

OVERVIEW

The following are the coverage for Character Strings:

- String Declaration, Initialization and Operations
- **String Input and Output**
- String Functions
- The ctype.h Character Functions
- String to Number Conversions
- Arrays of Character Strings

Content Copyright Nanyang Technological University

2

There are 6 main sections to cover for Character Strings as shown. This video lesson focuses on the second part on String Input and Output

LESSON OBJECTIVES

After this lesson, you should be able to:

After this lesson, you should be able to:

LESSON OBJECTIVES

After this lesson, you should be able to:

- Apply gets() to read a string

Apply get s() to read a string

LESSON OBJECTIVES

After this lesson, you should be able to:

- Apply gets() to read a string
- Apply puts() to print a string

Apply puts() to print a string

LESSON OBJECTIVES

After this lesson, you should be able to:

- Apply gets() to read a string
- Apply puts() to print a string
- Apply scanf() and printf() in a program

Apply scan f() and print f() in a program

LESSON OBJECTIVES

After this lesson, you should be able to:

- Apply gets() to read a string
- Apply puts() to print a string
- Apply scanf() and printf() in a program
- Apply printf() to print output

Apply print f() to print output

LESSON OBJECTIVES

After this lesson, you should be able to:

- Apply gets() to read a string
- Apply puts() to print a string
- Apply scanf() and printf() in a program
- Apply printf() to print output
- Compute string length

Content Copyright Nanyang Technological University

8

Compute string length

STRING INPUT

Reading strings

Two most commonly used standard library functions for reading strings:

- `gets()`
- `scanf()`

Content Copyright Nanyang Technological University

9

The two most commonly used standard library functions for reading strings are **get s()** and **scan f()**.

STRING OUTPUT

Printing strings

Two most commonly used standard library functions for printing strings:

- `puts()`
- `printf()`

Content Copyright Nanyang Technological University

10

For printing strings, the two standard library functions are `put s()` and `print f()`.

STRING INPUT

Creating memory space

Before reading a string, it is important to allocate enough storage space to store the string.

Content Copyright Nanyang Technological University

11

Before reading a string, it is important to allocate enough storage space to store the string.

STRING INPUT

Creating memory space

Before reading a string, it is important to allocate enough storage space to store the string.

Example: `char name[80];`

Content Copyright Nanyang Technological University

12

To create space for a string, we may include an explicit array size as shown in the following declaration:

character name [80];

STRING INPUT

Creating memory space

Before reading a string, it is important to allocate enough storage space to store the string.

Example: `char name[80];`
 `^`
 character array name

Content Copyright Nanyang Technological University

13

This declaration statement creates a character array **name**

STRING INPUT

Creating memory space

Before reading a string, it is important to allocate enough storage space to store the string.

Example: `char name[80];`

80 elements
character array name

Content Copyright Nanyang Technological University

14

of 80 elements.

Once the storage space has been acquired, we can read in the string through the C library functions.

STRING INPUT

gets() Function

The gets() function gets a string from the standard input device.



Content Copyright Nanyang Technological University

15

The gets() function gets a string from the standard input device.

STRING INPUT

gets() Function

Newline character:

- '\n'

Content Copyright Nanyang Technological University

16

It reads characters until it reaches a new line character represented by backlash n

STRING INPUT

gets() Function

Newline character:

- '\n'
- generated when the <Enter> key is pressed

Some input text \n



Content Copyright Nanyang Technological University

17

A newline character is generated when the <Enter> key is pressed.

STRING INPUT

gets() Function

The gets() function reads all the characters up to and including the newline character, replaces the newline character with a null character and passes them to the calling function as a string.

Some input text \0
string

Newline character change to null character

Content Copyright Nanyang Technological University 18

The **get s()** function reads all the characters up to and including the newline character, replaces the newline character with a null character and passes them to the calling function as a string.

STRING INPUT: gets()

```
#include <stdio.h>
int main()
{
    char name[80]; // allocate memory

    /*read name*/
    printf("Hi, what is your name?\n");

    gets(name);

    /*display name*/
    printf("Nice name, %s.\n", name);

    return 0;
}
```

Content Copyright Nanyang Technological University

19

In this program, it will read a string and prints the string to the screen.

STRING INPUT: gets()

```
#include <stdio.h>
int main()
{
    char name[80]; // allocate memory

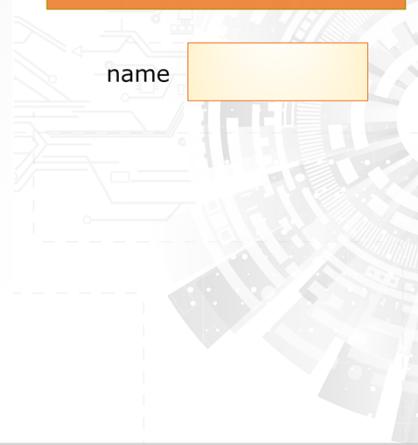
    /*read name*/
    printf("Hi, what is your name?\n");

    gets(name);

    /*display name*/
    printf("Nice name, %s.\n", name);

    return 0;
}
```

Make sure enough memory space is allocated to hold the input string, name



Content Copyright Nanyang Technological University

20

It is important to ensure that the array size of **name** is big enough to hold the input string. Otherwise, the extra characters can overwrite the adjacent memory variables.

STRING INPUT: gets()

```
#include <stdio.h>
int main()
{
    char name[80]; // allocate memory
    /*read name*/
    printf("Hi, what is your name?\n");

    gets(name);

    /*display name*/
    printf("Nice name, %s.\n", name);

    return 0;
}
```

name

Output

Hi, what is your name?

Content Copyright Nanyang Technological University 21

The system print message to prompt user to key in input.

STRING INPUT: gets()

```
#include <stdio.h>
int main()
{
    char name[80]; // allocate memory
    /*read name*/
    printf("Hi, what is your name?\n");

    gets(name);

    /*display name*/
    printf("Nice name, %s.\n", name);

    return 0;
}
```

name Hui Siu Cheung\0

User Input

Hi, what is your name?
Hui Siu Cheung\n

Content Copyright Nanyang Technological University 22

The user keys in and the characters are read.

The **get s()** function reads all the characters up to and including the newline character.

The **get s()** function replaces the new line character with a null character and passes them to the calling function as a string.

STRING INPUT: gets()

```
#include <stdio.h>
int main()
{
    char name[80]; // allocate memory
    /*read name*/
    printf("Hi, what is your name?\n");

    gets(name);

    /*display name*/
    printf("Nice name, %s.\n", name);

    return 0;
}
```

gets() returns Null if it fails,
otherwise a pointer to the
string is returned.

name Hui Siu Cheung\0

User Input

Hi, what is your name?

Hui Siu Cheung

STRING INPUT: gets()

```
#include <stdio.h>
int main()
{
    char name[80]; // allocate memory
    /*read name*/
    printf("Hi, what is your name?\n");

    gets(name);

    /*display name*/
    printf("Nice name, %s.\n", name);

    return 0;
}
```

name Hui Siu Cheung\0

Output

Hi, what is your name?
Hui Siu Cheung
Nice name, Hui Siu Cheung.

Content Copyright Nanyang Technological University 24

The string is then printed out.

STRING INPUT: gets()

gets() Function

- Function prototype of gets() is

```
char *gets(char *ptr);
```

Content Copyright Nanyang Technological University

25

The function prototype of **get s()** is
character asterisk get s (character asterisk pointer);

STRING INPUT: gets()

gets() Function

- Function prototype of gets() is
`char *gets(char *ptr);`
- gets() function returns a **null pointer** if it **fails**,
otherwise the same pointer ptr is returned.

Content Copyright Nanyang Technological University

26

The **get s()** function returns a null pointer if it fails, otherwise the same pointer is returned.

STRING INPUT: gets()

gets() Function

- Function prototype of gets() is
`char *gets(char *ptr);`
- gets() function returns a **null pointer** if it **fails**,
otherwise the same pointer ptr is returned.
- Important to make sure that sufficient storage is
allocated to hold the input string.

Content Copyright Nanyang Technological University

27

It is important to make sure that sufficient storage is allocated to hold the input string.

STRING INPUT: gets()

gets() Function

- **Null pointer** contains a null address that is the symbolic constant **NULL** defined in the header file *stdio.h*.

Content Copyright Nanyang Technological University

28

The null pointer contains a null address that is the symbolic constant **NULL** defined in the header file *standard i o dot h*

STRING INPUT: gets()

gets() Function

- **Null pointer** contains a null address that is the symbolic constant NULL defined in the header file stdio.h.
- Different from the null character ('\0') in that the null character is a data object of type char with ASCII value 0, whereas the **null pointer** contains an address.

Content Copyright Nanyang Technological University

29

This is different from the null character in that the null character is a data object of type **character** with ASCII value 0, whereas the null pointer contains an address.

STRING INPUT: gets()

gets() Function

- Size of the null character is 1 byte
- Size of **null pointer** is 4 bytes long

Content Copyright Nanyang Technological University

30

In addition, the size of the null character is 1 byte, while the null pointer is 4 bytes long.

STRING OUTPUT: puts()

```
#include <stdio.h>
int main( )
{
    //string with allocated memory
    char str[80];

    printf("Enter a line of string: \n");
    if (gets(str) == NULL) {
        printf("Error\n");
    }

    puts(str);

    return 0;
}
```

Content Copyright Nanyang Technological University

31

reads in
a string using the **get s()** function and prints the string on the screen using the
put s() function.

STRING OUTPUT: puts()

```
#include <stdio.h>
int main( )
{
    //string with allocated memory
    char str[80];

    printf("Enter a line of string: \n");
    if (gets(str) == NULL) {
        printf("Error\n");
    }

    puts(str);

    return 0;
}
```

str

Array declared can hold
up to 80 elements

Content Copyright Nanyang Technological University

32

STRING OUTPUT: puts()

```
#include <stdio.h>
int main( )
{
    //string with allocated memory
    char str[80];

    printf("Enter a line of string: \n");
    if (gets(str) == NULL) {
        printf("Error\n");
    }

    puts(str);

    return 0;
}
```

str

User Input

Enter a line of string:

Content Copyright Nanyang Technological University

33

STRING OUTPUT: puts()

```
#include <stdio.h>
int main( )
{
    //string with allocated memory
    char str[80];

    printf("Enter a line of string: \n");
    if (gets(str) == NULL) {
        printf("Error\n");
    }

    puts(str);

    return 0;
}
```

str 0 123456789 OK \0

User Input

Enter a line of string:

0 123456789 OK \n

Content Copyright Nanyang Technological University

34

The **get s()** function returns a null pointer if it fails, otherwise the same pointer is returned.

STRING OUTPUT: puts()

```
#include <stdio.h>
int main( )
{
    //string with allocated memory
    char str[80];

    printf("Enter a line of string: \n");
    if (gets(str) == NULL) {
        printf("Error\n");
    }

    puts(str);

    return 0;
}
```

str 0 123456789 OK \0

Output

```
Enter a line of string:
0 123456789 OK
0 123456789 OK
```

Content Copyright Nanyang Technological University

35

prints a string on the standard output device.

STRING OUTPUT: puts()

puts() Function

- The function prototype for the puts() function is

```
int puts(const char *ptr);
```

Content Copyright Nanyang Technological University

36

The function prototype for the puts() function is

```
integer puts (constant character asterisk pointer);
```

STRING OUTPUT: puts()

puts() Function

- The function prototype for the puts() function is

```
int puts(const char *ptr);
```
- Prints a string on the standard output device.

Content Copyright Nanyang Technological University

37

It prints a string on the standard output device.

STRING OUTPUT: puts()

puts() Function

- A newline character is automatically added to the end of the string. Thus, the newline character is printed after the string. EOF is returned if puts() fails, otherwise the number of characters written will be returned.

Content Copyright Nanyang Technological University

38

A newline character is automatically added to the end of the string. Thus, the *newline* character is printed after the string. **E O F** is returned if **put s()** fails, otherwise the number of characters written will be returned.

STRING OUTPUT: puts()

puts() Function

- In the program, it reads in a string using the gets() function and prints the string on the screen using the puts() function.

Content Copyright Nanyang Technological University

39

In the program, it reads in a string using the **get s()** function and prints the string on the screen using the **put s()** function.

STRING INPUT: `scanf()`

`scanf()` Function

- Reads the string up to the next **whitespace** character.

The `scanf()` function reads the string up to the next whitespace character.

STRING INPUT: `scanf()`

`scanf()` Function

- Reads the string up to the next **whitespace** character.
- %s format specification can be used to read in a string.

Content Copyright Nanyang Technological University

41

The **scanf()** function with the **percentage s** format specification can be used to read in a string.

STRING INPUT: `scanf()`

`scanf()` Function

- Reads the string up to the next **whitespace** character.
- %s format specification can be used to read in a string.
e.g. `scanf("%s", str);`

Content Copyright Nanyang Technological University

42

An example of the **scanf()** function to read in a string is given as follows:

scanf (double quote percentage s double quote, string);

STRING INPUT: `scanf()`

`scanf()` Function

- Reads the string up to the next **whitespace** character.
- %s format specification can be used to read in a string.
e.g. `scanf("%s", str);`
- Two ways to read input string in `scanf()`:

Content Copyright Nanyang Technological University

43

There are two ways to read input string in **scanf()**.

STRING INPUT: scanf()

scanf() Function

- Reads the string up to the next **whitespace** character.
- %s format specification can be used to read in a string.
e.g. `scanf("%s", str);`
- Two ways to read input string in scanf():
 - **%s** conversion specifier

Content Copyright Nanyang Technological University

44

We may use either a **percentage s** conversion specifier

STRING INPUT: `scanf()`

`scanf()` Function

- Reads the string up to the next **whitespace** character
- **%s** format specification can be used to read in a string
 - e.g. `scanf("%s", str);`
- Two ways to read input string in `scanf()`:
 - **%s** conversion specifier
 - **%ns** conversion specifier, where **n** is the field width specification

Content Copyright Nanyang Technological University

45

or **percentage n s** conversion specifier, where **n** is the field width specification.

STRING INPUT: scanf()

scanf() Function

- %s or %ns conversion specifier: reading starts at the first non-whitespace character.

Content Copyright Nanyang Technological University

46

In both cases, reading starts at the first non-whitespace character.

STRING INPUT: scanf()

scanf() Function

- **%s** or **%ns** conversion specifier: reading starts at the first non-whitespace character.
- **%ns** conversion specifier:

Content Copyright Nanyang Technological University

47

When **percentage n s** is used,

STRING INPUT: `scanf()`

`scanf()` Function

- `%s` or `%ns` conversion specifier: reading starts at the first non-whitespace character.
- `%ns` conversion specifier: `scanf()` reads up to n input characters or when the first whitespace character (i.e. blank, tab or newline) is encountered.

Content Copyright Nanyang Technological University

48

Scan f() reads up to n input characters or when the first whitespace character that is when blank, tab or new line is encountered.

STRING INPUT: scanf()

scanf() Function

- Example **%6s**:

Content Copyright Nanyang Technological University 49

For example, **percentage 6 s**

STRING INPUT: scanf()

scanf() Function

- Example `%6s`: will stop after the first six characters are read.

Content Copyright Nanyang Technological University

50

will stop after the first six characters are read

STRING INPUT: scanf()

scanf() Function

- Example `%6s`: will stop after the first six characters are read.
- `%s`:

Content Copyright Nanyang Technological University

51

When **percentage s** is specified,

STRING INPUT: scanf()

scanf() Function

- Example `%6s`: will stop after the first six characters are read.
- `%s`: `scanf()` function reads the string up to the next whitespace character (i.e. blank, tab or newline).

Content Copyright Nanyang Technological University

52

the `scanf()` function reads the string up to the next whitespace character.

STRING INPUT: `scanf()`

`scanf()` Function

- `scanf()` returns EOF if fails, otherwise the number of items read by `scanf()`.

Content Copyright Nanyang Technological University

53

This function will return **E O F** if it fails, otherwise the number of items read by **scanf()** will be returned.

STRING INPUT: scanf()

scanf() Function

- scanf() returns EOF if fails, otherwise the number of items read by scanf().
- Unlike other data input, there is no need to have the address-of (&) operator to be placed before the name of the array because the array name is the address of the first element of the array.

Content Copyright Nanyang Technological University

54

Unlike other data input, there is no need to have the address-of (**ampersand**) operator to be placed before the name of the array because the array name is the address of the first element of the array.

STRING INPUT: scanf()

scanf() Function

- Important to ensure that sufficient storage is allocated to hold the string.

Content Copyright Nanyang Technological University

55

It is important to ensure that **sufficient storage** is allocated to hold the string.

STRING INPUT: scanf()

scanf() Function

- Important to ensure that sufficient storage is allocated to hold the string.
- If more characters are read in than the storage space allocated to hold the string, the additional characters will be overwritten to the adjacent memory locations.

Content Copyright Nanyang Technological University

56

If more characters are read in than the storage space allocated to hold the string, the additional characters will be overwritten to the adjacent memory locations.

STRING INPUT: scanf()

scanf() Function

- For example, consider the following declaration statements:

Content Copyright Nanyang Technological University

57

For example, consider the following declaration statements:

STRING INPUT: scanf()

scanf() Function

- For example, consider the following declaration statements:

```
char str[] = "string space";
```

Content Copyright Nanyang Technological University

58

character string = double quote string space double quote;

STRING INPUT: scanf()

scanf() Function

- For example, consider the following declaration statements:

```
char str[] = "string space";  
scanf("%s", str);
```

Content Copyright Nanyang Technological University

59

scanf(double quote percentage s double quote, string);

STRING INPUT: scanf()

scanf() Function

- For example, consider the following declaration statements:

```
char str[] = "string space";  
scanf("%s", str);
```

- String pointer constant is created, and 13 bytes of storage are created to hold the string constant.

Content Copyright Nanyang Technological University

60

A string pointer constant is created, and 13 bytes of storage are created to hold the string constant.

STRING INPUT: scanf()

scanf() Function

- For example, consider the following declaration statements:

```
char str[] = "string space";  
scanf("%s", str);
```

- If the string read in by the scanf() function is longer than 13 bytes, it will be overwritten to the adjacent memory locations.

Content Copyright Nanyang Technological University

61

If the string read in by the **scanf()** function is longer than 13 bytes, it will be overwritten to the adjacent memory locations.

STRING OUTPUT: printf()

printf() Function

- Function prototype for the printf() function is

```
printf(control-string, argument-list);
```

Content Copyright Nanyang Technological University

62

The function prototype for the **print f()** function is
print f (control-string, argument-list);

STRING OUTPUT: printf()

printf() Function

- Function prototype for the printf() function is
`printf(control-string, argument-list);`
- Prints formatted output on the standard output device.

Content Copyright Nanyang Technological University

63

It prints formatted output on the standard output device.

STRING OUTPUT: printf()

printf() Function

- Function prototype for the printf() function is
`printf(control-string, argument-list);`
- Prints formatted output on the standard output device.
- Returns a negative value if printf() fails, otherwise the
number of characters printed will be returned.

Content Copyright Nanyang Technological University

64

It returns a negative value if print f() fails, otherwise the number of characters printed will be returned.

STRING OUTPUT: printf()

printf() Function

- Differs from the puts() function in that no newline is added at the end of the string.

Content Copyright Nanyang Technological University

65

It differs from the **put s()** function in that no newline is added at the end of the string.

STRING OUTPUT: printf()

printf() Function

- Differs from the puts() function in that no newline is added at the end of the string.
- Less convenient to use than the puts() function.

Content Copyright Nanyang Technological University

66

The **printf()** function is less convenient to use than the **puts()** function

STRING OUTPUT: printf()

printf() Function

- Differs from the puts() function in that no newline is added at the end of the string.
- Less convenient to use than the puts() function.
- Provides the flexibility to the user to control the format of the data to be printed.

Content Copyright Nanyang Technological University

67

However, the **printf()** function provides the flexibility to the user to control the format of the data to be printed.

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```

Content Copyright Nanyang Technological University

68

This example will show the use of scan f() and print f() function.

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```



Content Copyright Nanyang Technological University

69

Character array string of size 20 are declared for name1, name2 and name3

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```



Content Copyright Nanyang Technological University

70

Integer count is declared as a variable to keep count.

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```

Output

Please enter your strings.



Content Copyright Nanyang Technological University

71

String Please enter your strings printed on screen to prompt user input

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

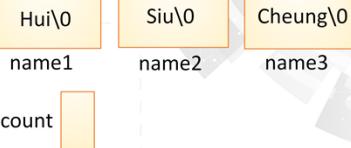
    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```

Input

Please enter your strings.
Hui Siu Cheung



Content Copyright Nanyang Technological University

72

In the program, the `scanf()` function is used to read a string from the input.

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```

Input

Please enter your strings.
Hui Siu Cheung

Hui\0
name1
Siu\0
name2
Cheung\0
name3
count 3

Content Copyright Nanyang Technological University

73

The **scanf()** function returns the **count** on the number of input strings.

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```

Output

```
Please enter your strings.
Hui Siu Cheung
I read the 3 strings: Hui Siu Cheung.
```



Content Copyright Nanyang Technological University

74

The final print f give the output as shown.

scanf() AND printf(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    printf("Please enter your strings.\n");

    count = scanf("%s %s %s", name1, name2, name3);

    printf("I read the %d strings: %s %s %s.\n", count, name1, name2, name3);
    return 0;
}
```

Output

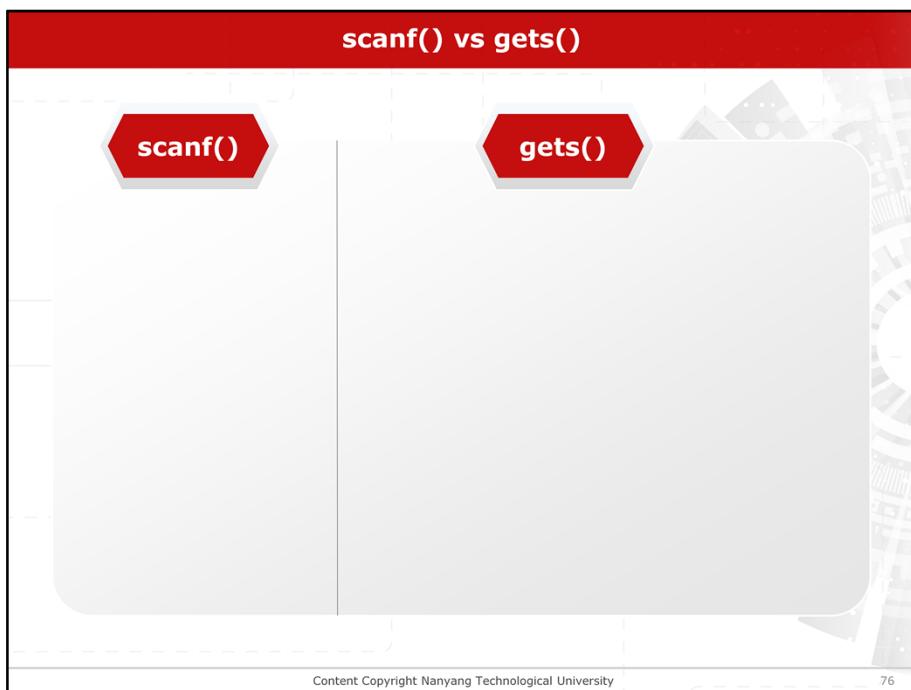
```
Please enter your strings.  
Hui Siu Cheung  
I read the 3 strings: Hui Siu Cheung.
```

Hui\0	Siu\0	Cheung\0
name1	name2	name3
count	3	

Content Copyright Nanyang Technological University

75

[audio pause]



Scan f() vs get s()

scanf() vs gets()

scanf()

- Less convenient to use than the gets() function for reading string input.

gets()

Content Copyright Nanyang Technological University

77

The **scanf()** function is less convenient to use than the **gets()** function for reading string input.

scanf() vs gets()

scanf()

- Less convenient to use than the gets() function for reading string input.

gets()

- Differs from the scanf() function in that gets() reads an entire line up to the first newline character.

Content Copyright Nanyang Technological University

78

The **get s()** function differs from the **scn f()** function in that **get s()** reads an entire line up to the first newline character

scanf() vs gets()

scanf()

- Less convenient to use than the gets() function for reading string input.

gets()

- Differs from the scanf() function in that gets() reads an entire line up to the first newline character.
- Stores any characters, including whitespace, up to the first newline character.

Content Copyright Nanyang Technological University

79

The **gets()** function also stores any characters, including whitespace, up to the first newline character.

scanf() vs gets()

scanf()

- Less convenient to use than the gets() function for reading string input.

gets()

- Differs from the scanf() function in that gets() reads an entire line up to the first newline character.
- Stores any characters, including whitespace, up to the first newline character.
- Faster than the scanf() function.

Content Copyright Nanyang Technological University

80

The **get s()** function is also faster than the **scn f()** function.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

Content Copyright Nanyang Technological University

81

In the program, it illustrates the difference between array and pointer notations for processing strings.

Two functions are implemented in the program.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

To compute the length of a string, function

length1() uses array notation

Content Copyright Nanyang Technological University

82

The function **length 1()** uses the array notation to compute the length of a string.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

To compute the length of a string, function

length1() uses array notation

length2() uses pointer notation

Content Copyright Nanyang Technological University

83

and the function **length 2()** uses the pointer notation to compute the length of a string.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

creates a pointer variable called greeting that points to the first character of the string "hello".



Content Copyright Nanyang Technological University

84

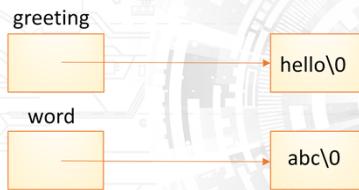
In the **main()** function, the declaration

character asterisk greeting = double quote hello double quote;
creates a pointer variable called **greeting** that points to the first character of the string "**hello**".

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

array of type char called word[] contains four elements including the null character



Content Copyright Nanyang Technological University

85

The declaration

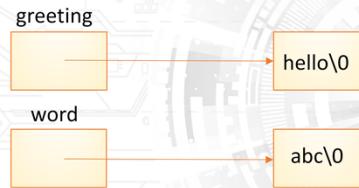
character array word[] = double quote a b c double quote;
creates an array of type **character** called **word[]**. This array contains four elements including the null character.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

In the function **length1()**, it uses the index notation for the implementation.

```
int length1(char string[])
{
    int count = 0;
    while (string[count] != '\0')
        count++;
    return(count);
}
```



Content Copyright Nanyang Technological University

86

In the function **length 1()**, it uses the index notation for the implementation.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

while loop statement used to check for the null character ('\0') while measuring the length of the string.

```
int length1(char string[])
{
    int count = 0;
    while (string[count] != '\0')
        count++;
    return(count);
}
```



Content Copyright Nanyang Technological University

87

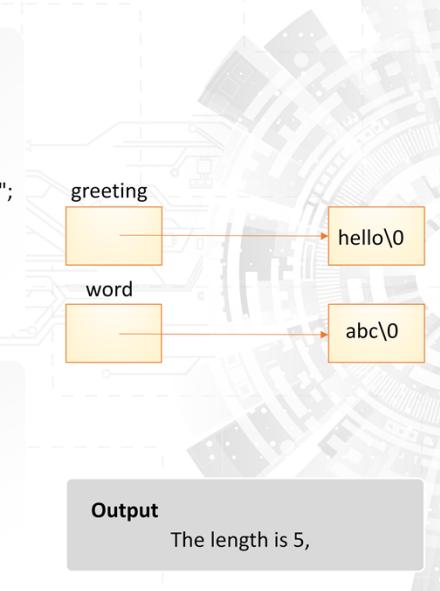
The function **length 1()** uses the statement
while (string[count] not = null character)

to check for the null character in the **while** loop while measuring the length of the string.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

```
int length1(char string[])
{
    int count = 0;
    while (string[count] != '\0')
        count++;
    return(count);
}
```



Content Copyright Nanyang Technological University

88

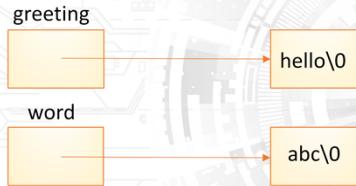
[audio pause]

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

In the function **length2()**, it uses the pointer notation for the implementation.

```
int length2(char *string)
{
    int count = 0;
    while (*(string+count) != '\0')
        count++;
    return(count);
}
```



Content Copyright Nanyang Technological University

89

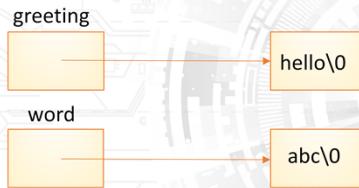
In the function **length 2()**, it uses the pointer notation for the implementation.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

while loop statement used to check for terminating null character ('\0')

```
int length2(char *string)
{
    int count = 0;
    while (*string+count) != '\0'
        count++;
    return(count);
}
```



Content Copyright Nanyang Technological University

90

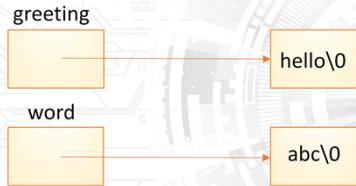
The function **length 2()** uses the statement
while (asterisk(string + count) not = null character)
to check for terminating null character.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

```
int length2(char *string)
{
    int count = 0;
    while (*string+count) != '\0'
        count++;
    return(count);
}
```

The expression `(*string + count)` first gets the character stored at the address location contained in `(string + count)`.



Content Copyright Nanyang Technological University

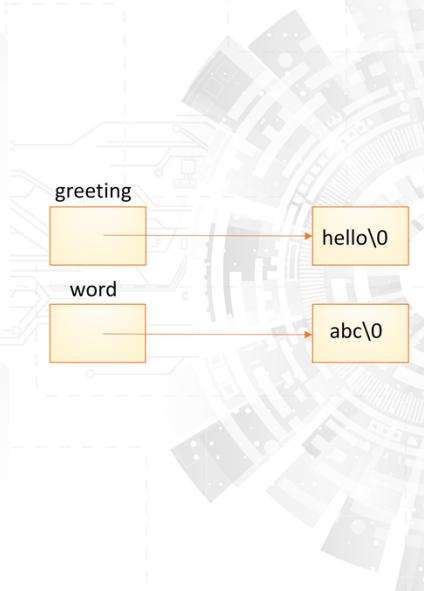
91

The expression `(asterisk(string + count))` first gets the character stored at the address location contained in `(string + count)`, and then tests whether it is a null character.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}

int length2(char *string)
{
    int count = 0;
    while (*(string+count) != '\0')
        count++;
    return(count);
}
```



Content Copyright Nanyang Technological University

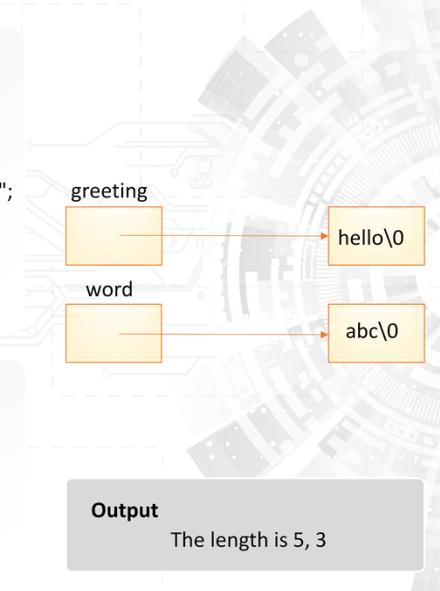
92

For any characters other than the null character, the condition will be true, and the statements in the body of the **while** loop will be executed to measure the length of the string.

STRING PROCESSING – COMPUTE STRING LENGTH

```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
           length1(greeting), length2(word));
    return 0;
}
```

```
int length2(char *string)
{
    int count = 0;
    while (*(string+count) != '\0')
        count++;
    return(count);
}
```



Output

The length is 5, 3

Content Copyright Nanyang Technological University

93

SUMMARY

After this lesson, you should be able to:

- Apply gets() to read a string
- Apply puts() to print a string
- Apply scanf() and printf() in a program
- Apply printf() to print output
- Compute string length

Content Copyright Nanyang Technological University

94

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