside the function body.
- Also, the **return** statement terminates the execution of the function and passes the control back to the calling function.

Content Copyright Nanyang Technological University

50

Multiple Return Statements

The **return** statement may appear in any place and in more than one place inside the function body.



Multiple Return Statements

The **return** statement may appear in any place and in more than one place inside the function body. Also, the **return** statement terminates the execution of the function and passes the control back to the calling function. In the program, the function **fact()** has **return** statements in various locations in the function body. If **n** is less than 0, then an error message is displayed, and the function returns the value 0 to the calling function . If **n** equals to 0, then the function returns a value 1. If **n** is greater than 0, then the factorial of **n** is evaluated using a **for** loop, and the result is returned.

The value returned by the function can be assigned to a variable or used in other functions. In the program, the **main()** function calls the function **fact()**. The value 24 returned by the function **fact()** is then assigned to the variable **num**.

MULTIPLE RETURN STATEMENTS

```
void hello_n_times(int n)
{
    int count;
    if (n <= 0)
        return; /* return control to the calling function */
    else
        for (count = 0; count < n; count++)
            printf("Hello!\n");
}
```

Content Copyright Nanyang Technological University 52

A type **void** function may also have a **return** statement to terminate the function. However, it must not have a **return** expression.

If the function does not have a **return** statement, then the control will be passed back to the calling function when the closing brace of the function body is encountered.

Sometimes, it is not necessary to use the value returned by a function. This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**. The **printf()** statement returns a value of type **int** that counts the number of characters printed, whereas the **scanf()** statement returns the number of items that are successfully read. However, if we do not require this information, we do not need to use the return value returned by these functions.

MULTIPLE RETURN STATEMENTS

- Sometimes, it is not necessary to use the value returned by a function.

Content Copyright Nanyang Technological University

53

Sometimes, it is not necessary to use the value returned by a function.

MULTIPLE RETURN STATEMENTS

- Sometimes, it is not necessary to use the value returned by a function.
- This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**.

Content Copyright Nanyang Technological University

54

This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**.

MULTIPLE RETURN STATEMENTS

- Sometimes, it is not necessary to use the value returned by a function.
- This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**.
- The **printf()** statement returns a value of type **int** that counts the number of characters printed, whereas the **scanf()** statement returns the number of items that are successfully read.

Content Copyright Nanyang Technological University

55

The **printf()** statement returns a value of type **int** that counts the number of characters printed, whereas the **scanf()** statement returns the number of items that are successfully read.

MULTIPLE RETURN STATEMENTS

- Sometimes, it is not necessary to use the value returned by a function.
- This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**.
- The **printf()** statement returns a value of type **int** that counts the number of characters printed, whereas the **scanf()** statement returns the number of items that are successfully read.
- However, if we do not require this information, we do not need to use the value returned by these functions.

Content Copyright Nanyang Technological University

56

However, if we do not require this information, we do not need to use the value returned by these functions.

FUNCTION: EXAMPLES

Compute Grade:

```
char findGrade(float marks) {
    char grade; // variable

    /* function body */
    if (marks >= 50)
        grade = 'P';
    else
        grade = 'F';
    return grade;
}
```

Content Copyright Nanyang Technological University 57

Function: **findGrade()**

The function **findGrade()** expects a parameter of type **float**

FUNCTION: EXAMPLES

Compute Grade:

```
char findGrade(float marks) {
    char grade; // variable

    /* function body */
    if (marks >= 50)
        grade = 'P';
    else
        grade = 'F';
    return grade;
}
```

Content Copyright Nanyang Technological University 58

and returns a value of type **char**.

FUNCTION: EXAMPLES

Compute Grade:

```
char findGrade(float marks) {
    char grade; // variable

    /* function body */
    if (marks >= 50)
        grade = 'P';
    else
        grade = 'F';
    return grade;
}
```

Content Copyright Nanyang Technological University 59

There is one variable defined in the function. The variable is a local variable and can only be accessed within the function.

FUNCTION: EXAMPLES

Compute Grade:

```
char findGrade(float marks) {
    char grade; // variable

    /* function body */
    if (marks >= 50)
        grade = 'P';
    else
        grade = 'F';
    return grade;
}
```

Content Copyright Nanyang Technological University 60

The variable is created when the function is called, and destroyed when the function ends.

The slide has a red header bar with the text 'FUNCTION: EXAMPLES'. Below the header, there is a section titled 'Compute Circle Area:' containing the following C-like pseudocode:

```
float areaOfCircle(float radius) {
    const float pi = 3.14;
    float area;

    /* function body */
    area = pi*radius*radius;
    return area;
}
```

At the bottom of the slide, there is a footer bar with the text 'Content Copyright Nanyang Technological University' and the number '61'.

Another example to compute circle area is shown here. The function **areaOfCircle()** expects one parameter of type **float**.

The slide has a red header bar with the text 'FUNCTION: EXAMPLES'. Below the header, there is a section titled 'Compute Circle Area:' containing the following code:

```
float areaOfCircle(float radius) {  
    const float pi = 3.14;  
    float area;  
  
    /* function body */  
    area = pi*radius*radius;  
    return area;  
}
```

At the bottom left of the slide, it says 'Content Copyright Nanyang Technological University' and at the bottom right it says '62'.

It returns a value of type **float**.

FUNCTION: EXAMPLES

Compute Circle Area:

```
float areaOfCircle(float radius) {  
    const float pi = 3.14;  
    float area;  
  
    /* function body */  
    area = pi*radius*radius;  
    return area;  
}
```

Content Copyright Nanyang Technological University 63

A local variable **area** is also declared in the function

FUNCTION: EXAMPLES

Compute Circle Area:

```
float areaOfCircle(float radius) {  
    const float pi = 3.14;  
    float area;  
  
    /* function body */  
    area = pi*radius*radius;  
    return area;  
}
```

Content Copyright Nanyang Technological University 64

This variable is only accessible within the function **areaOfCircle()**. It is also created when the function is called, and will be destroyed when the function exits.

SUMMARY

After this lesson, you should be able to:

- Define a function
- Design a program involving functions

Content Copyright Nanyang Technological University

65

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