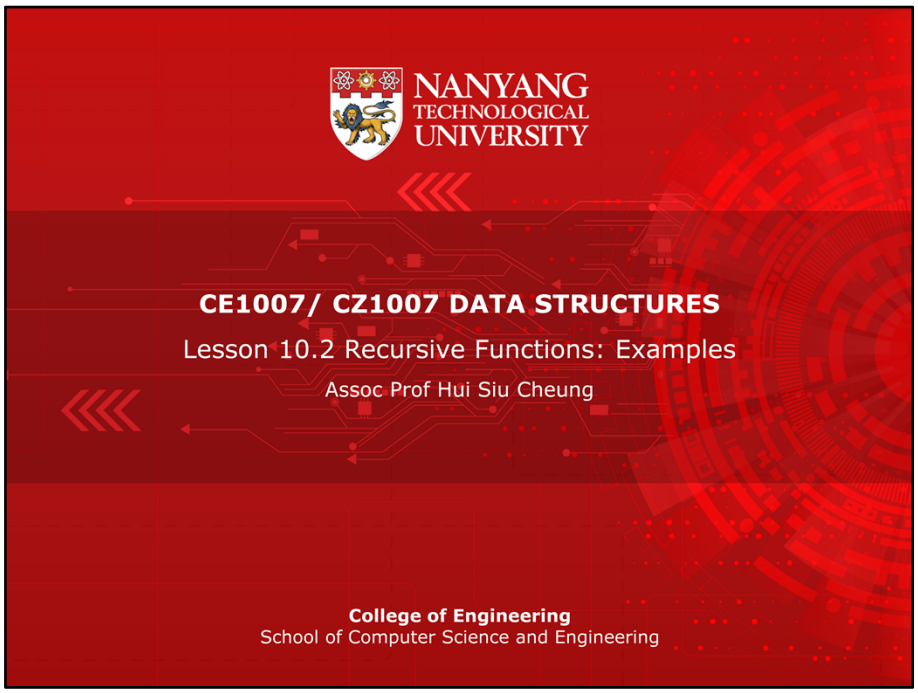
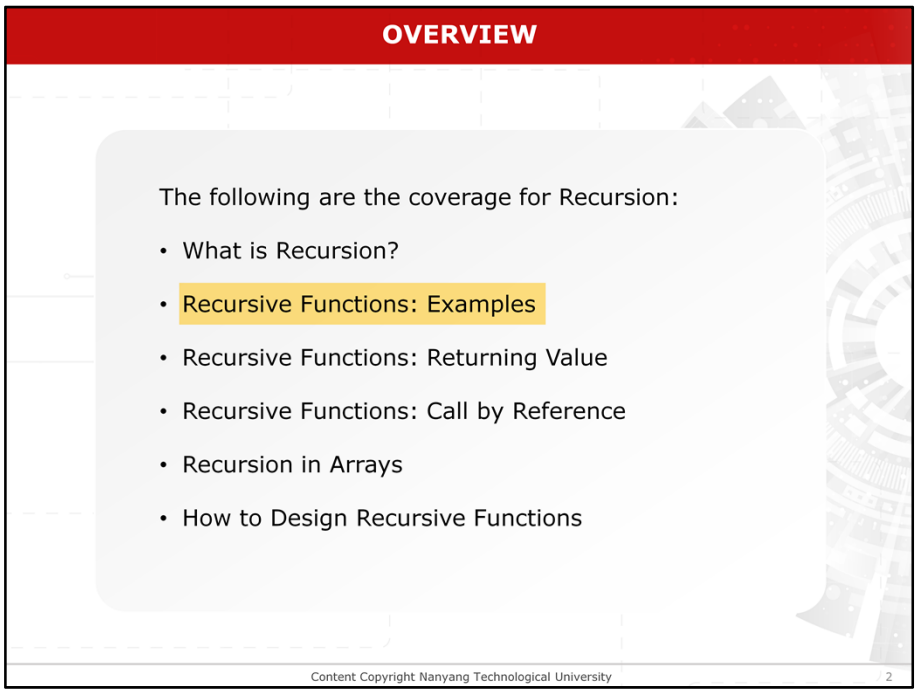
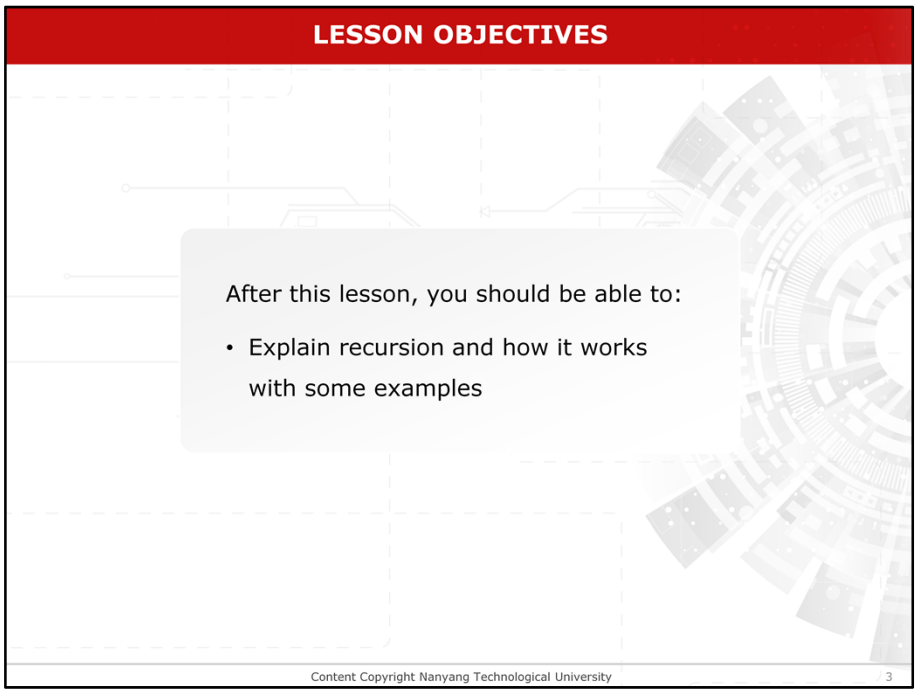
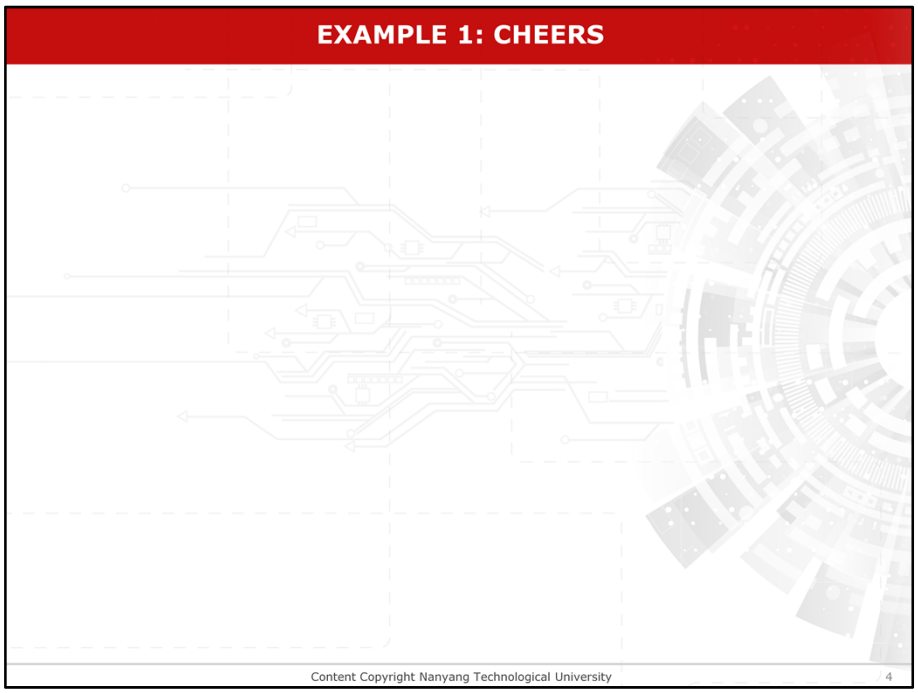

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 10.2 Recursive Functions: Examples
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering




There are 6 main sections to cover for Recursion as shown. This video lesson focuses on the second part where we will look into some examples of recursive functions.




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



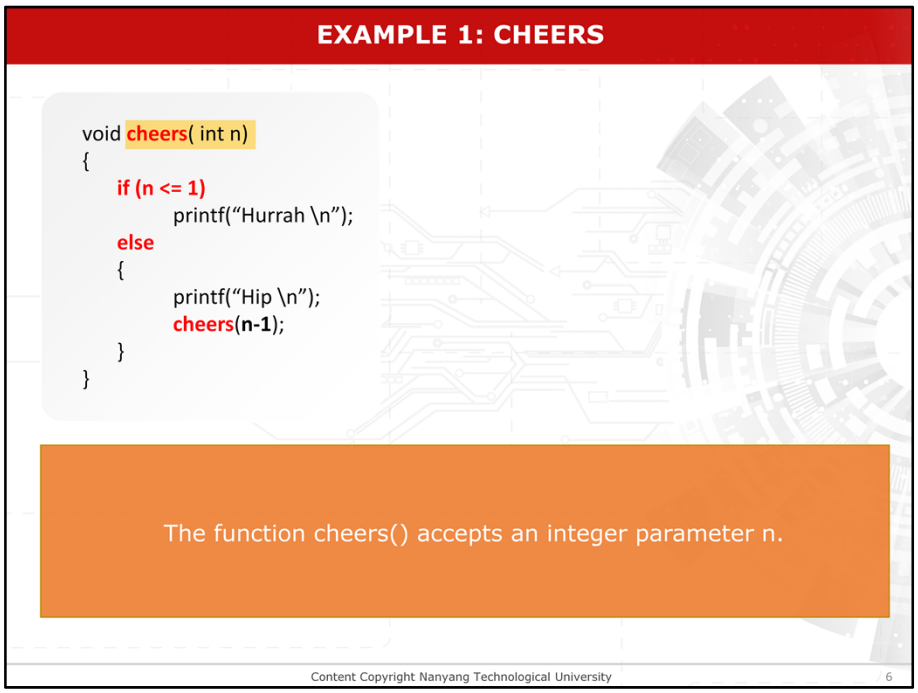

Example 1: Cheers

## EXAMPLE 1: CHEERS

```
void cheers( int n)
{
    if (n <= 1)
        printf("Hurrah \n");
    else
    {
        printf("Hip \n");
        cheers(n-1);
    }
}
```

Content Copyright Nanyang Technological University


The recursive function called **cheers()** is implemented as shown.



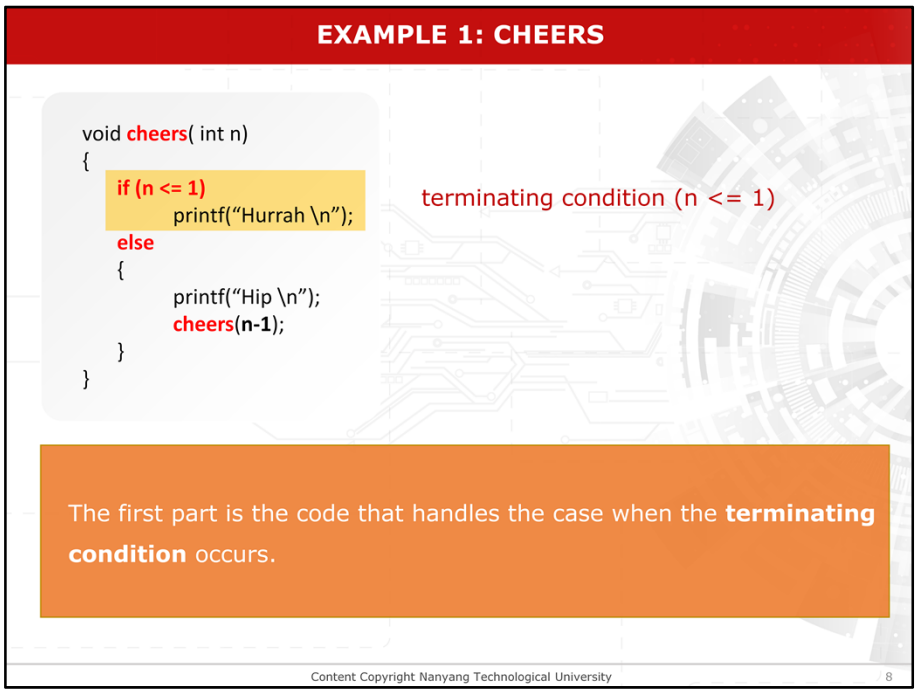

The function **cheers()** accepts an integer parameter **n**.

## EXAMPLE 1: CHEERS

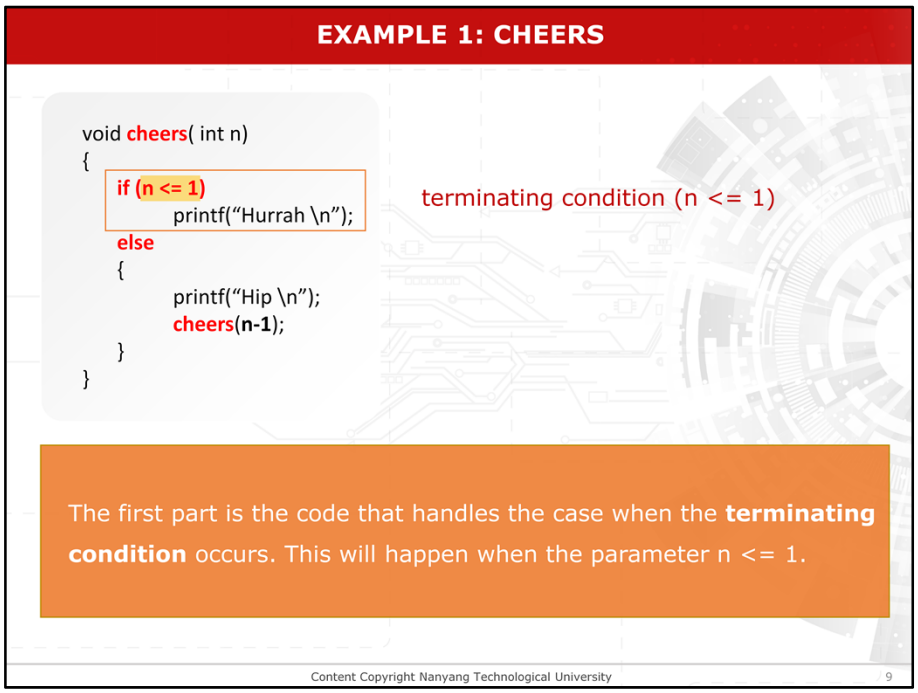
```
void cheers( int n)
{
    if (n <= 1)
        printf("Hurrah \n");
    else
    {
        printf("Hip \n");
        cheers(n-1);
    }
}
```

Content Copyright Nanyang Technological University

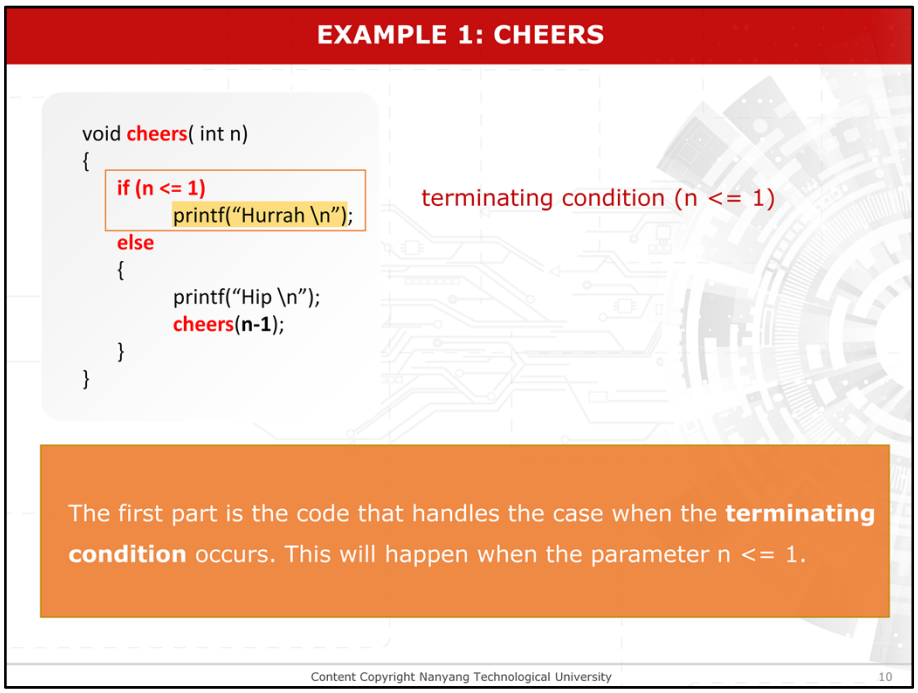

There are two parts of coding in the function.



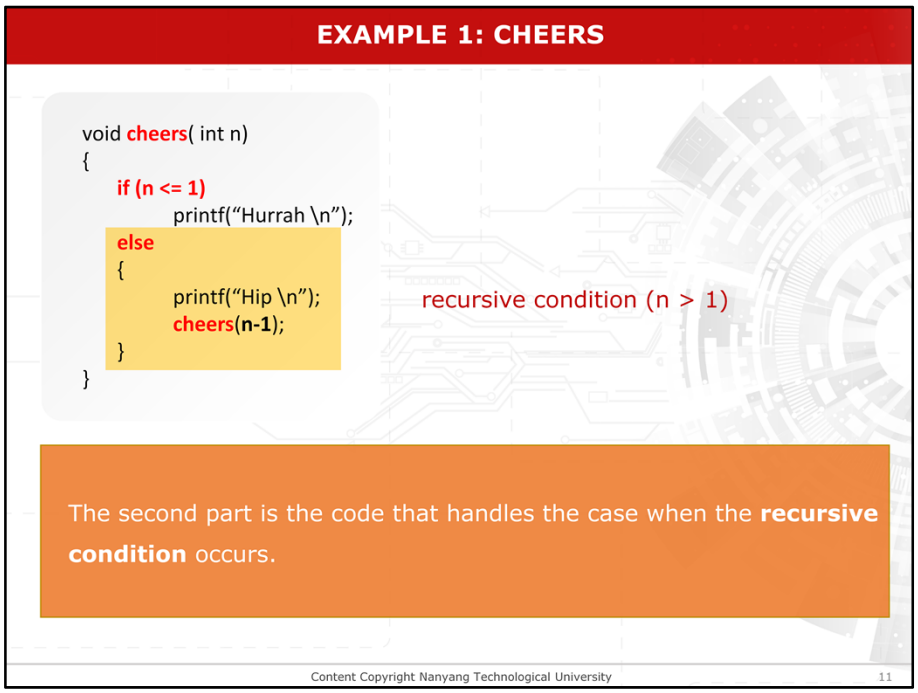

The first part is the code that handles the case when the terminating condition occurs.



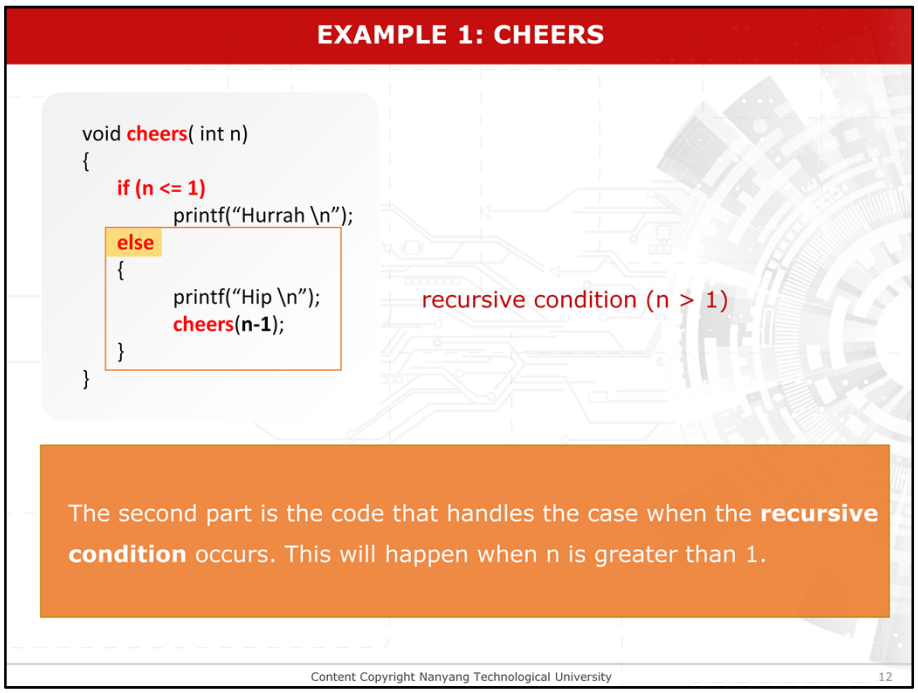

This will happen when the parameter n is lesser than or equal to 1.



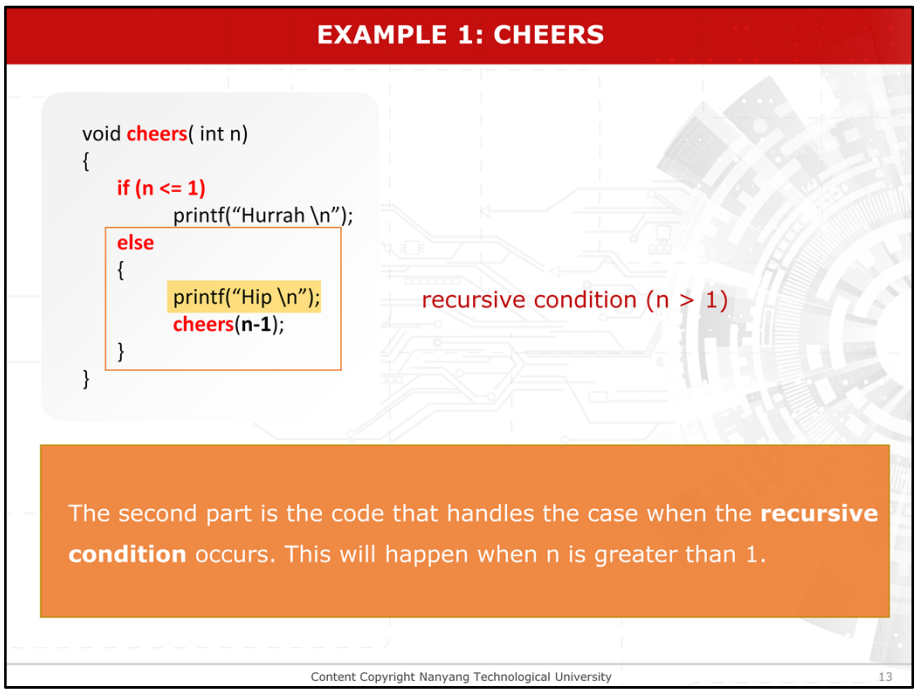

In this case, the character string "Hurrah" will be printed on the screen.



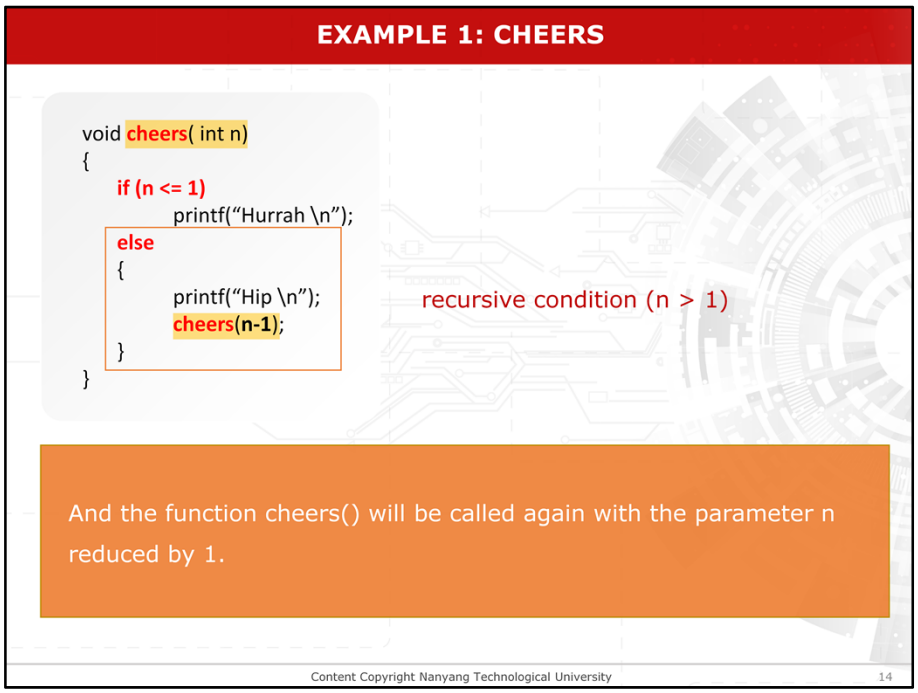

The second part is the code that handles the case when the recursive condition occurs.




This will happen when n is greater than 1.




In this case, the character string **"Hip"** will be printed on the screen.




And the function cheers() will be called again with the parameter n reduced by 1.

## EXAMPLE 1: CHEERS

```
void cheers( int n)
{
    if (n <= 1)
        printf("Hurrah \n");
    else
    {
        printf("Hip \n");
        cheers(n-1);
    }
}
```

terminating condition (n <= 1)

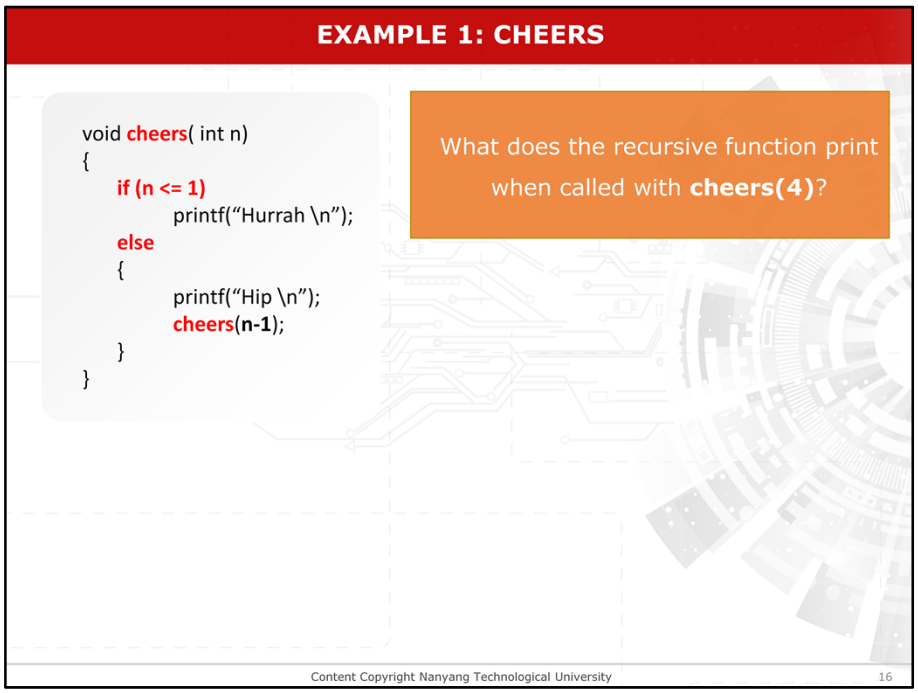
recursive condition (n > 1)

recursive function with updated parameter getting closer to terminating condition

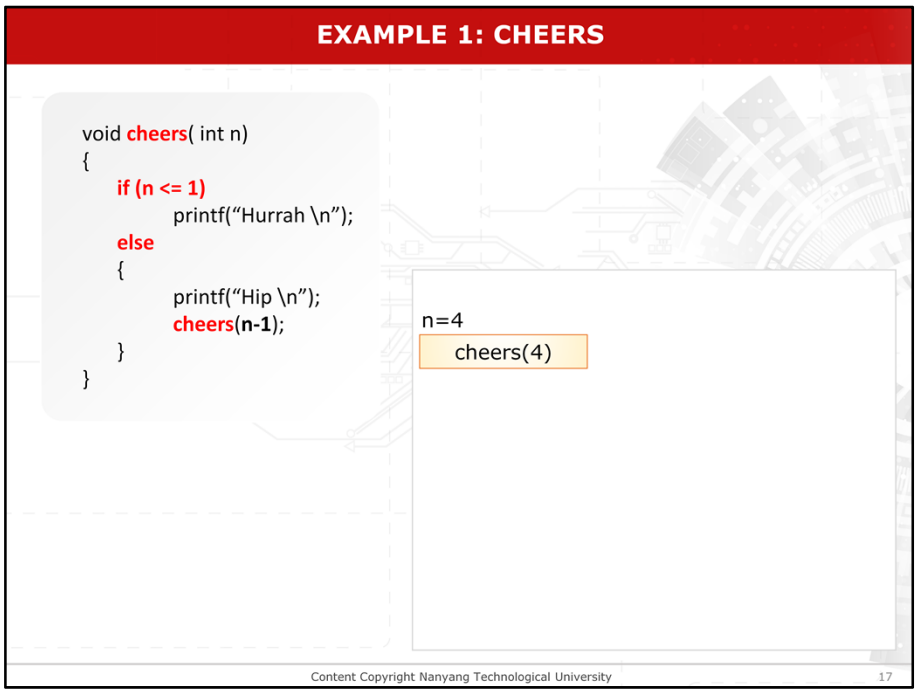
Note that the recursive condition is implemented to ensure that the recursive function will be called again with the updated parameter getting closer to the terminating condition.

Content Copyright Nanyang Technological University

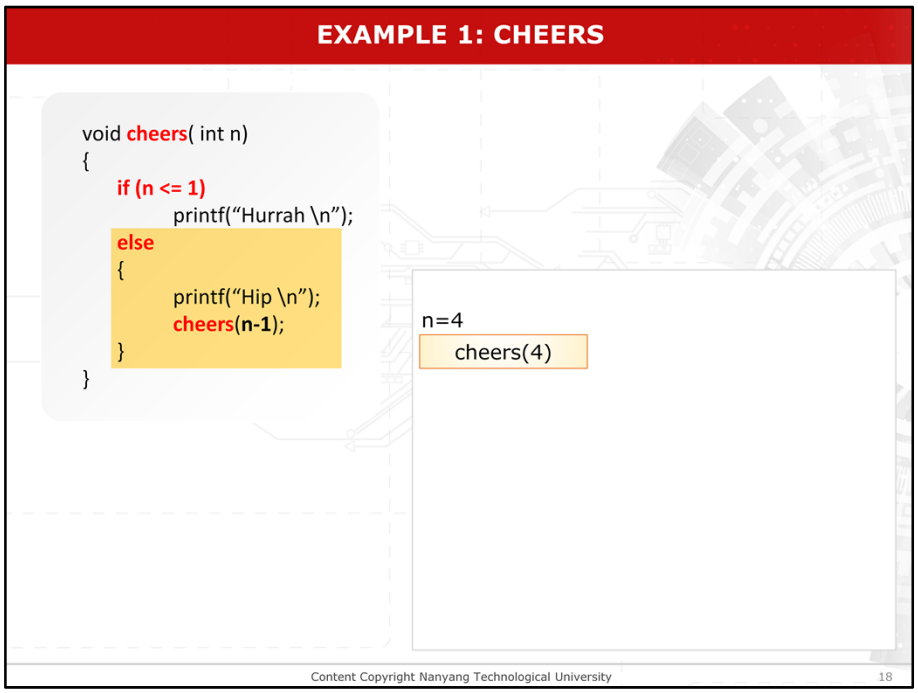
Note that the recursive condition is implemented to ensure that the recursive function will be called again with the updated parameter getting closer to the terminating condition.



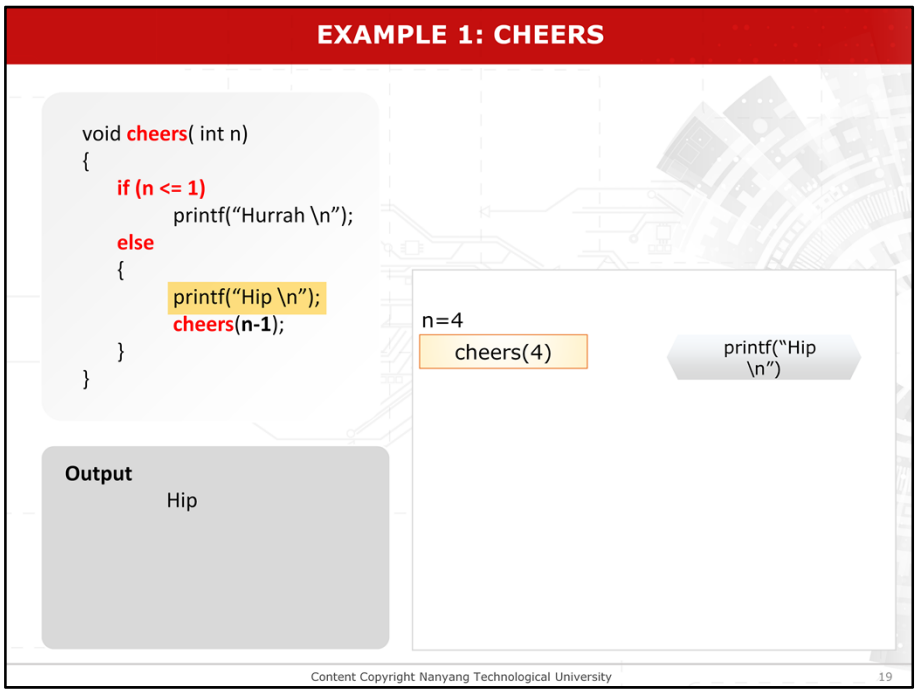

The program tracing for **cheers (4)** is illustrated as follows.



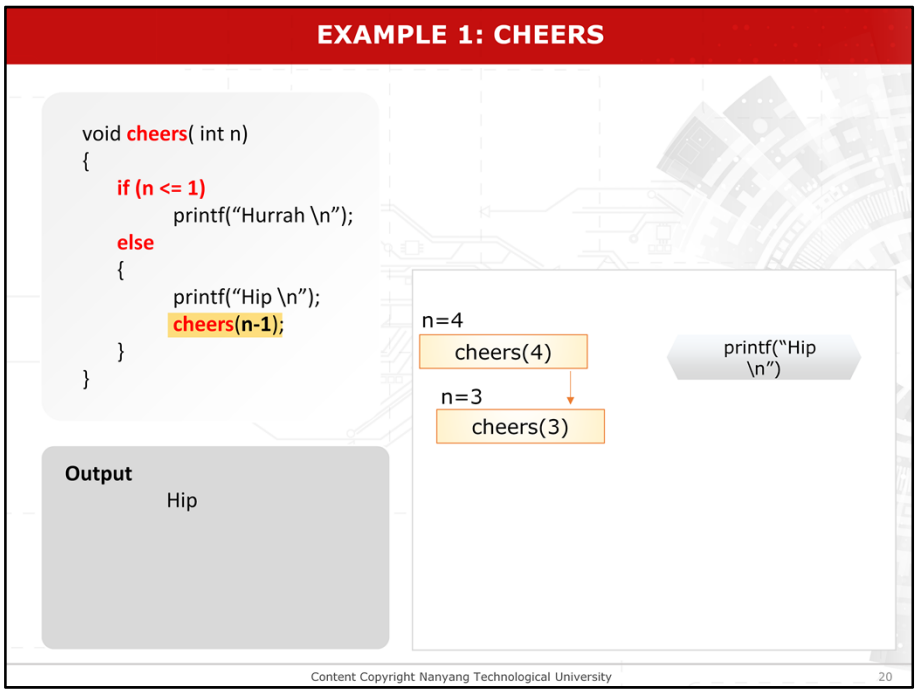

When the parameter **n** is equal to 4,



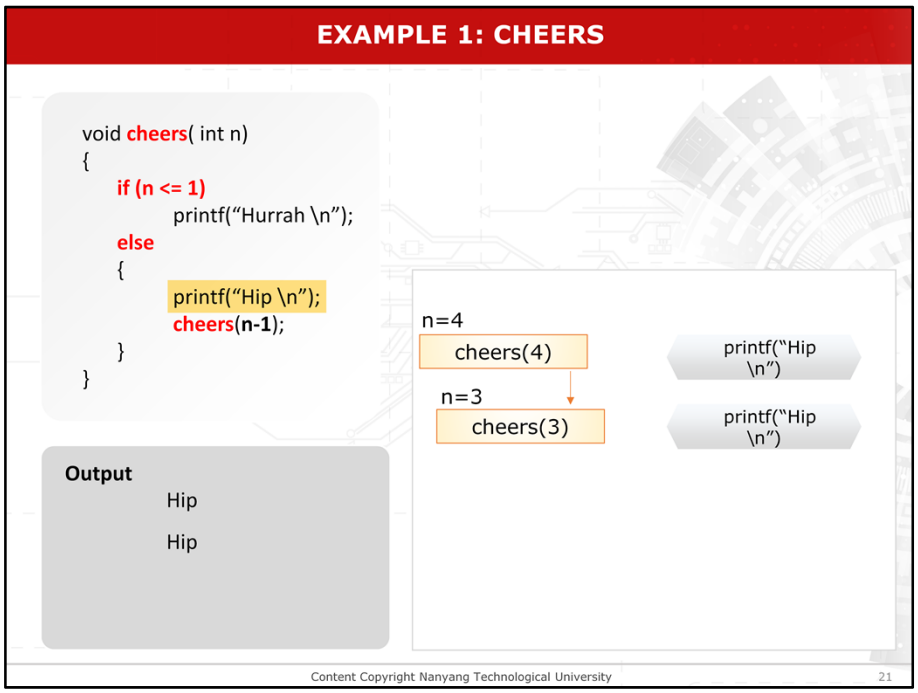

the code for recursive condition is then executed.



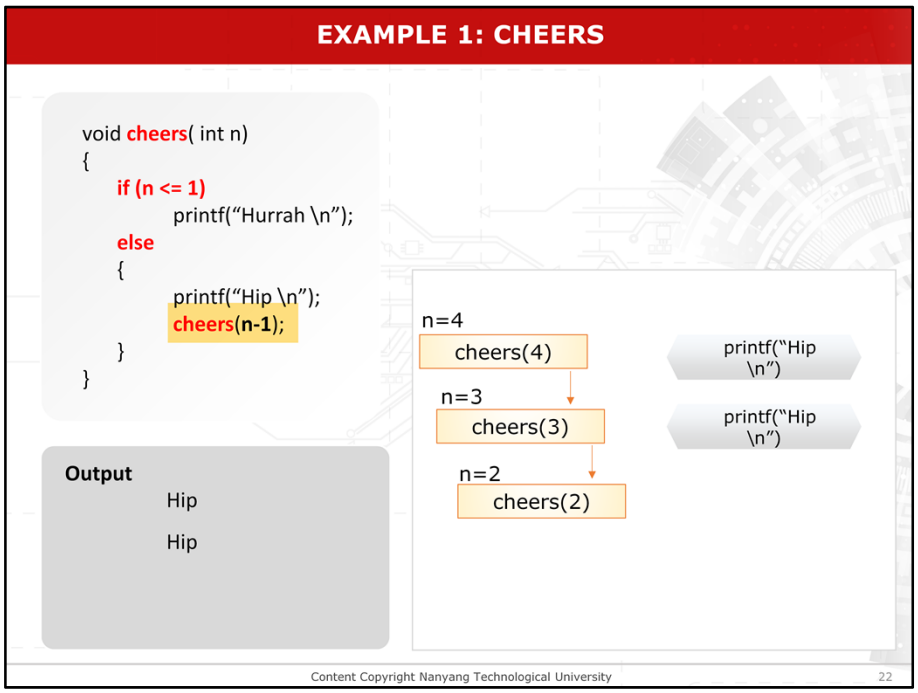

The string **"Hip"** will be printed on the screen,



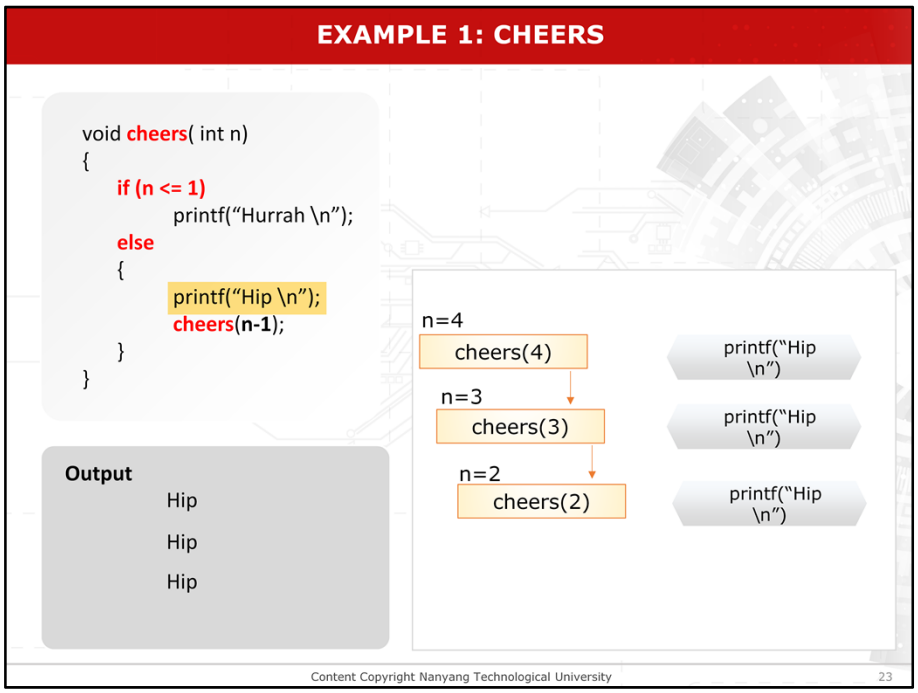

and the function **cheers (3)** will be called with **n** is equal to 3.



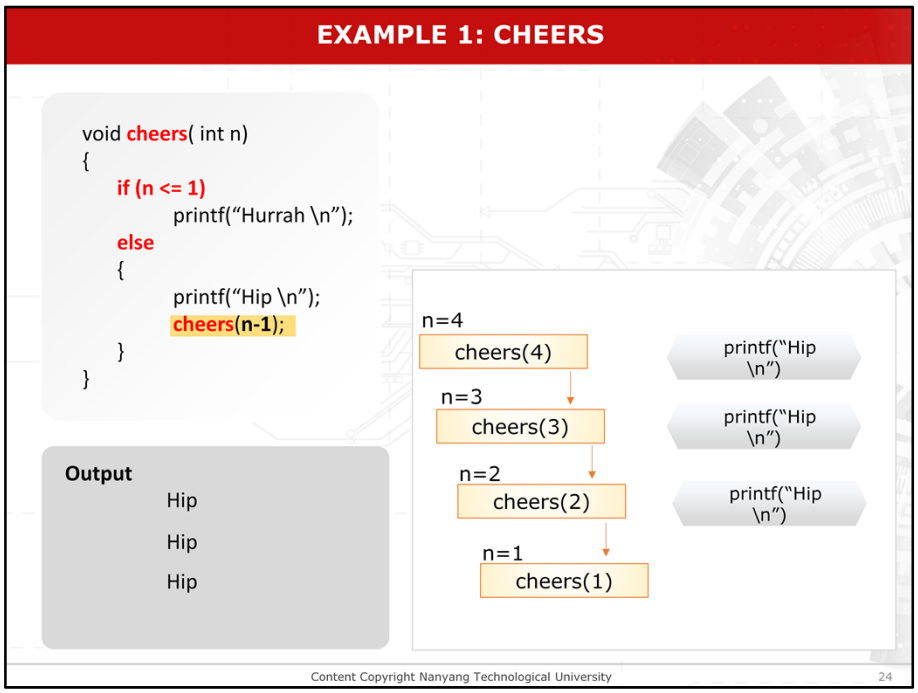

This will be repeated to print the string **"Hip"** for two more times until the function **cheers (1)** is called with **n** reduced to 1.


EXAMPLE 1: CHEERS
void cheers( int n)
{
if (n <= 1)
printf("Hurrah \n");
else
{
printf("Hip \n");
cheers(n-1);
}
}
Output
Hip
Hip
n=4
cheers(4)
n=3
cheers(3)
n=2
cheers(2)
printf("Hip \n")
printf("Hip \n")
Content Copyright Nanyang Technological University
22

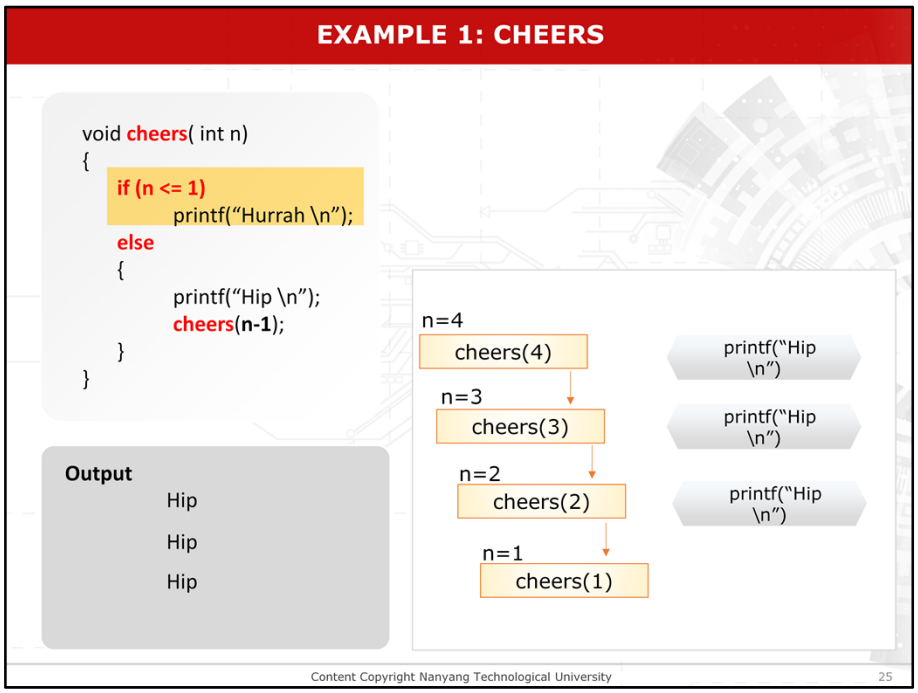
[no audio]


EXAMPLE 1: CHEERS
void cheers( int n)
{
if (n <= 1)
printf("Hurrah \n");
else
{
printf("Hip \n");
cheers(n-1);
}
}
Output
Hip
Hip
Hip
n=4
cheers(4)
n=3
cheers(3)
n=2
cheers(2)
printf("Hip \n")
printf("Hip \n")
printf("Hip \n")
Content Copyright Nanyang Technological University
23

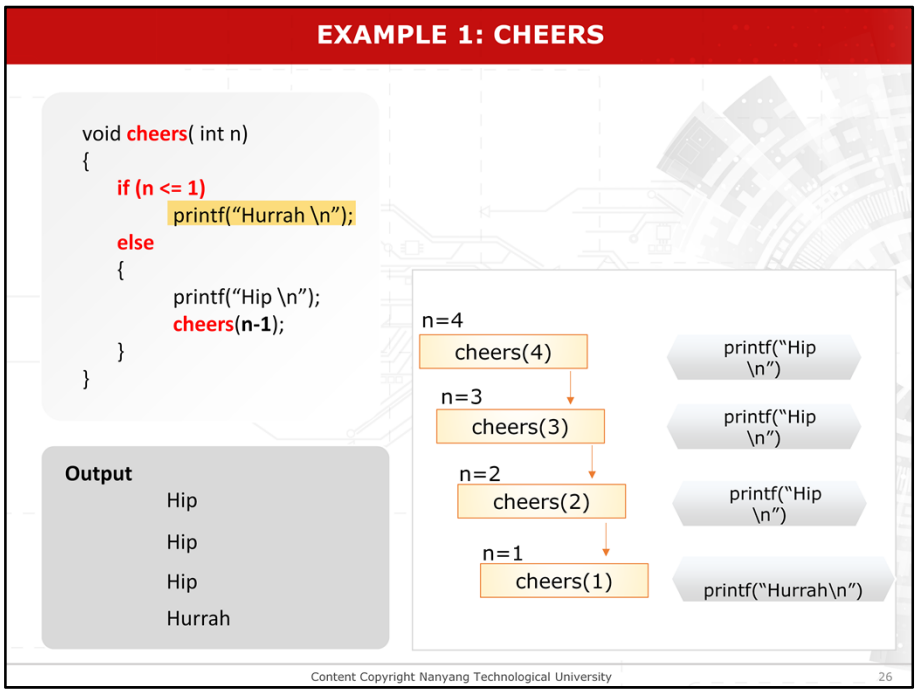
[no audio]


EXAMPLE 1: CHEERS
void cheers( int n)
{
if (n <= 1)
printf("Hurrah \n");
else
{
printf("Hip \n");
cheers(n-1);
}
}
Output
Hip
Hip
Hip
n=4
cheers(4)
n=3
cheers(3)
n=2
cheers(2)
n=1
cheers(1)
printf("Hip \n")
printf("Hip \n")
printf("Hip \n")
Content Copyright Nanyang Technological University
24

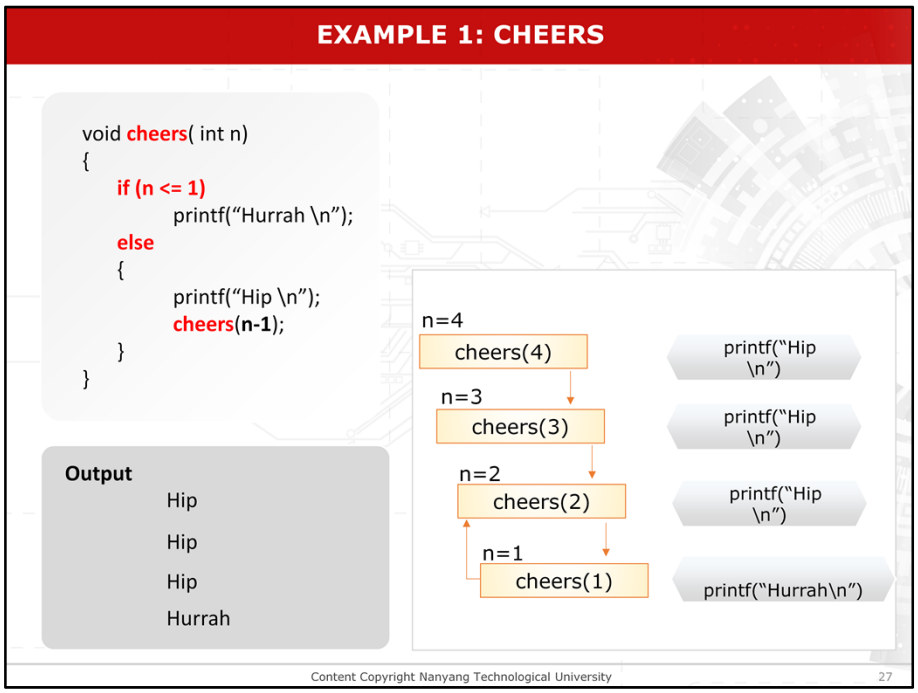
[no audio]



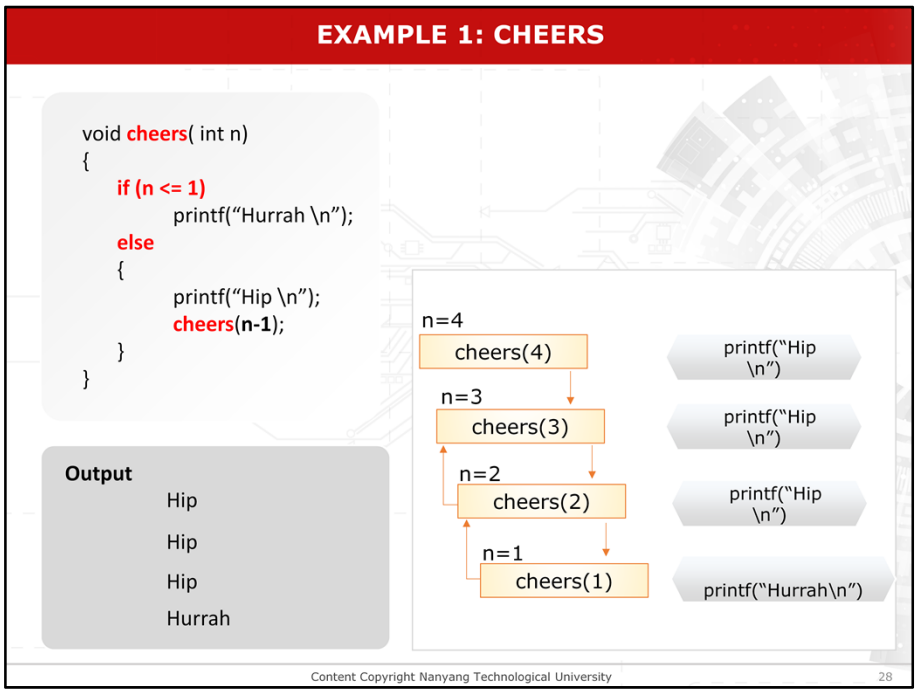

When n is equal to 1.



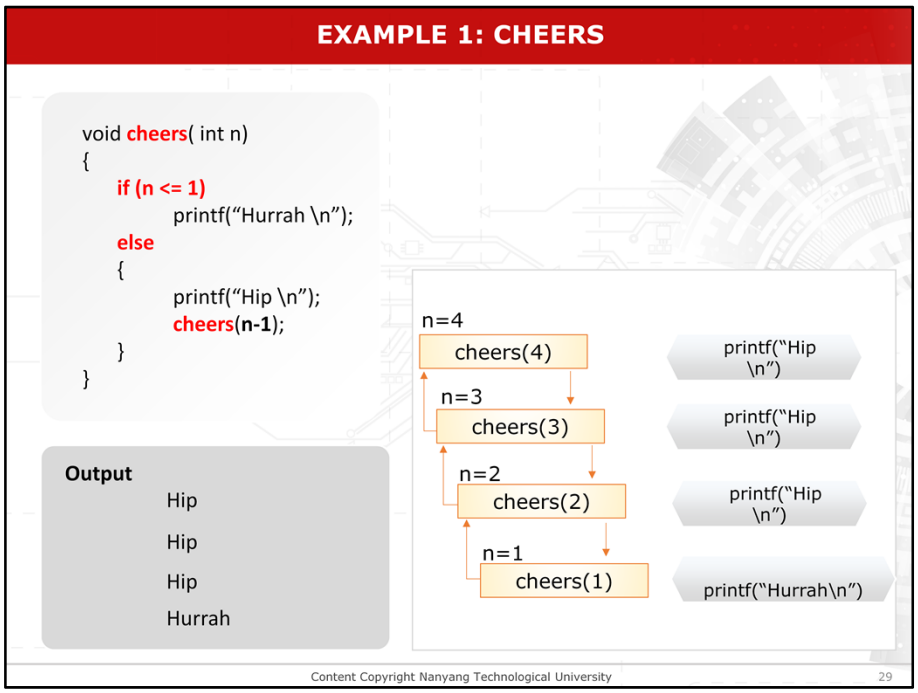

the string **"Hurrah"** will be printed on the screen.



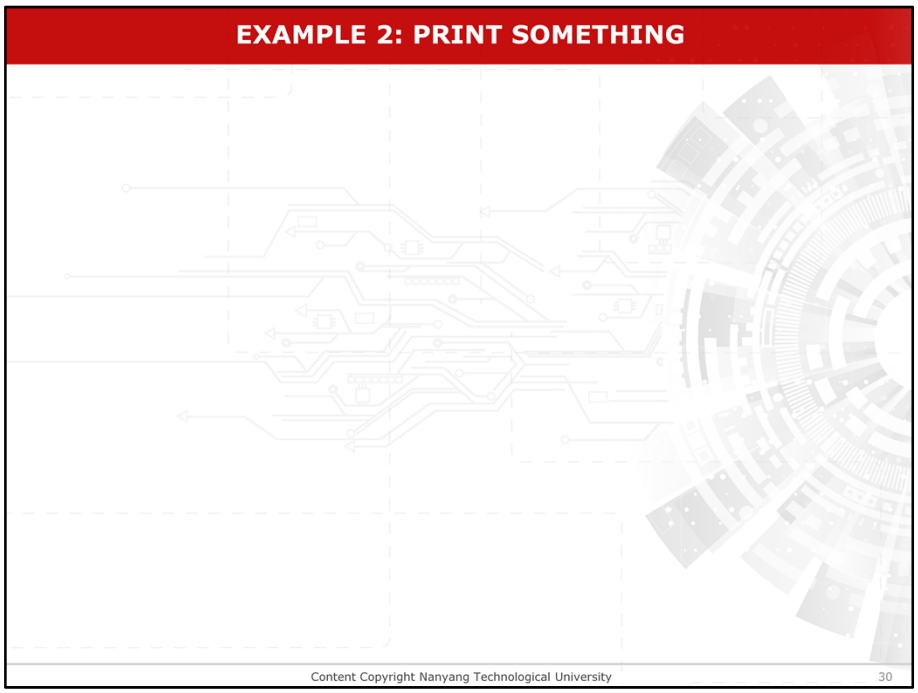

When **cheers(1)** has completed execution, the control is returned to the previous function call executing **cheers(2)**.




It will in turn complete the execution, and the control returns to the previous calling function **cheers(3)**.




The program stops execution when **cheers(4)** has completed the execution.



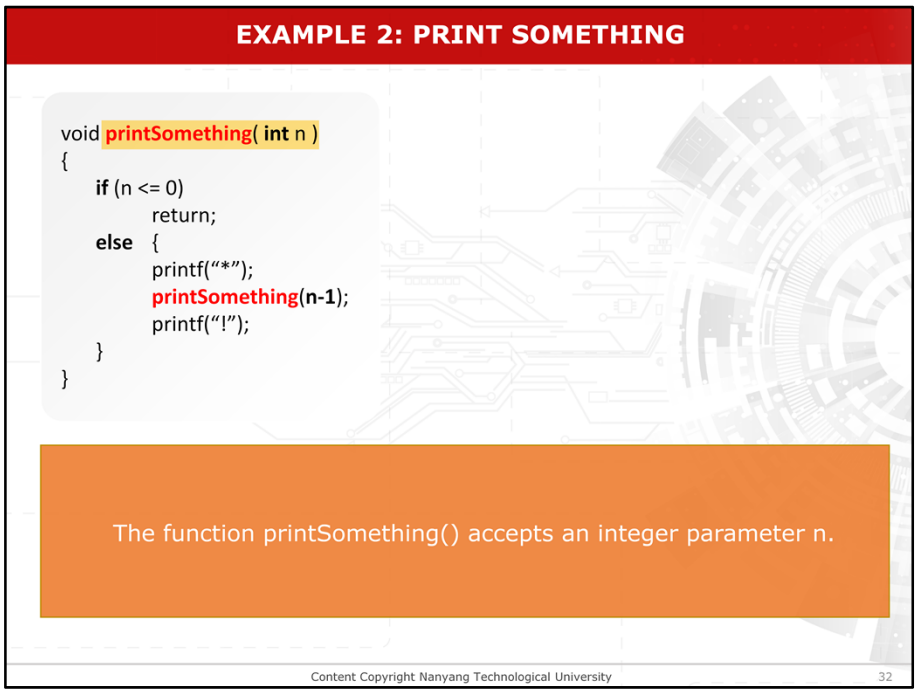

Example 2: Print Something

## EXAMPLE 2: PRINT SOMETHING

```
void printSomething( int n )
{
    if (n <= 0)
        return;
    else  {
        printf("*");
        printSomething(n-1);
        printf("!");
    }
}
```

Content Copyright Nanyang Technological University


The recursive function called **print Something()** is implemented as shown.



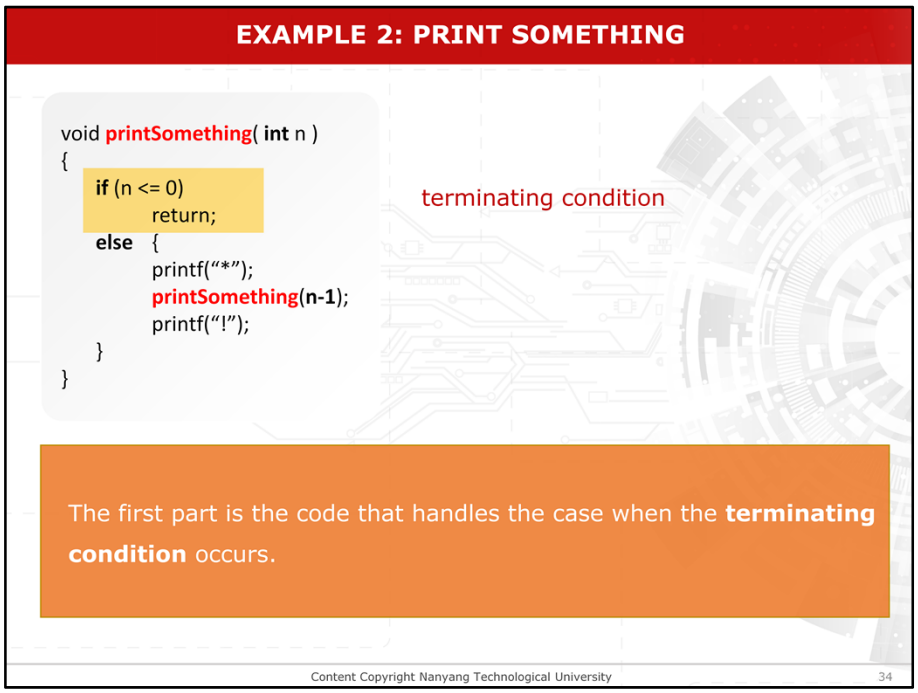

The function **print Something()** accepts an integer parameter **n**.

## EXAMPLE 2: PRINT SOMETHING

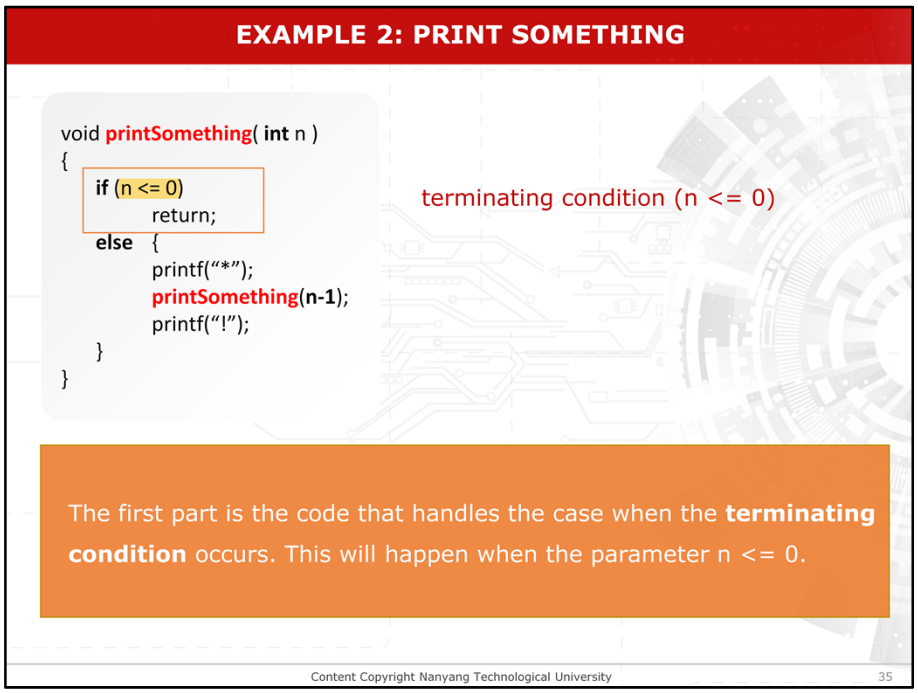
```
void printSomething( int n )
{
    if (n <= 0)
        return;
    else  {
        printf("*");
        printSomething(n-1);
        printf("!");
    }
}
```

Content Copyright Nanyang Technological University

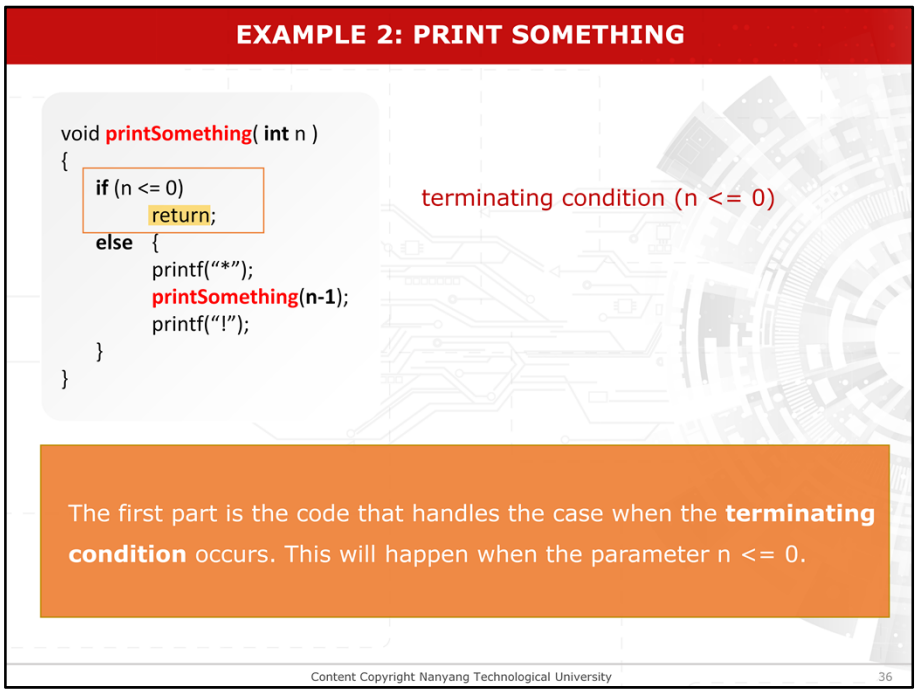

There are two parts of coding in the function.



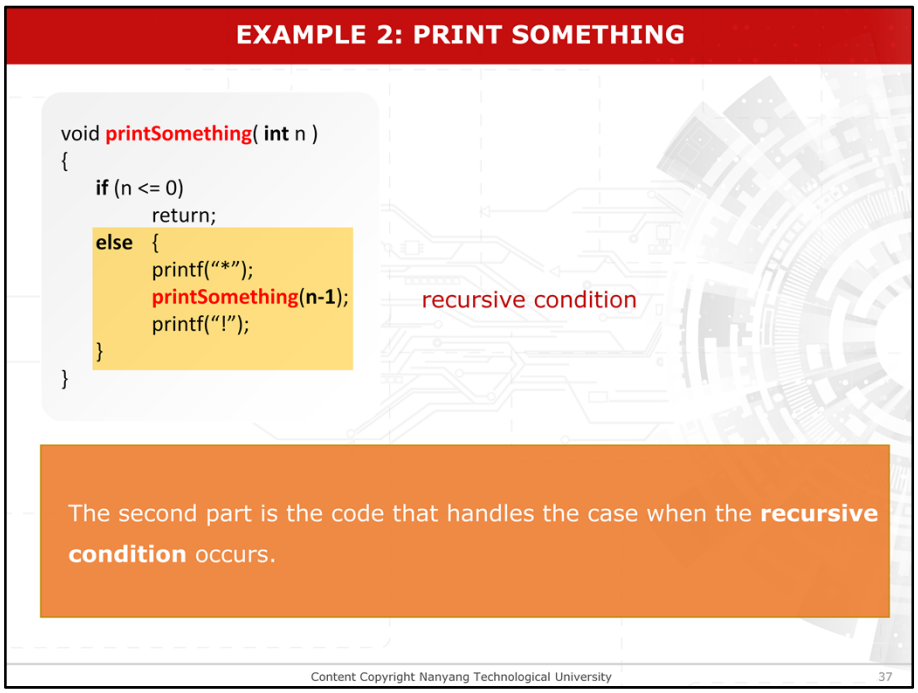

The first part is the code that handles the case when the terminating condition occurs.



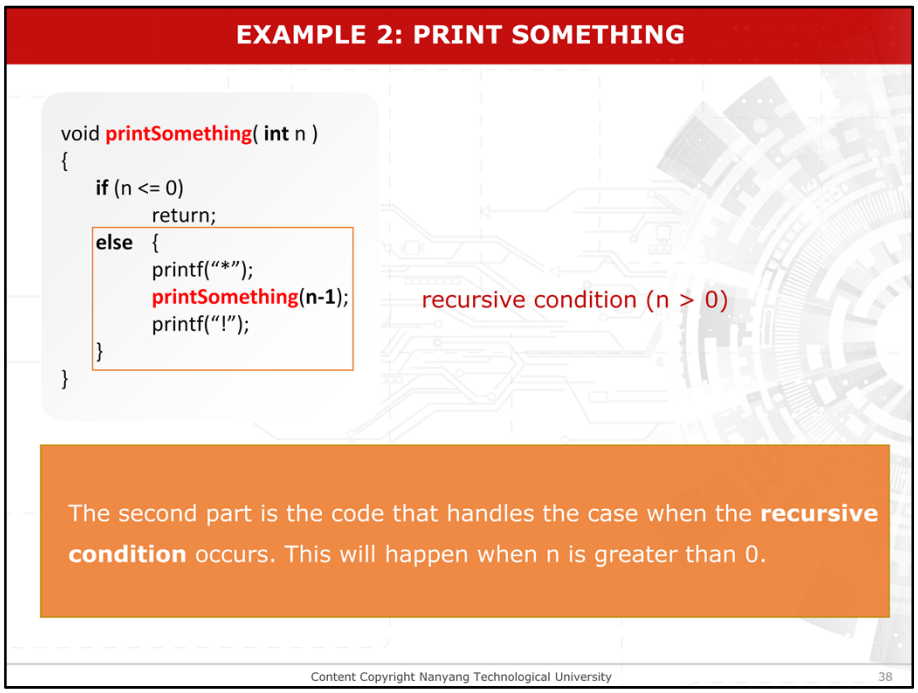

This will happen when the parameter n is lesser than or equal to 0.



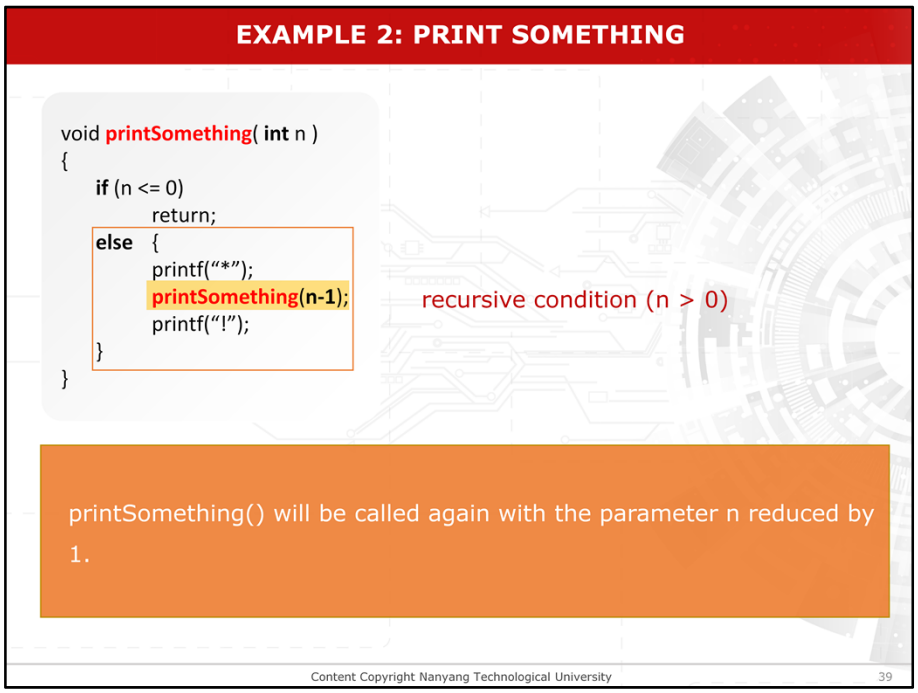

In this case, the control is returned.



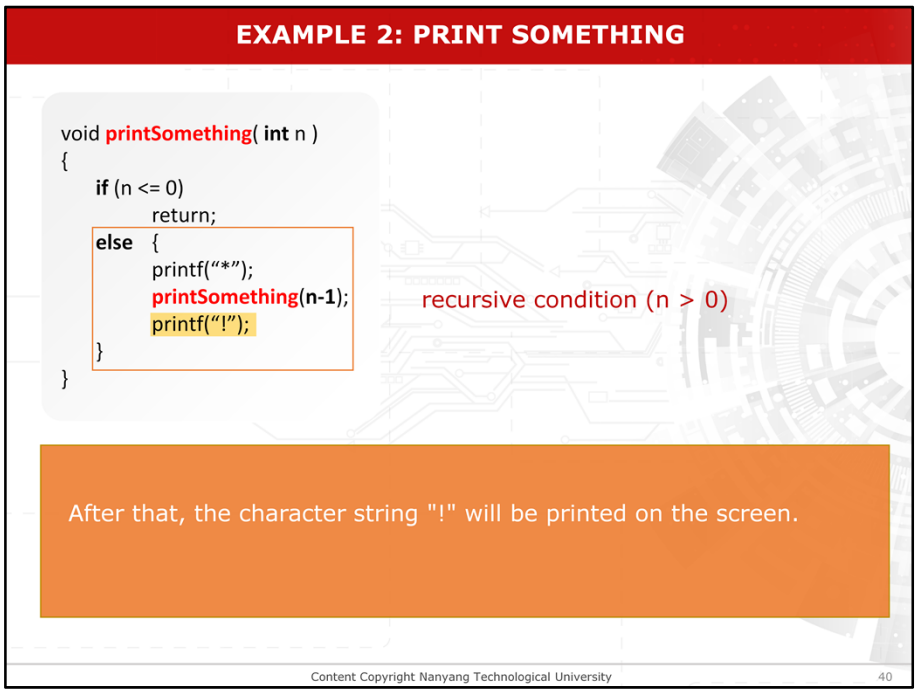

The second part is the code that handles the case when the recursive condition occurs.



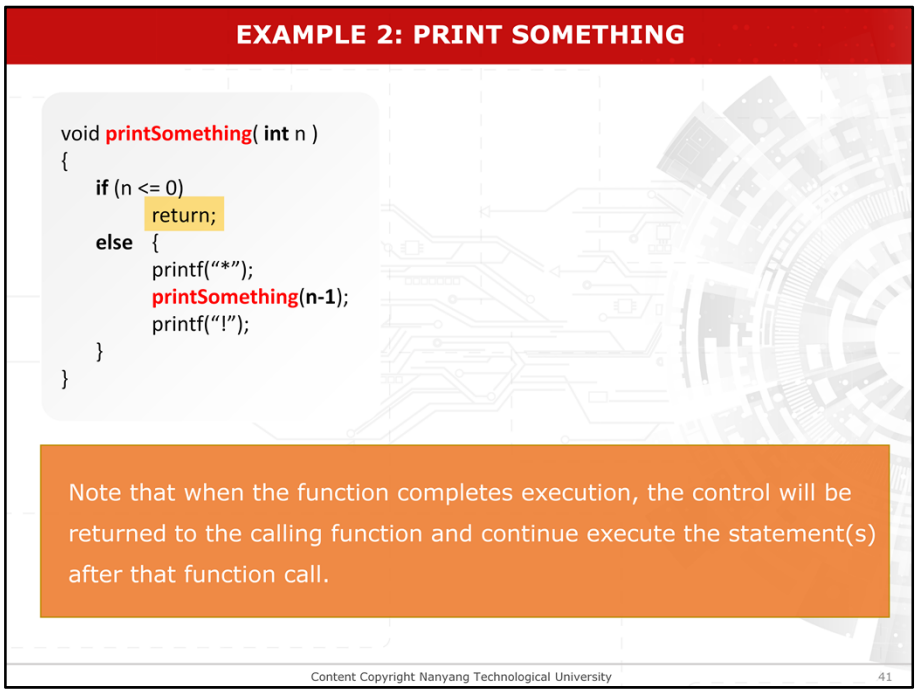

This will happen when n is greater than 0.




**Print Something()** will be called again with the parameter **n** reduced by 1.




After that, the character string **exclamation mark** will be printed on the screen.




Note that when the function completes execution, the control will be returned to the calling function and continue execute the statements after that function call.

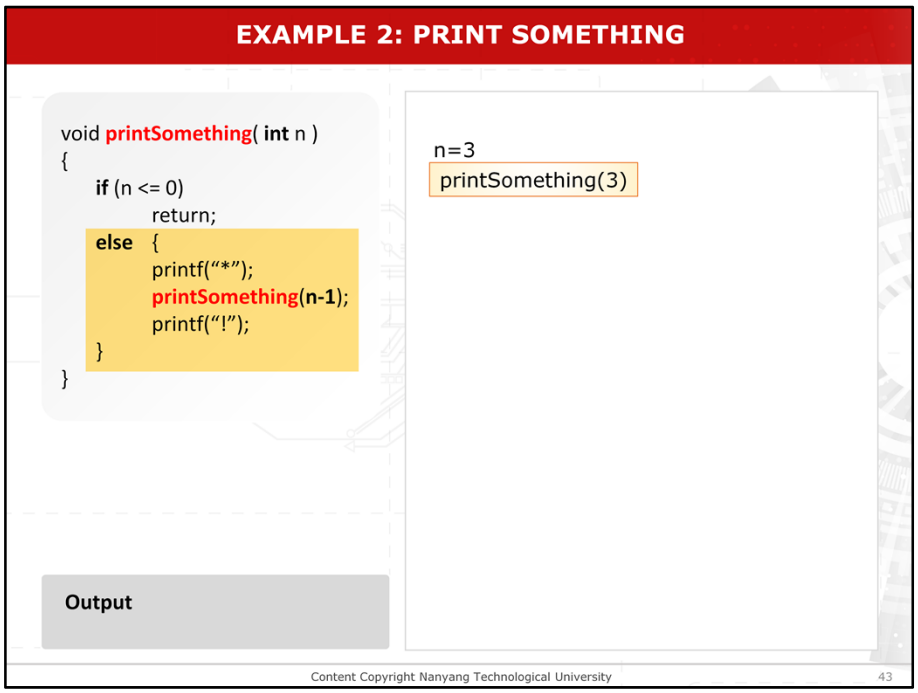
## EXAMPLE 2: PRINT SOMETHING

```
void printSomething( int n )
{
    if (n <= 0)
        return;
    else  {
        printf("*");
        printSomething(n-1);
        printf("!");
    }
}
```

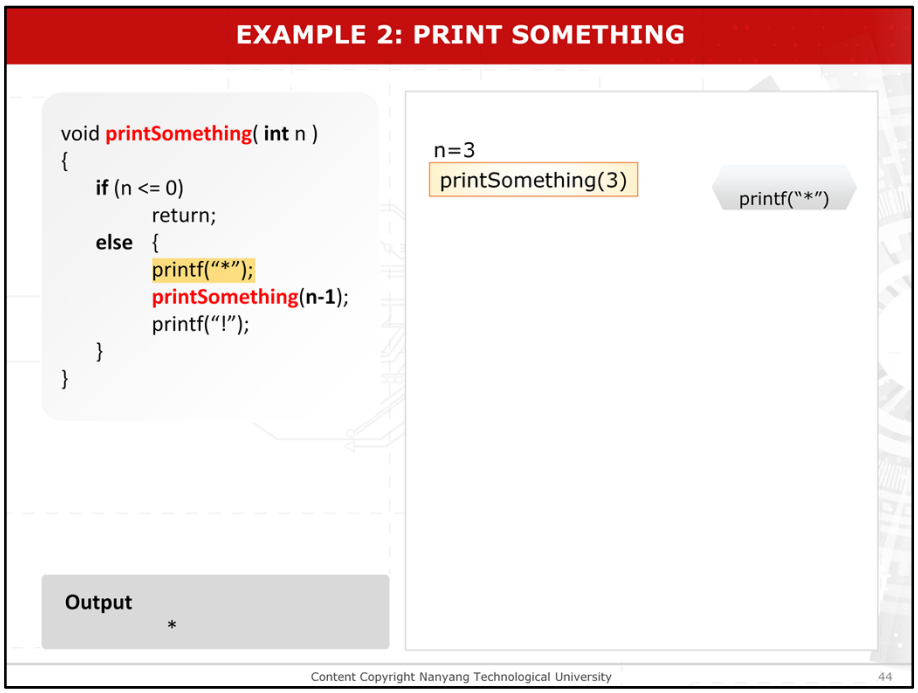
What does the recursive function print when called with **printSomething(3)**?

Content Copyright Nanyang Technological University

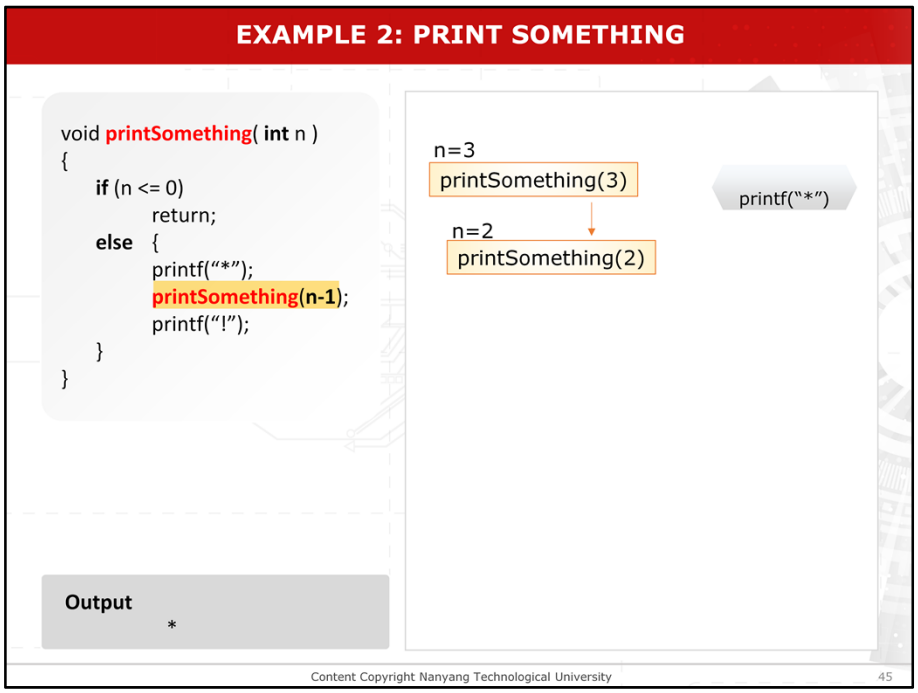

The program tracing for is illustrated as follows.




When the parameter **n** is equal to 3, the code for recursive condition is then executed.

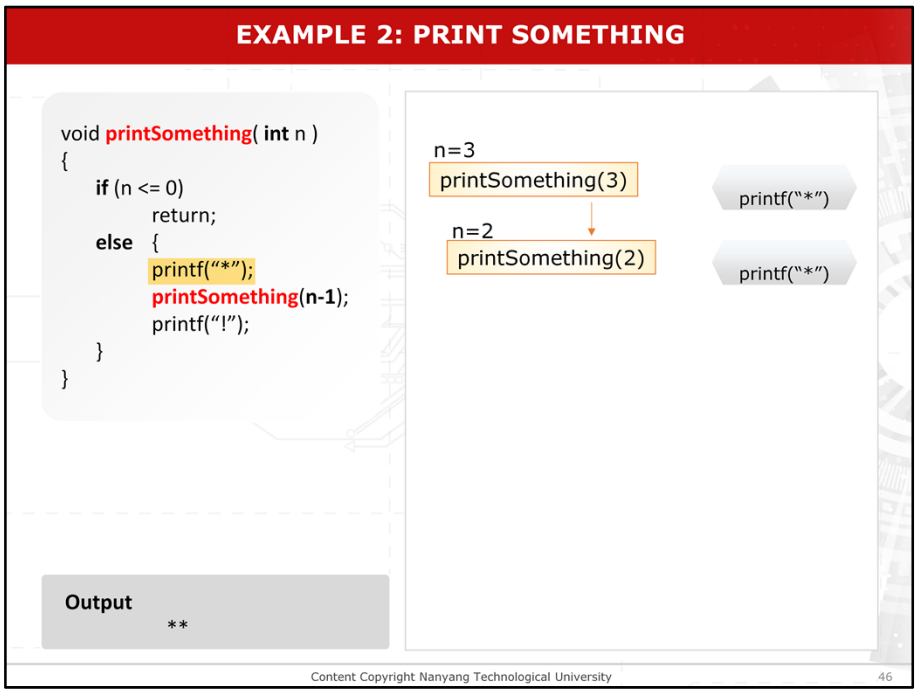



The string asterisk will be printed on the screen.

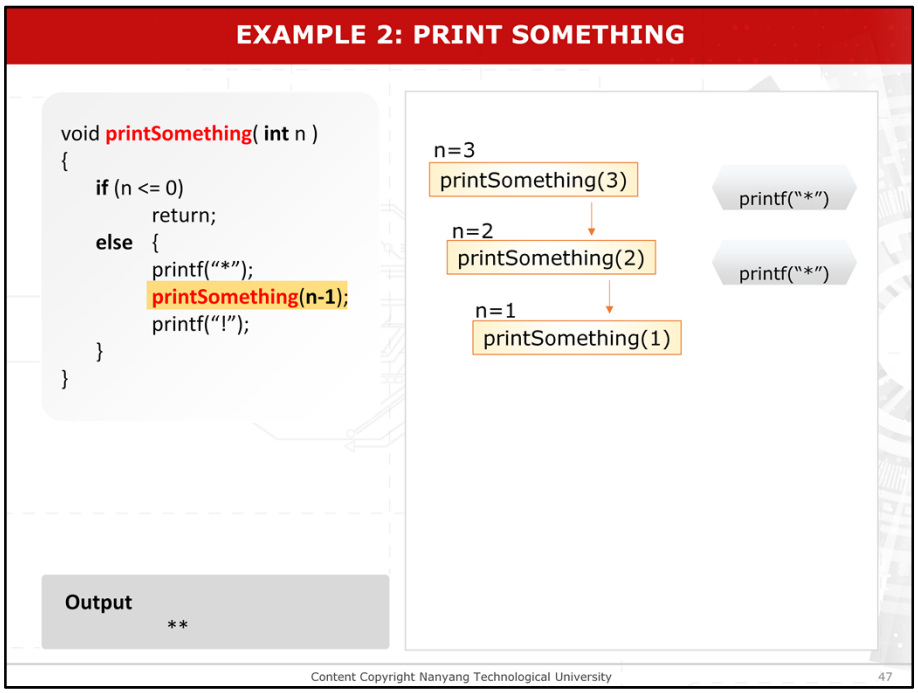



The function **print Something (2)** will be called with **n** is equal to 2.

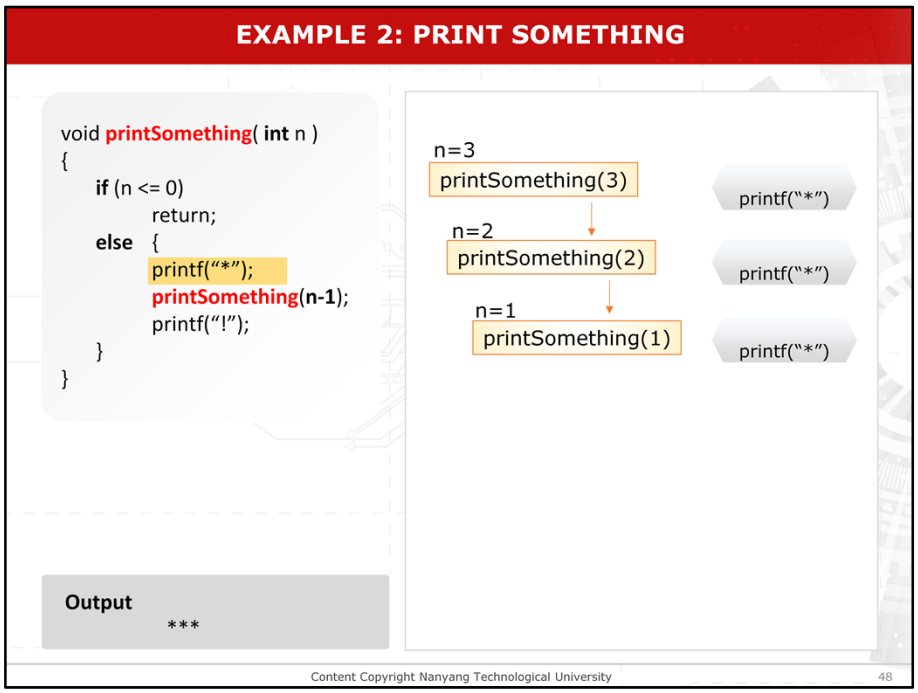
This will be repeated to print the **asterisk** for two more times until the function **print Something (0)** is called with **n** reduced to 0.


EXAMPLE 2: PRINT SOMETHING
void printSomething( int n )
{
if (n <= 0)
return;
else {
printf("*");
printSomething(n-1);
printf("!");
}
}
n=3
printSomething(3)
n=2
printSomething(2)
printf("*")
printf("*")
Output
**
Content Copyright Nanyang Technological University
46

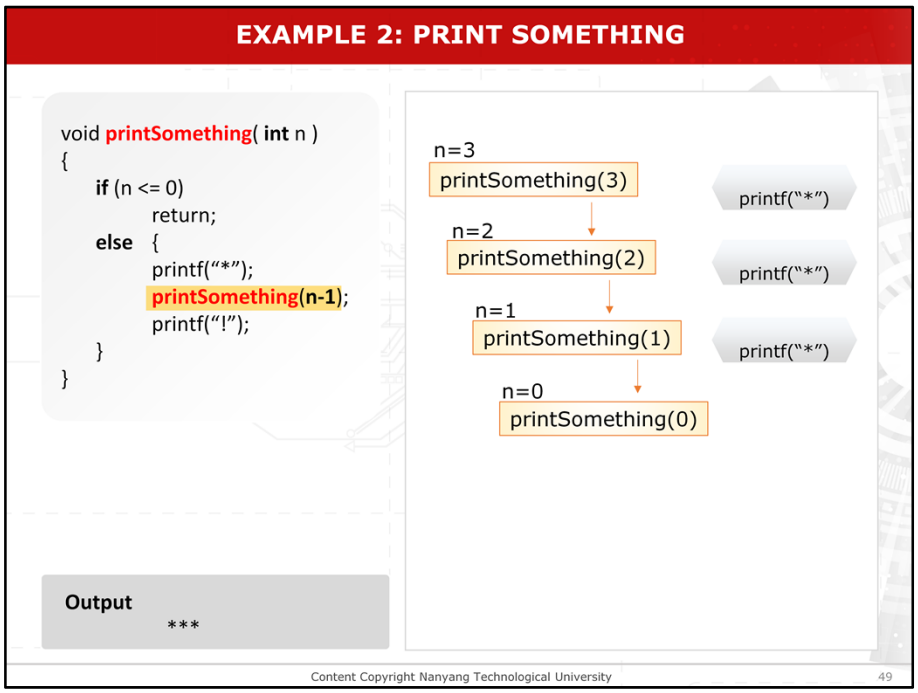
[no audio]


EXAMPLE 2: PRINT SOMETHING
void printSomething( int n )
{
if (n <= 0)
return;
else {
printf("*");
printSomething(n-1);
printf("!");
}
}
n=3
printSomething(3)
n=2
printSomething(2)
n=1
printSomething(1)
printf("*")
printf("*")
Output
**
Content Copyright Nanyang Technological University
47

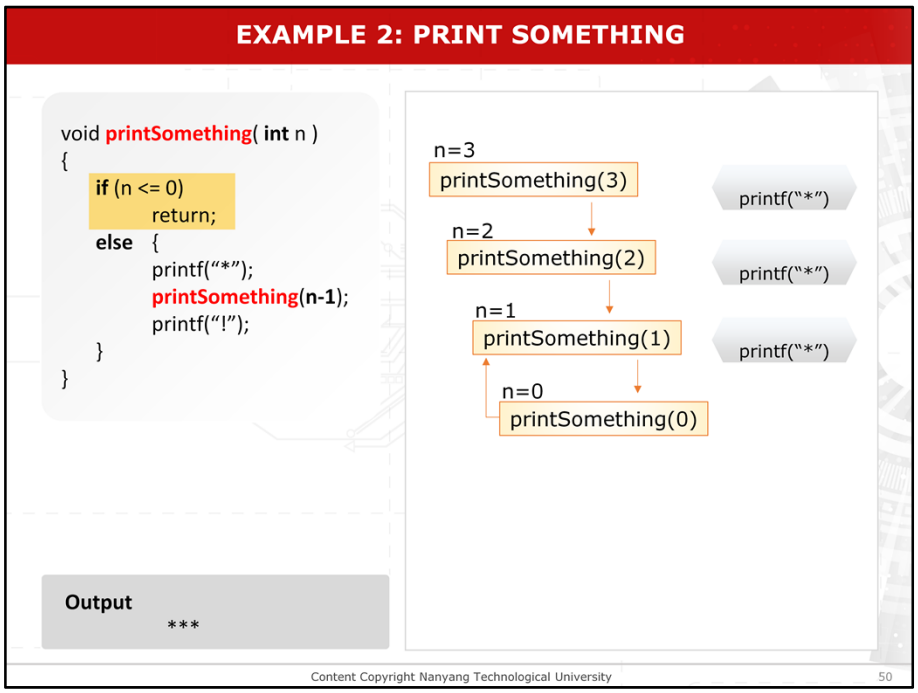
[no audio]


EXAMPLE 2: PRINT SOMETHING
void printSomething( int n )
{
if (n <= 0)
return;
else {
printf("*");
printSomething(n-1);
printf("!");
}
}
n=3
printSomething(3)
n=2
printSomething(2)
n=1
printSomething(1)
printf("*")
printf("*")
printf("*")
Output
***
Content Copyright Nanyang Technological University
48

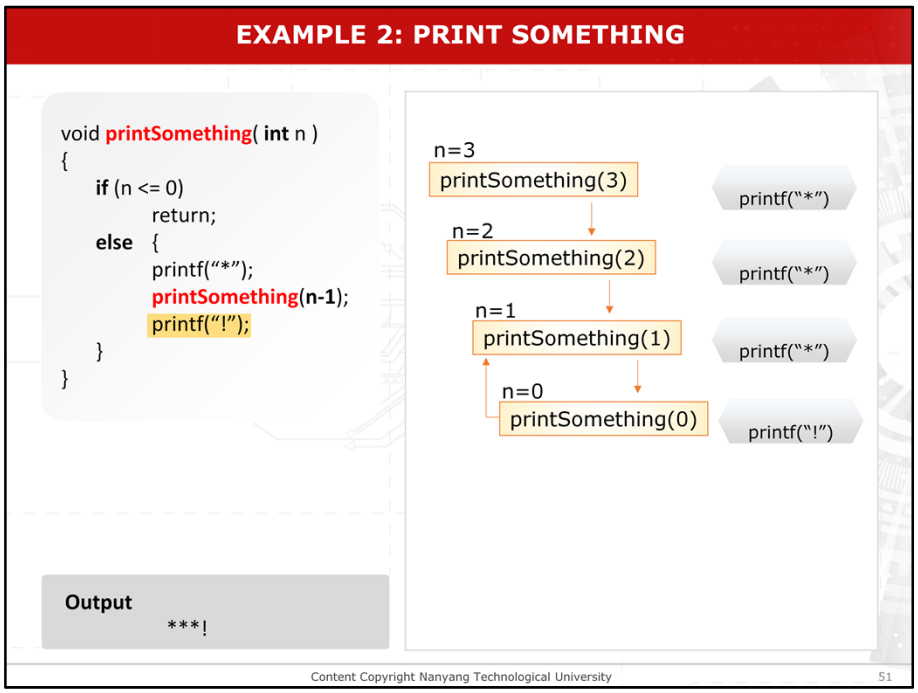
[no audio]


EXAMPLE 2: PRINT SOMETHING
void printSomething( int n )
{
if (n <= 0)
return;
else {
printf("*");
printSomething(n-1);
printf("!");
}
}
n=3
printSomething(3)
printf("*")
n=2
printSomething(2)
printf("*")
n=1
printSomething(1)
printf("*")
n=0
printSomething(0)
Output
***
Content Copyright Nanyang Technological University
49

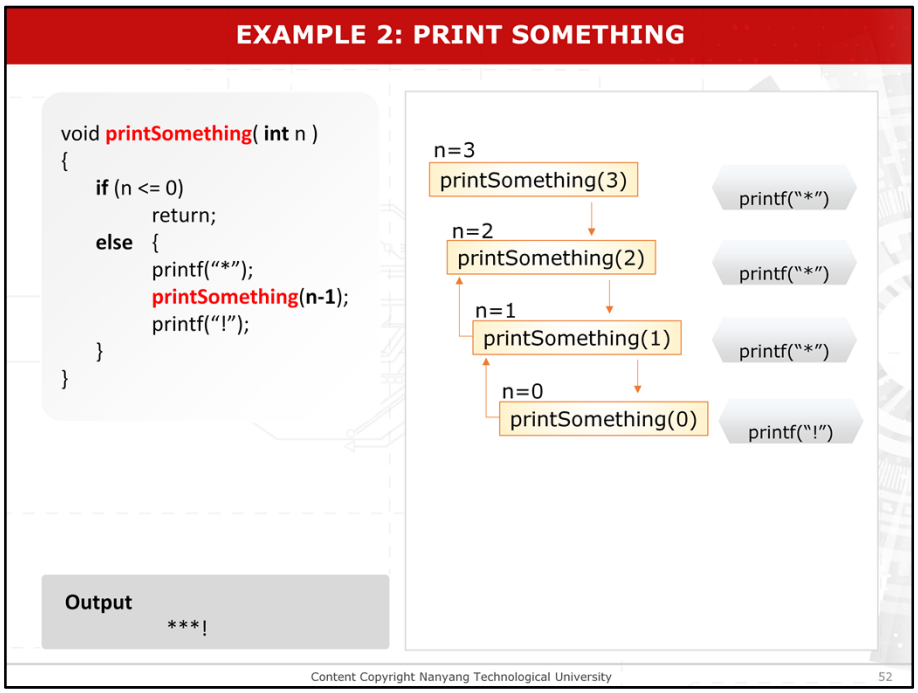
[no audio]



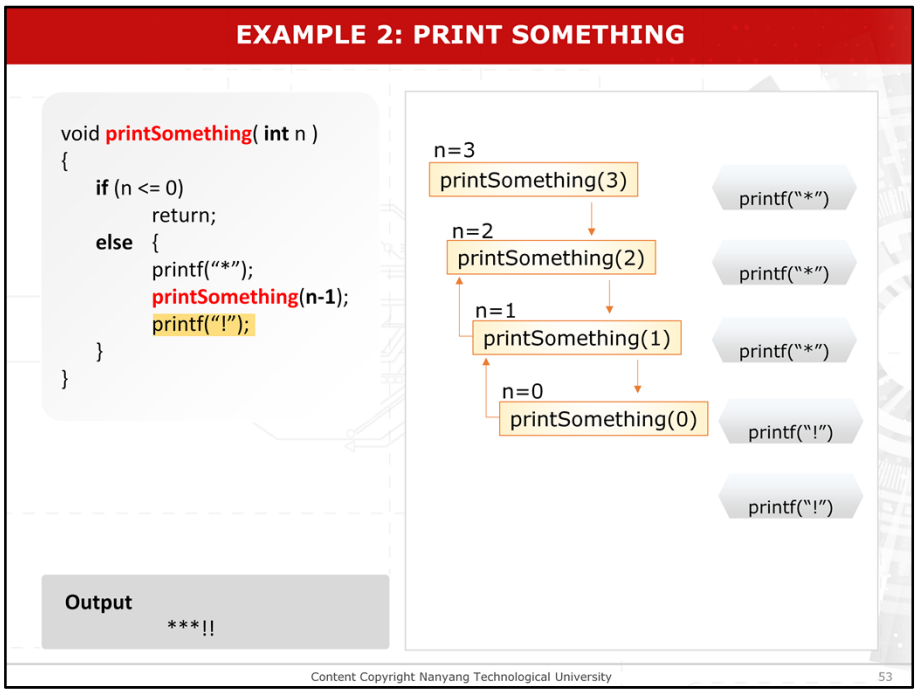

At n is equal to 0, the **return** statement will return the control to the previous function call **print Something (1)**.



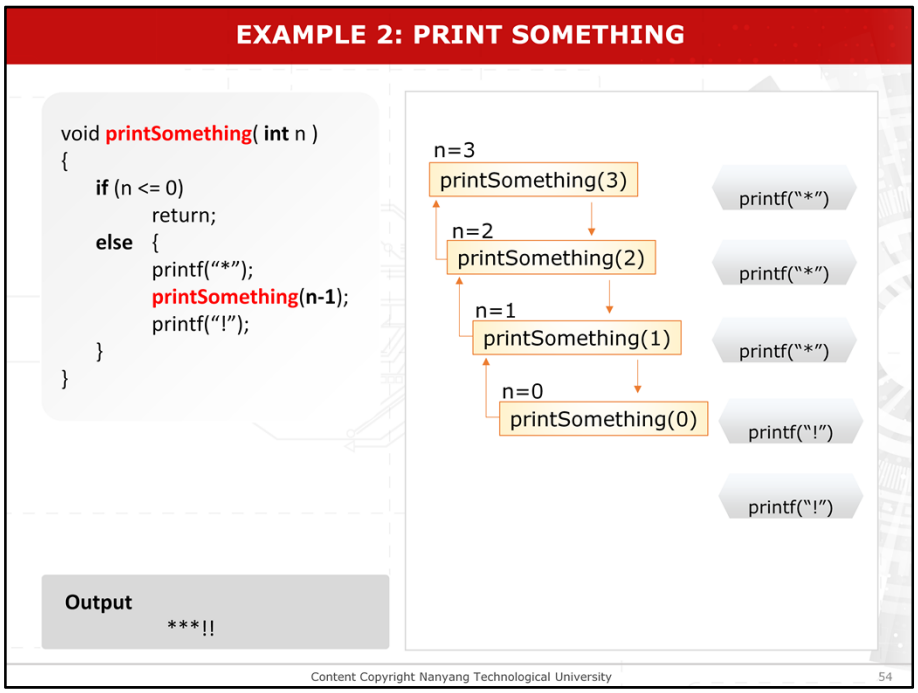

In **print Something (1)**, it will continue its execution with the next statement **print f (exclamation mark);** and the string **exclamation mark** will be printed.



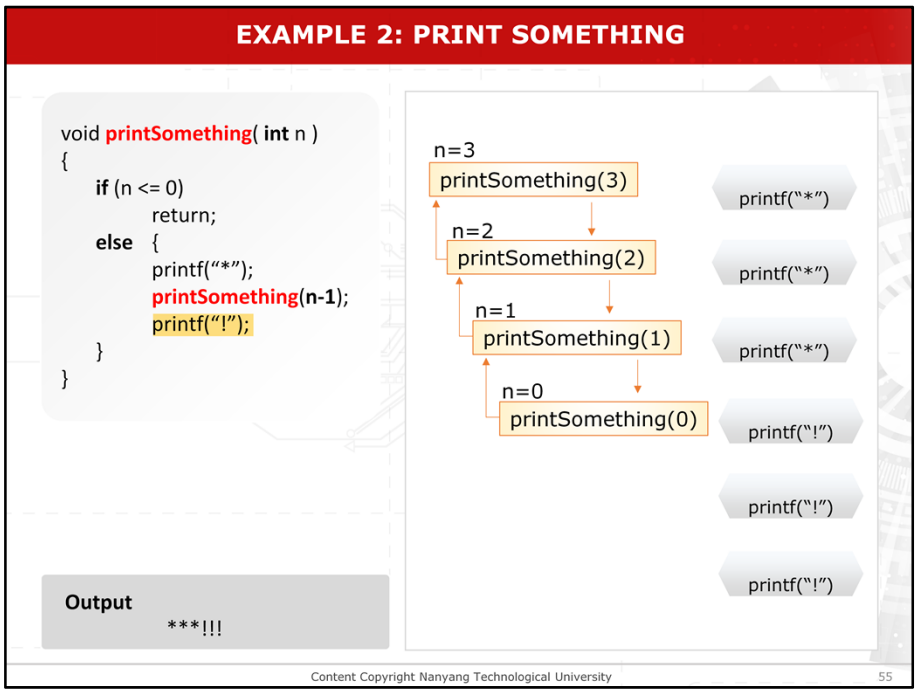

After that, the function has completed execution and the control is returned to the previous function call executing **print Something (2)**.



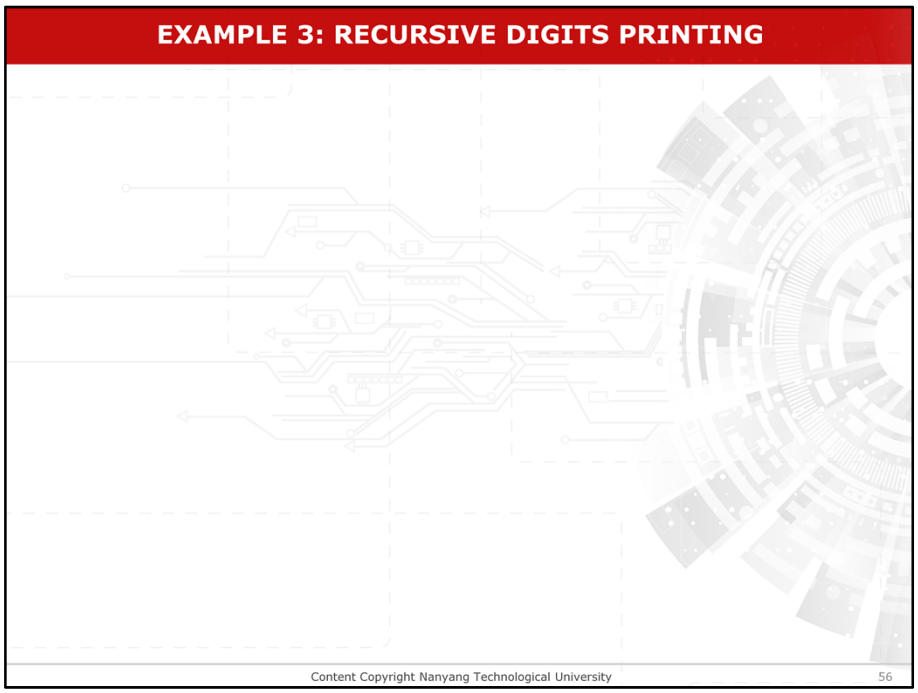

It will in turn complete the execution and print the string **exclamation mark**,



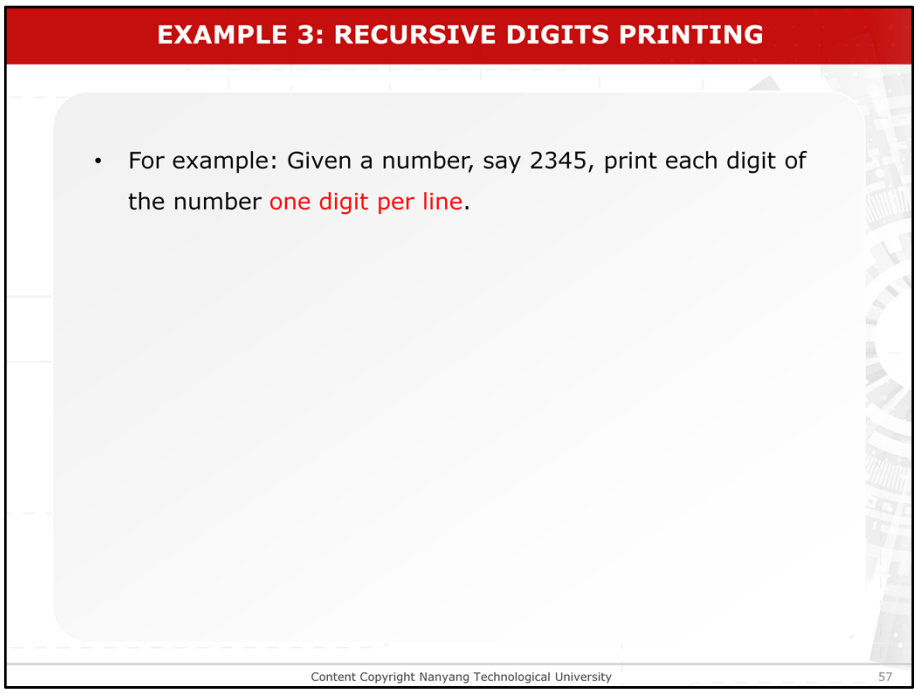

the control will then be returned to the previous calling function **print Something (3)**.




The function **print Something (3)** prints the string **exclamation mark** and returns the control to the calling **main()** function.




Example 3: Recursive Digits Printing



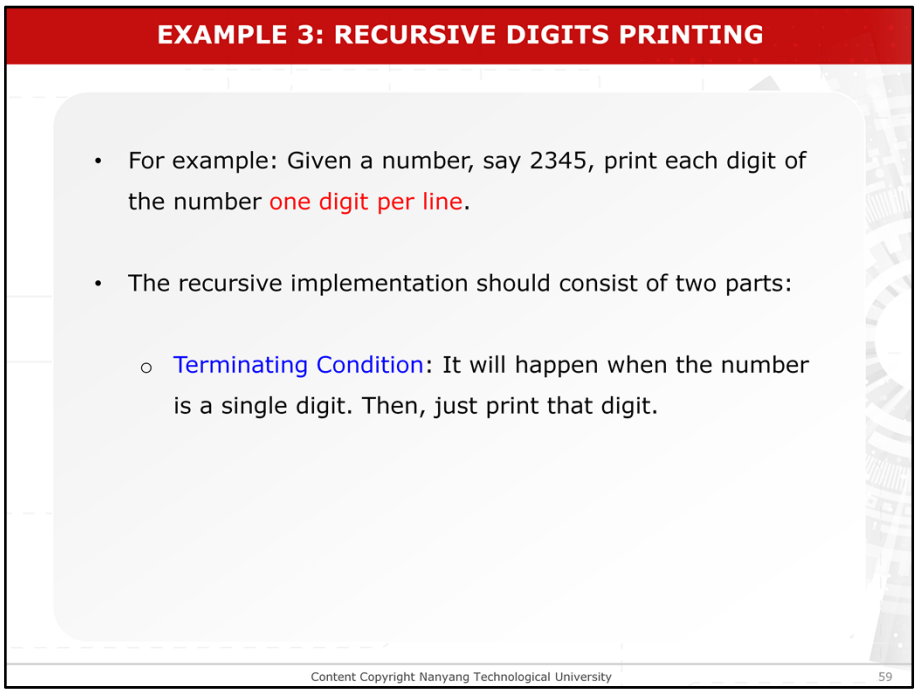

In this example, we write a recursive function to print the digits of an integer number one digit per line.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

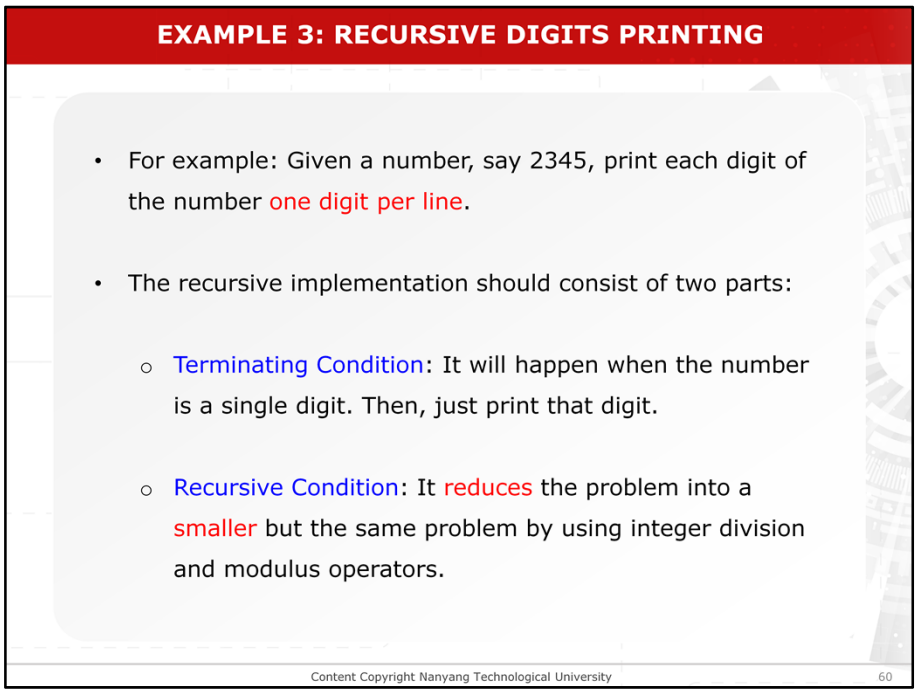
- For example: Given a number, say 2345, print each digit of the number one digit per line.
- The recursive implementation should consist of two parts:

Content Copyright Nanyang Technological University


To write the code, we need to divide the code into two parts, one for the terminating condition, and the other for the recursive condition.

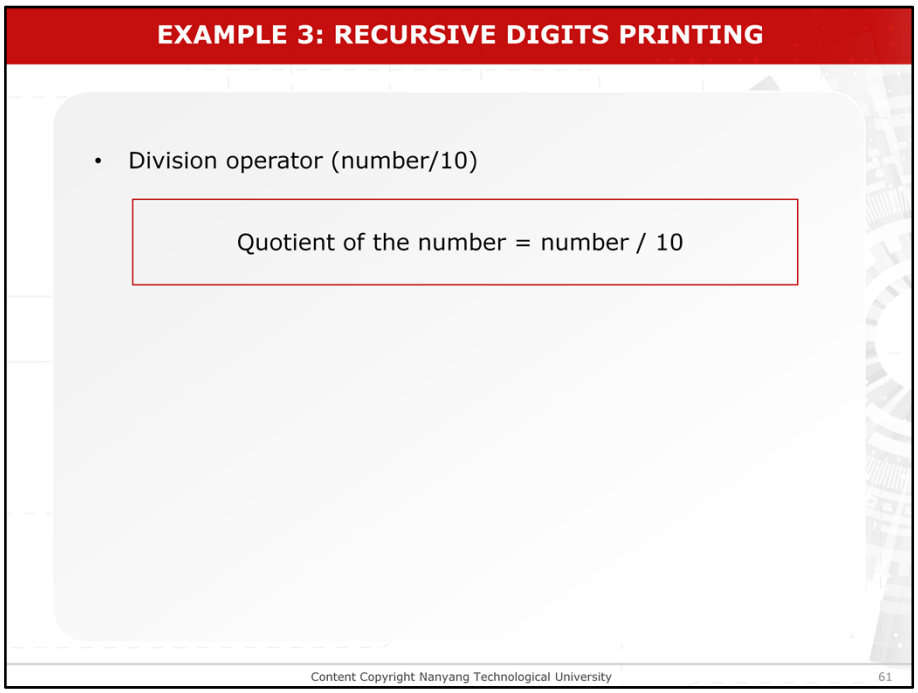



For the **terminating condition**, it will happen when the number is a single digit. In this case, the digit is just printed on the screen.

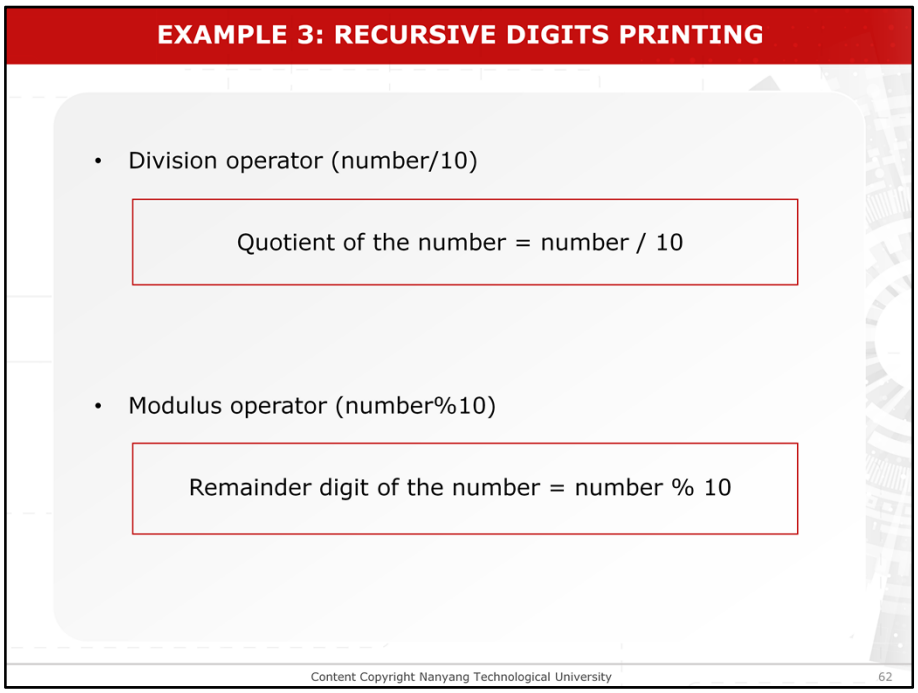



For the **recursive condition**, it will reduce the problem into a smaller but the same problem by calling the function itself again.

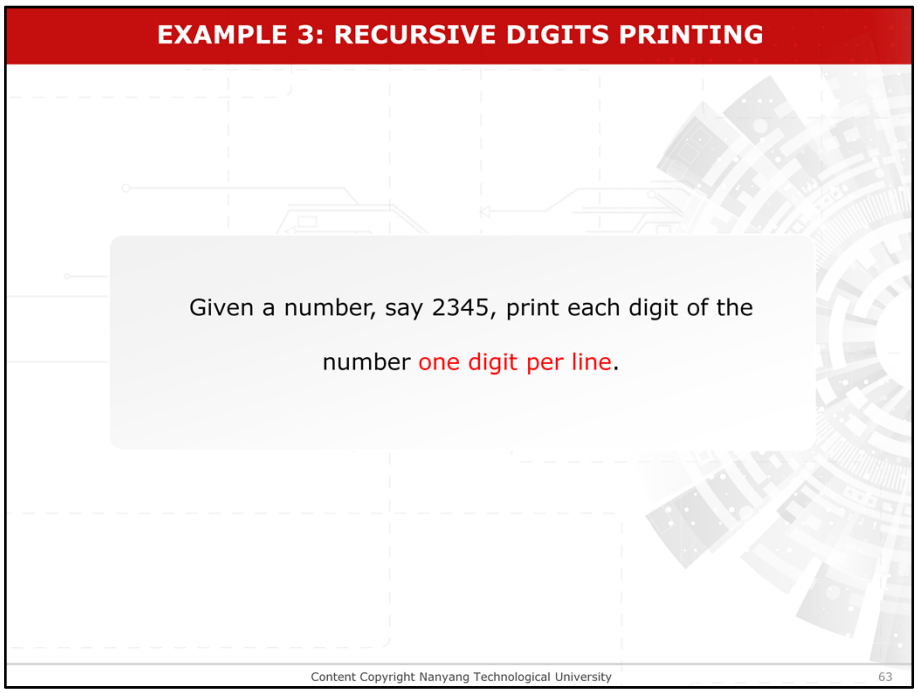
Here, we will need to use the division operator and modulus operator in order to obtain the quotient and remainder of a **number**.




The division operator will be used to determine the quotient of the number because quotient = **number** divided by 10




the modulus operator that is digit = **number modulus** 10 will give the remainder digit of the number.




Let's look at the code that uses the division and modulus operator in recursive function such that given a number, say 2345, the code will print each digit of the number one digit per line.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
            printf("%d", num);
    else   {
            printDigit(num/10);
            printf( "%d\n", num%10 );
    }
}
```

Content Copyright Nanyang Technological University


The recursive function **print Digit()** is implemented as shown.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
        printf("%d", num);
    else {
        printDigit(num/10);
        printf( "%d\n", num%10 );
    }
}
```

recursive function

Content Copyright Nanyang Technological University


In the recursive function **print Digit()**, it receives an integer argument **num** from the calling function.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
            printf("%d", num);
     else   {
            printDigit(num/10);
            printf( "%d\n", num%10 );
    }
}
```

terminating condition
(argument num is a single digit)

Content Copyright Nanyang Technological University


The terminating condition is defined when the argument **numb** is a single digit.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
        printf("%d", num);
    else {
        printDigit(num/10);
        printf( "%d\n", num%10 );
    }
}
```

recursive condition

Content Copyright Nanyang Technological University


The function calls itself for the recursive condition

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
        printf("%d", num);
    else {
        printDigit(num/10);
        printf( "%d\n", num%10 );
    }
}
```

recursive condition (argument num will be updated to get the quotient of num by using the division operator where num/10)

Content Copyright Nanyang Technological University


and each time the argument **numb** will be updated to get the quotient of **numb** by using the division operator where **numb is divided by 10**.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
            printf("%d", num);
    else   {
            printDigit(num/10);
            printf( "%d\n", num%10 );
    }
}
```

Content Copyright Nanyang Technological University


The corresponding digit of the input number is obtained through **numb modulus 10** which will be printed on the screen after the recursive call.

The recursive condition will be called repeatedly until the terminating condition is reached. In which case, the final digit of the input number will be printed on the screen.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
            printf("%d", num);
    else   {
            printDigit(num/10);
            printf( "%d\n", num%10 );
    }
}
```

**Output**

```
Enter a number: 2345
```

Content Copyright Nanyang Technological University



The program tracing for **print Digit (2345)** will be illustrated as follow.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
#include <stdio.h>
void printDigit(int num);
int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printDigit(num);
    return 0;
}
void printDigit(int num)
{
    if (num < 10)
        printf("%d", num);
    else {
        printDigit(num/10);
        printf( "%d\n", num%10 );
    }
}
```

**Output**

```
Enter a number: 2345
```

Content Copyright Nanyang Technological University


We will look into detail on the recursive function print Digit() for numb is equal to 2345

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

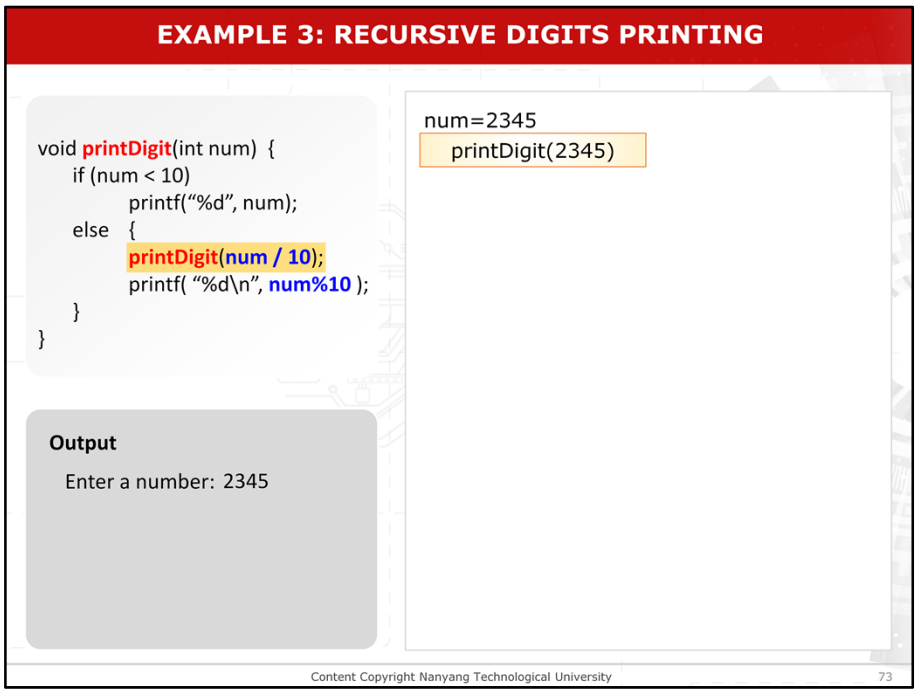
```
void printDigit(int num)  {
    if (num < 10)
            printf("%d", num);
    else   {
            printDigit(num / 10);
            printf( "%d\n", num%10 );
    }
}
```

**Output**

Enter a number: 2345

Content Copyright Nanyang Technological University


[no audio]




When the parameter **numb is equal to** 2345, the code for recursive condition is then executed.

## EXAMPLE 3: RECURSIVE DIGITS PRINTING

```
void printDigit(int num) {
    if (num < 10)
        printf("%d", num);
    else {
        printDigit(num / 10);
        printf( "%d\n", num%10 );
    }
}
```

**Output**

Enter a number: 2345

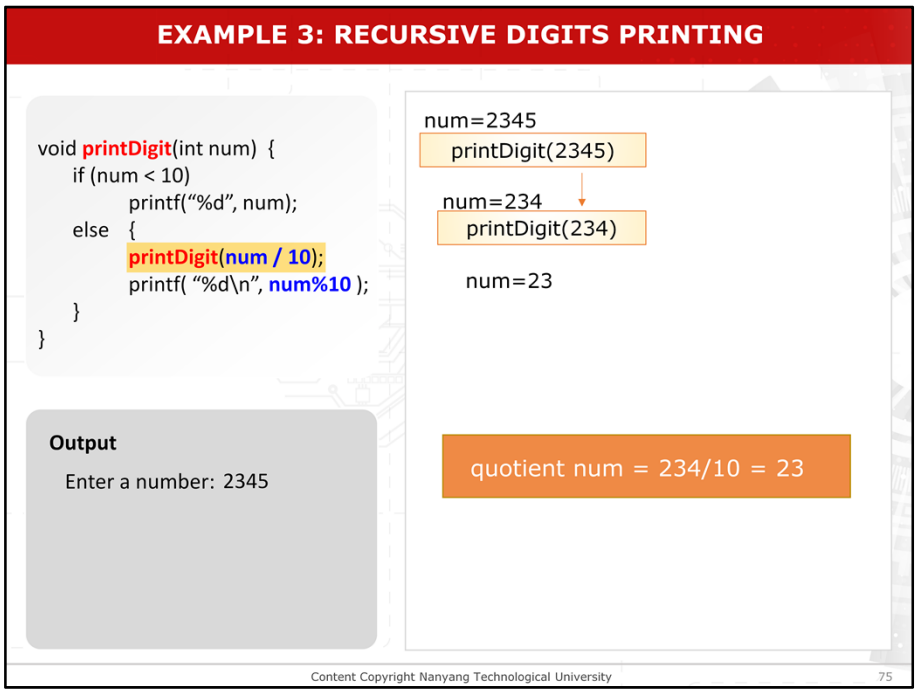
num=2345
printDigit(2345)

num=234

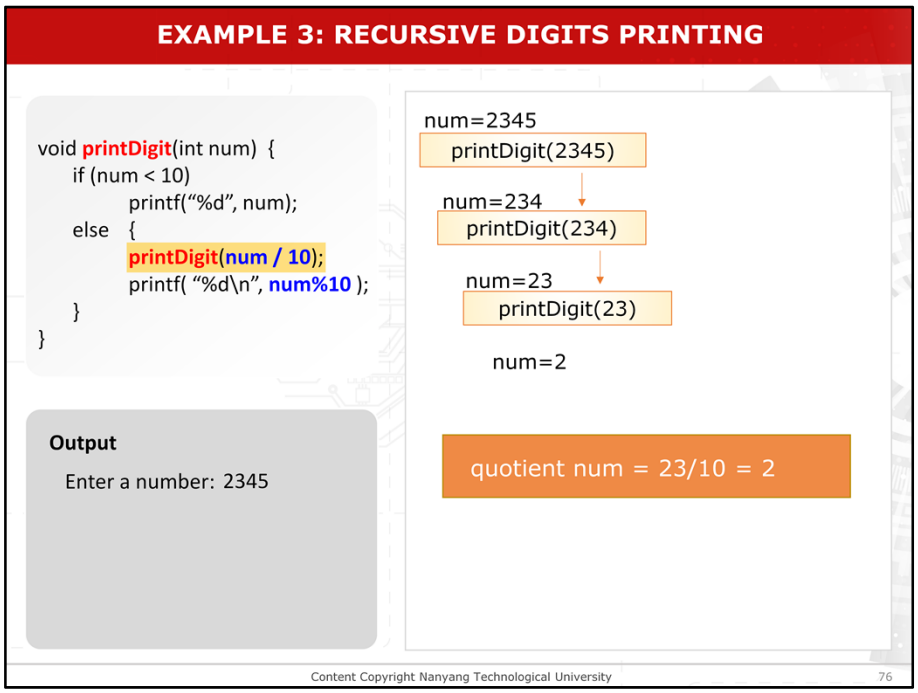
quotient num = 2345/10 = 234

Content Copyright Nanyang Technological University

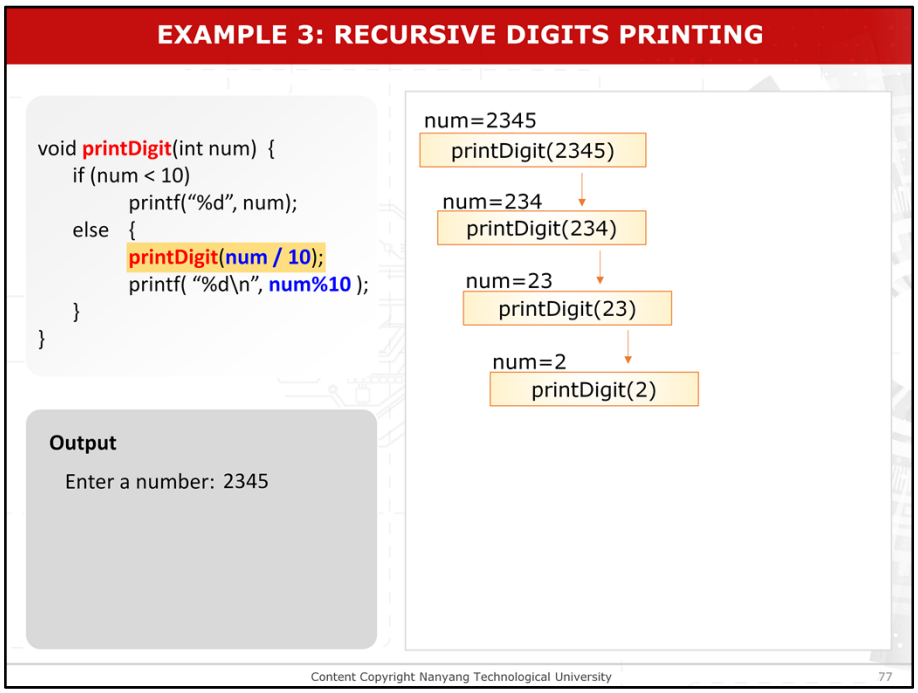

The function **print Digit (numb divided by 10)** will be called with the quotient **numb is equal to** 2345 divided by 10 is equal to **234**.



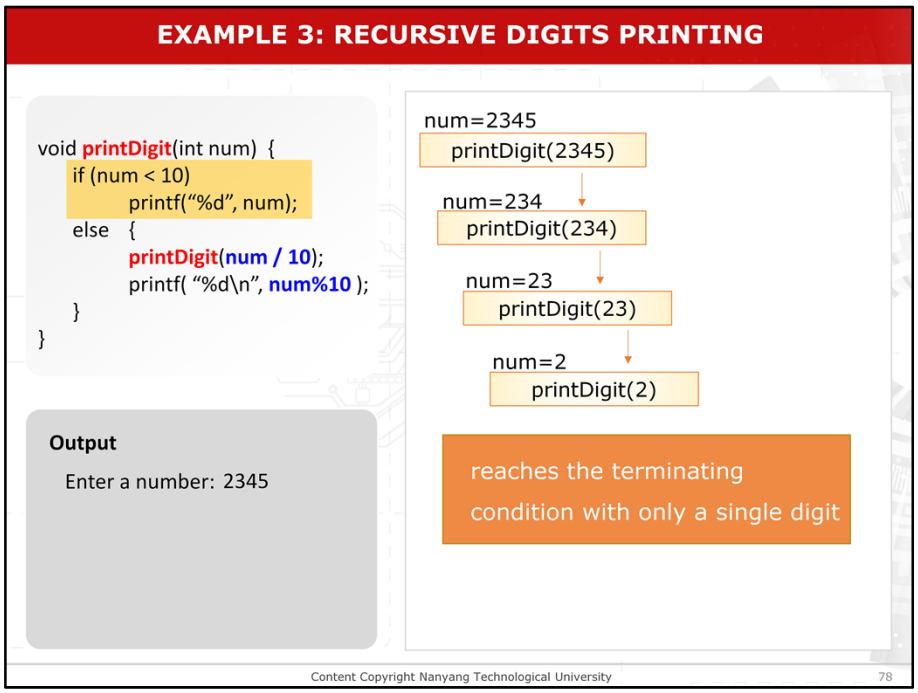

Again the function **print Digit (numb divided by 10)** will be called with the quotient **numb** is equal to 234 divided by 10 is equal to **23**.



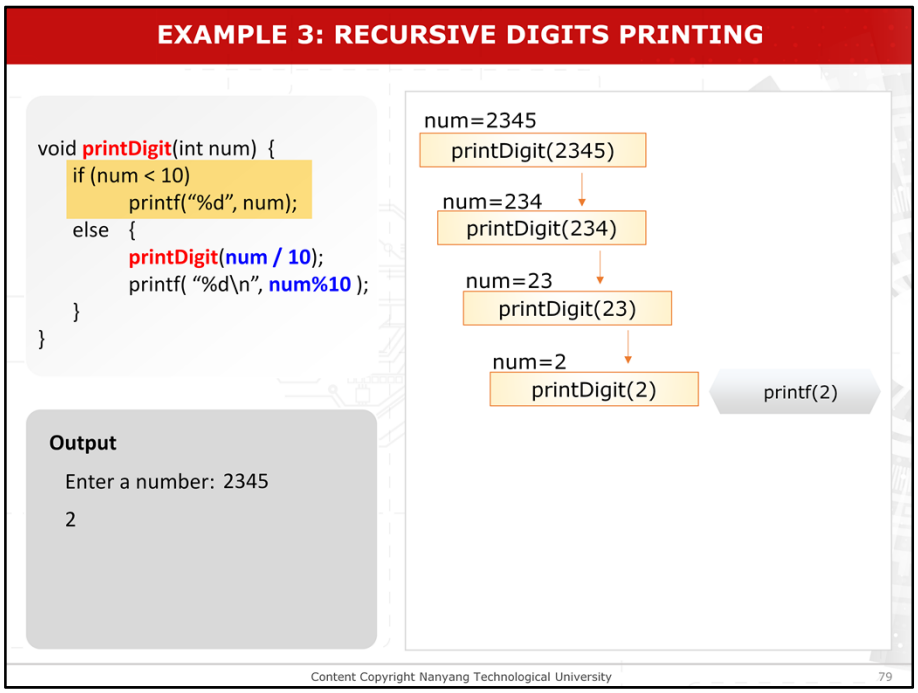

The function **print Digit (numb divided by 10)** will be called with the quotient **numb** is equal to 23 divided by 10 is equal to **2**.



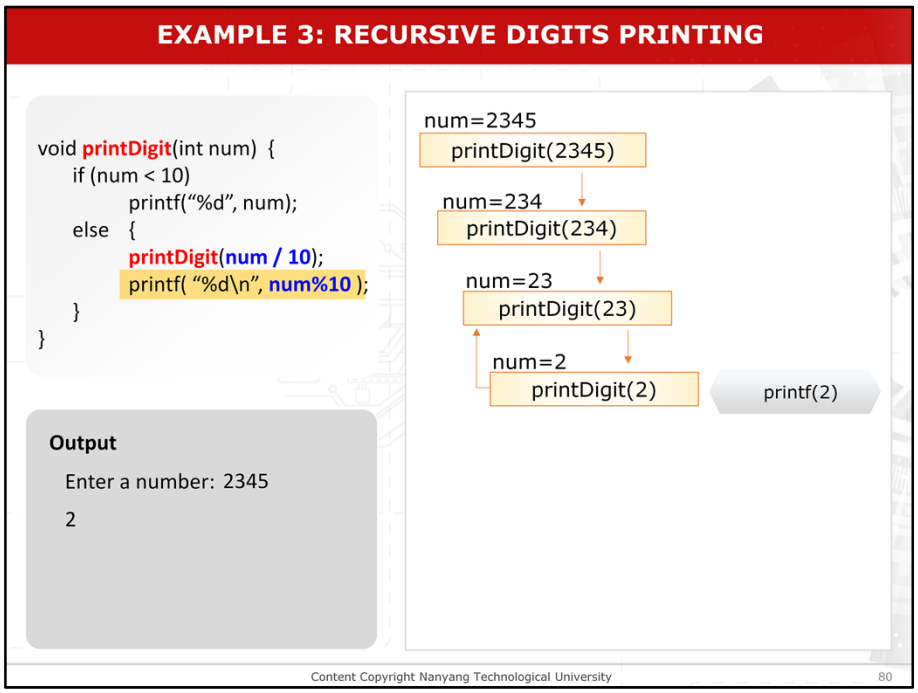

Similarly, the function **print Digit (2)** is called again.



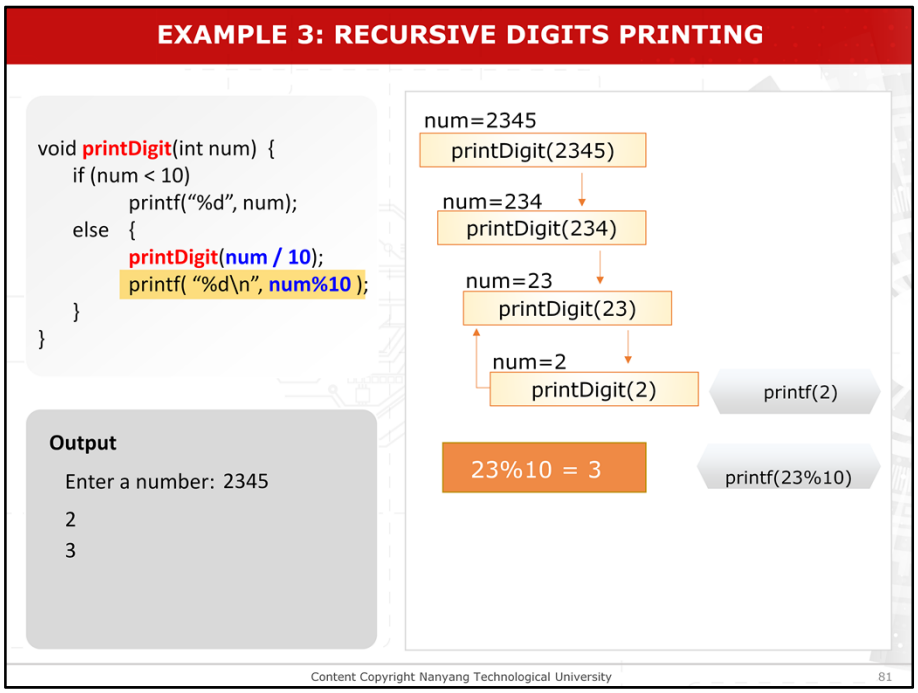

When this function is called, it reaches the terminating condition with only a single digit.



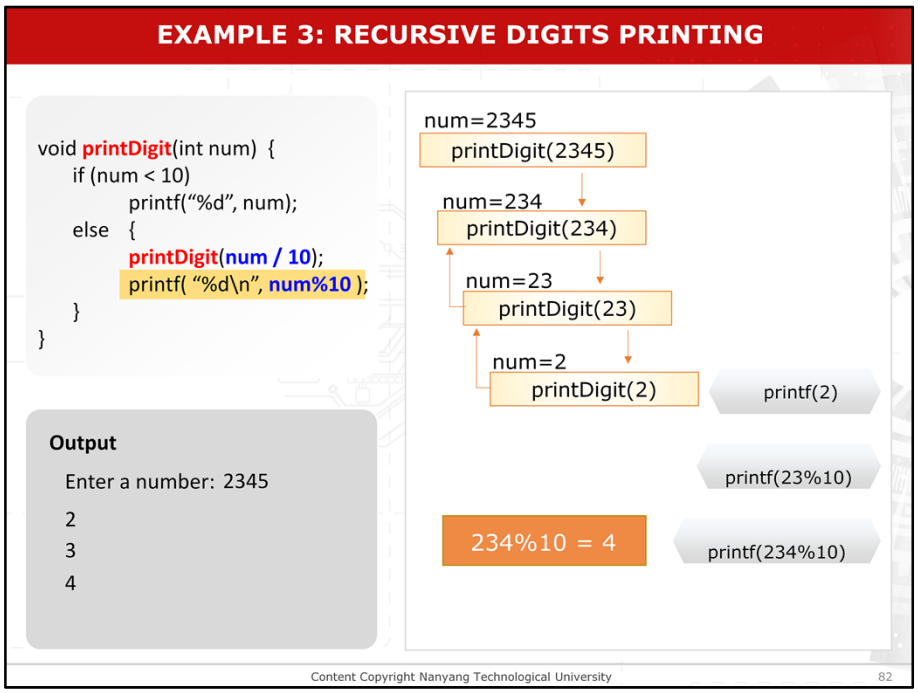

The value 2 is printed to the screen.



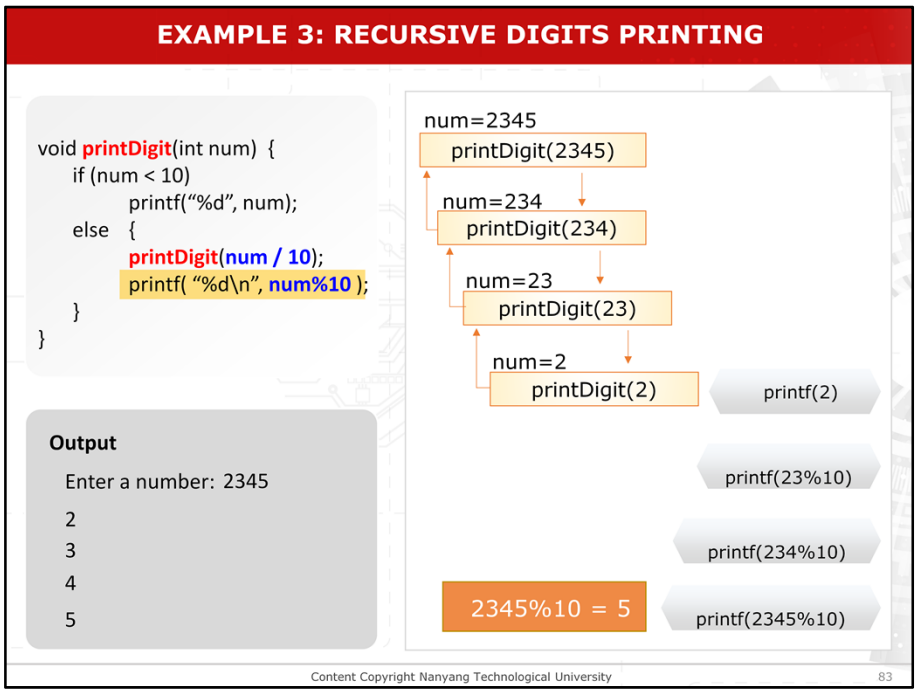

The control is then returned to the previous calling function **print Digit (23)**.



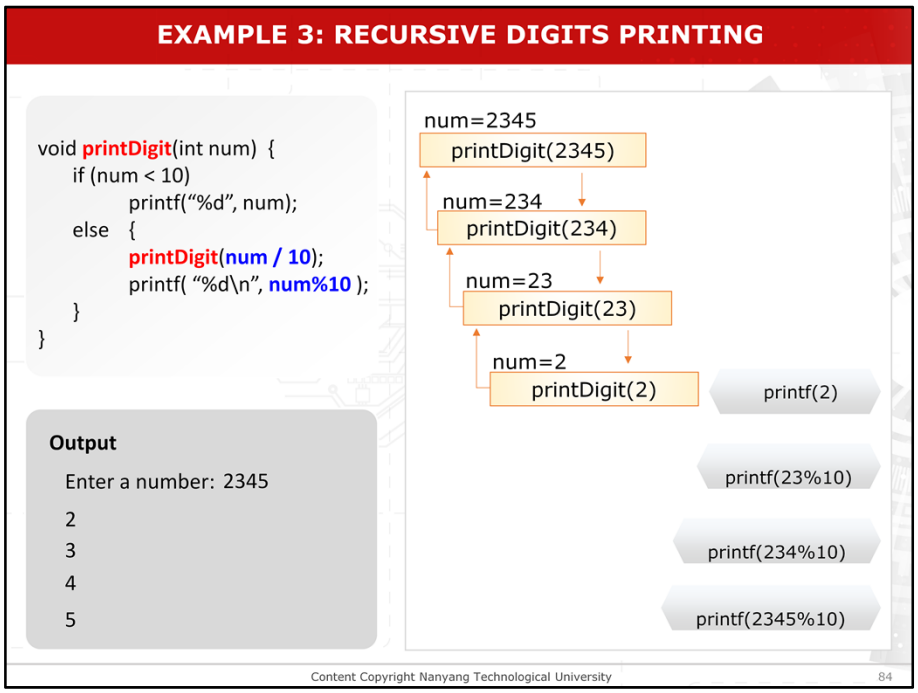

The function continues execution and prints the digit 3 to the screen because 23 modulus 10 is equal to 3.




Similarly, the digits 4 and 5 will then be printed when the control is returned to the previous calling functions.


EXAMPLE 3: RECURSIVE DIGITS PRINTING
void printDigit(int num) {
if (num < 10)
printf("%d", num);
else {
printDigit(num / 10);
printf( "%d\n", num%10 );
}
}
Output
Enter a number: 2345
2
3
4
5
num=2345
printDigit(2345)
num=234
printDigit(234)
num=23
printDigit(23)
num=2
printDigit(2)
printf(2)
printf(23%10)
printf(234%10)
2345%10 = 5
printf(2345%10)
Content Copyright Nanyang Technological University
83

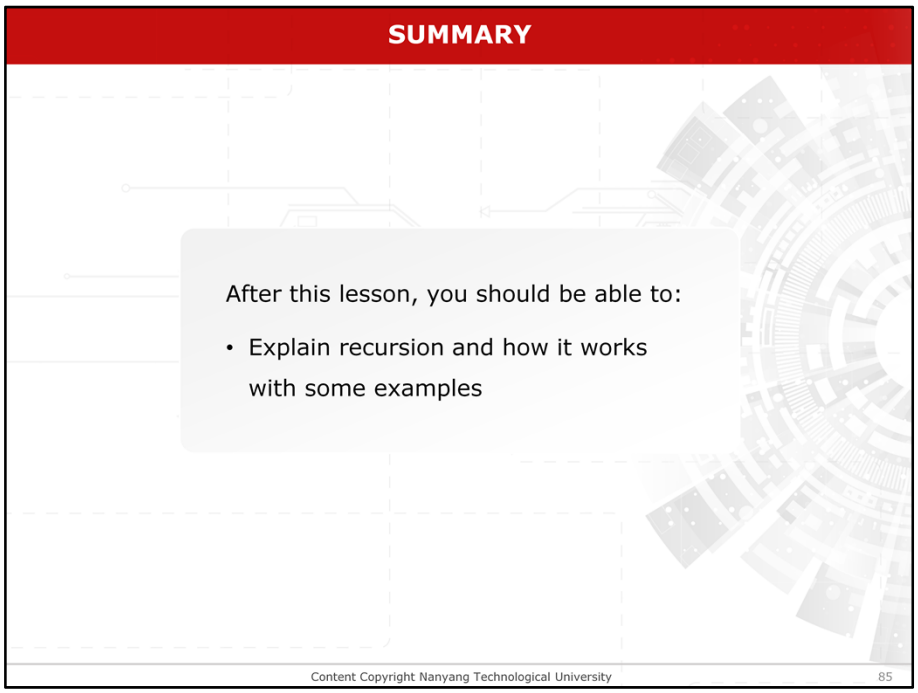
[no audio]




In this program, it is important to observe the execution order of the statements when the function has completed execution.

The control will be returned to the previous calling function and continues execution. It is also important to observe that the **print f()** statement is placed after the recursive function call in this program.

And also observe that each call to a function has its own set of values for the actual arguments and local variables.




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

We will look into more examples of recursion in the next video lecture.