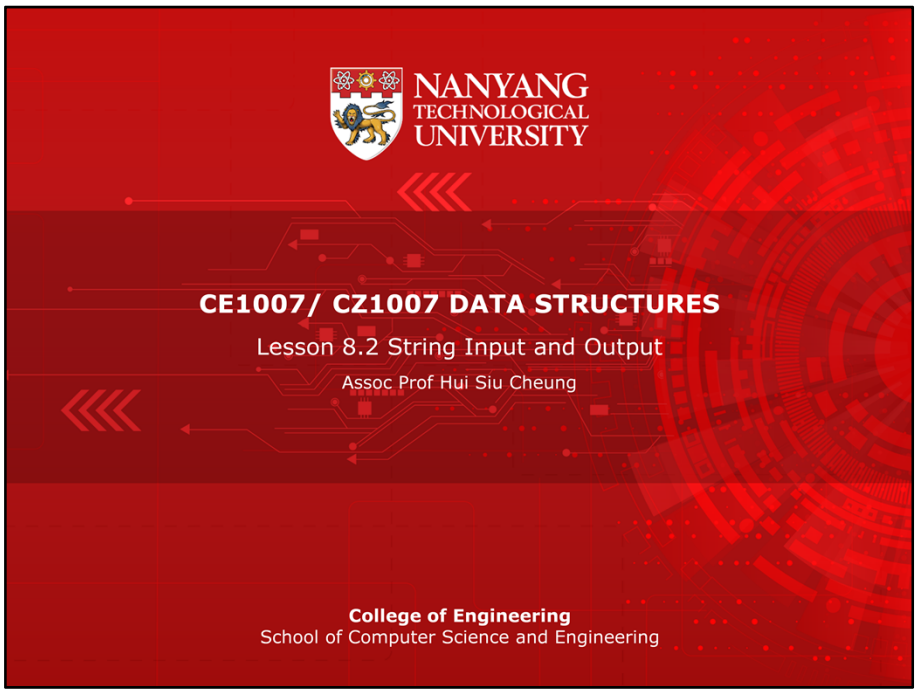
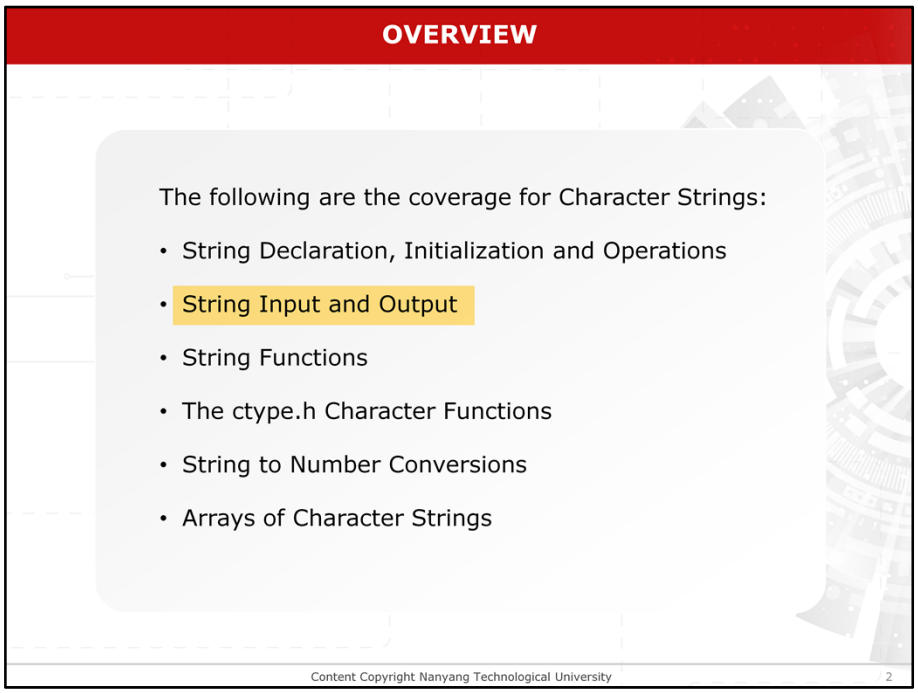
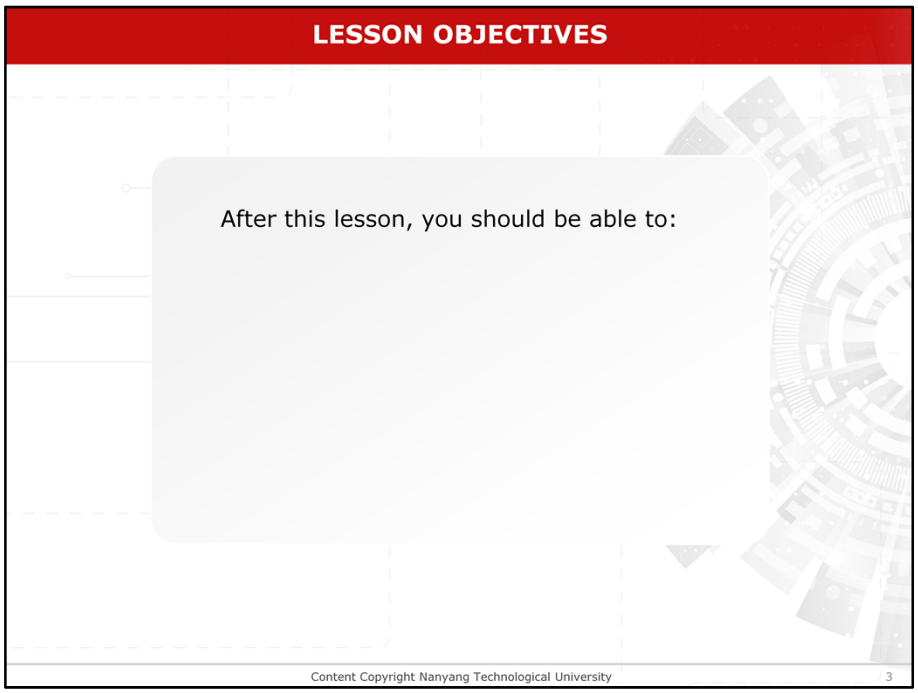
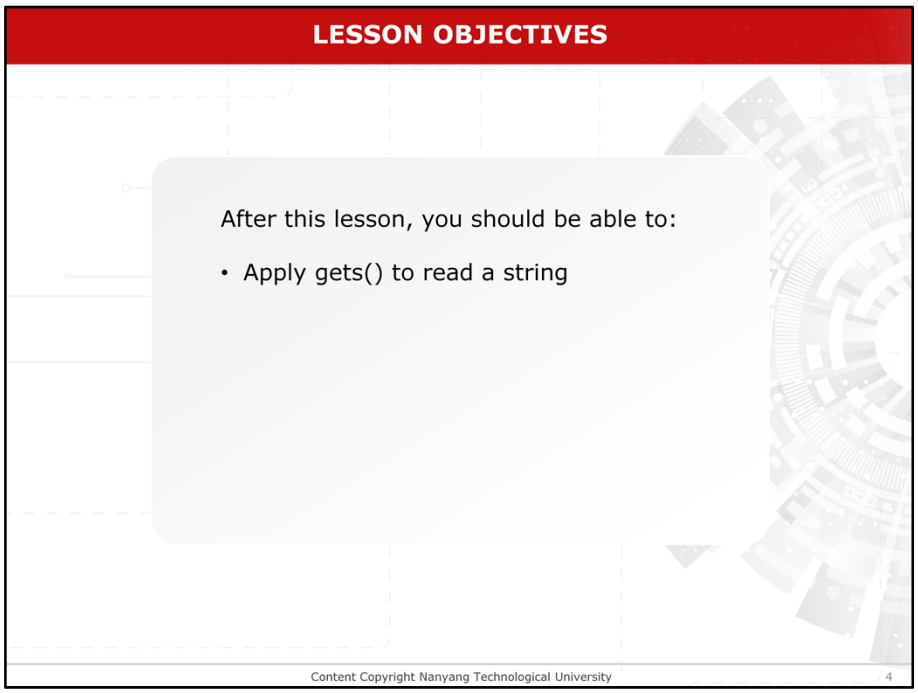

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 8.2 String Input and Output
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering



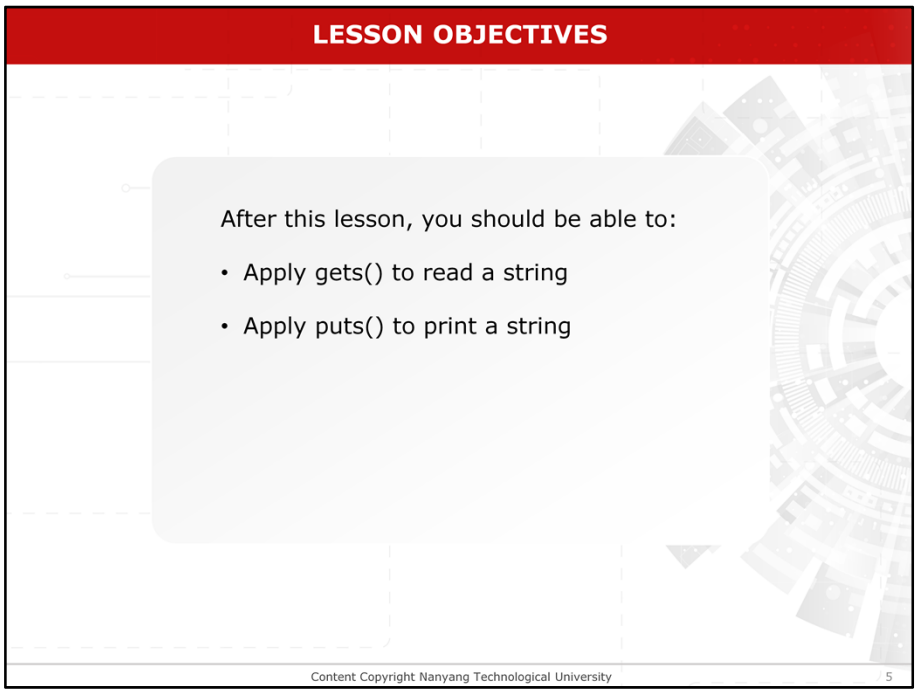
There are 6 main sections to cover for Character Strings as shown. This video lesson focuses on the second part on String Input and Output



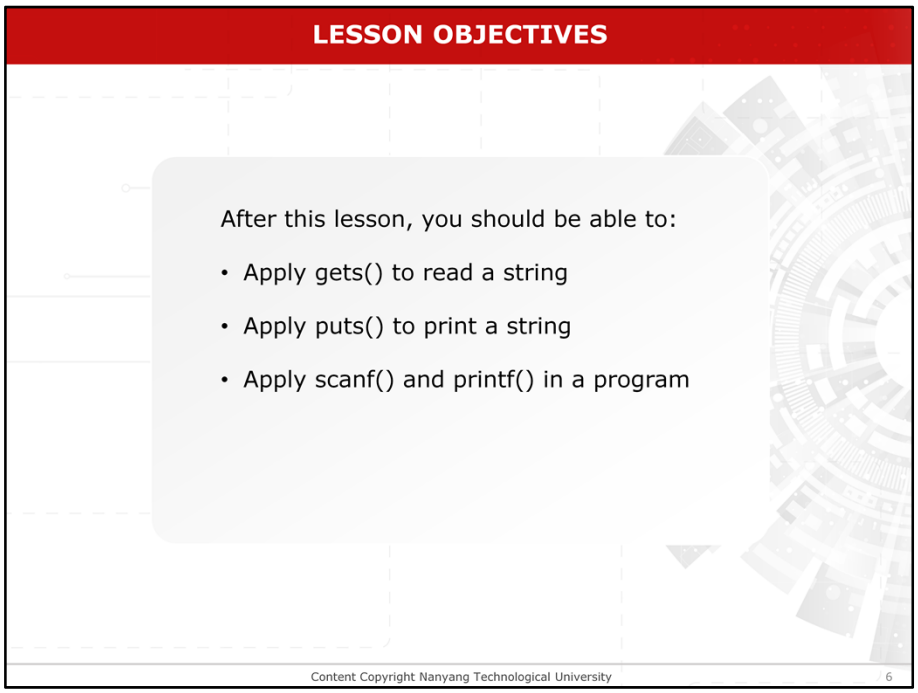
After this lesson, you should be able to:



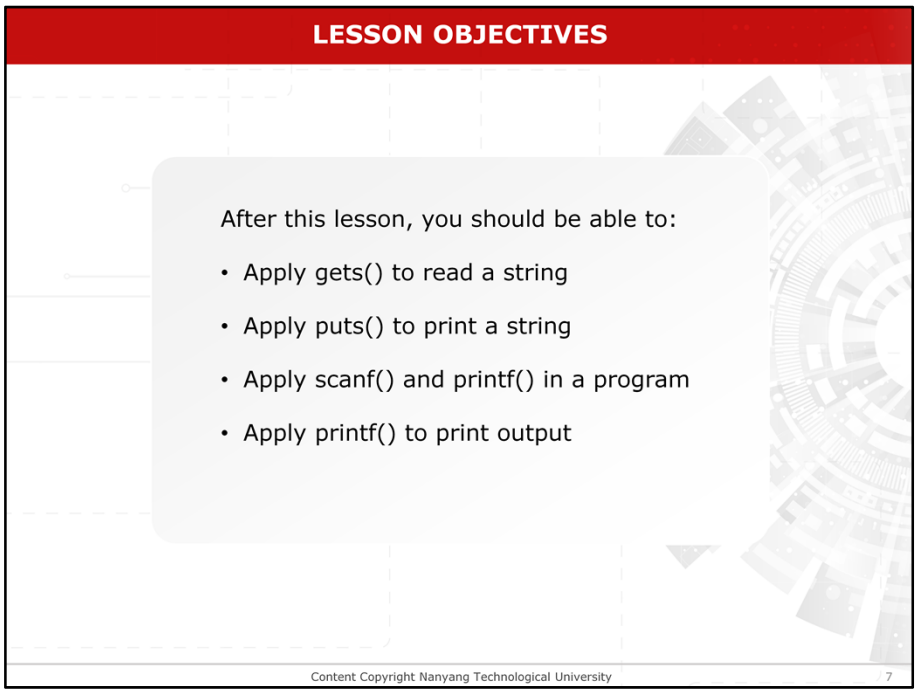
Apply get s() to read a string



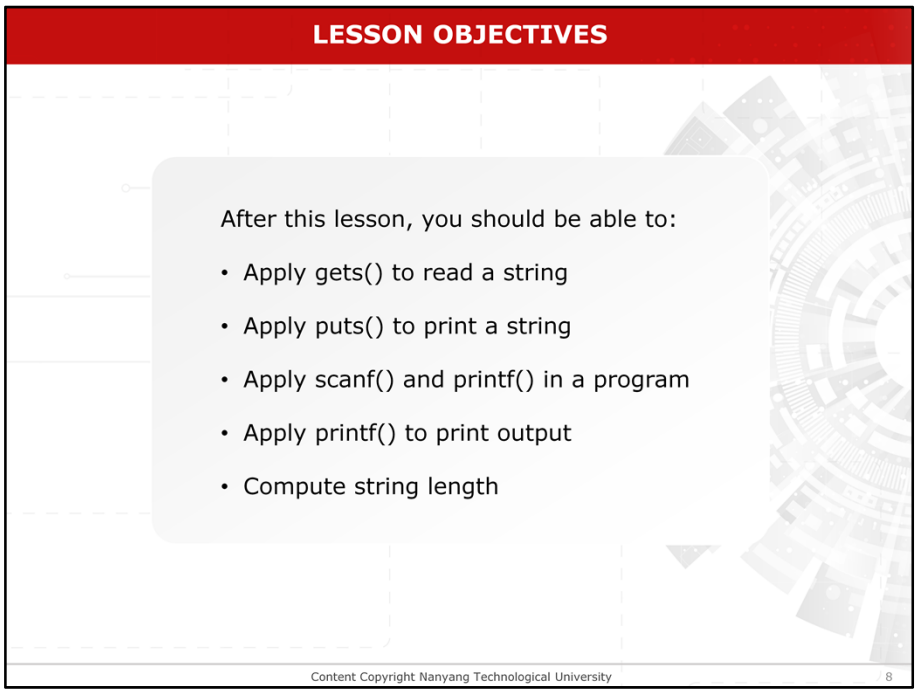
Apply put s() to print a string



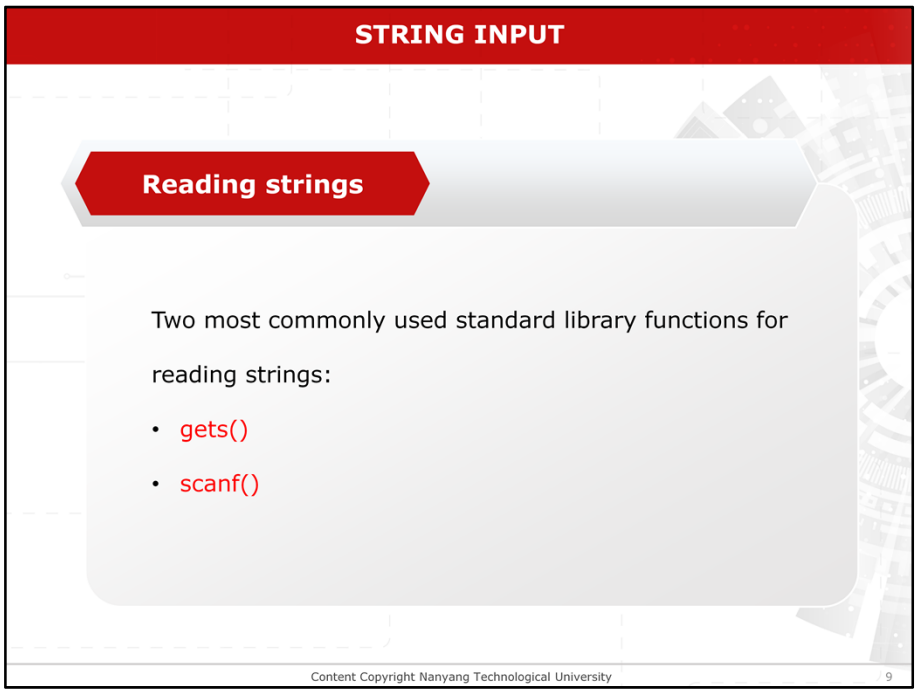
Apply scan f() and print f() in a program



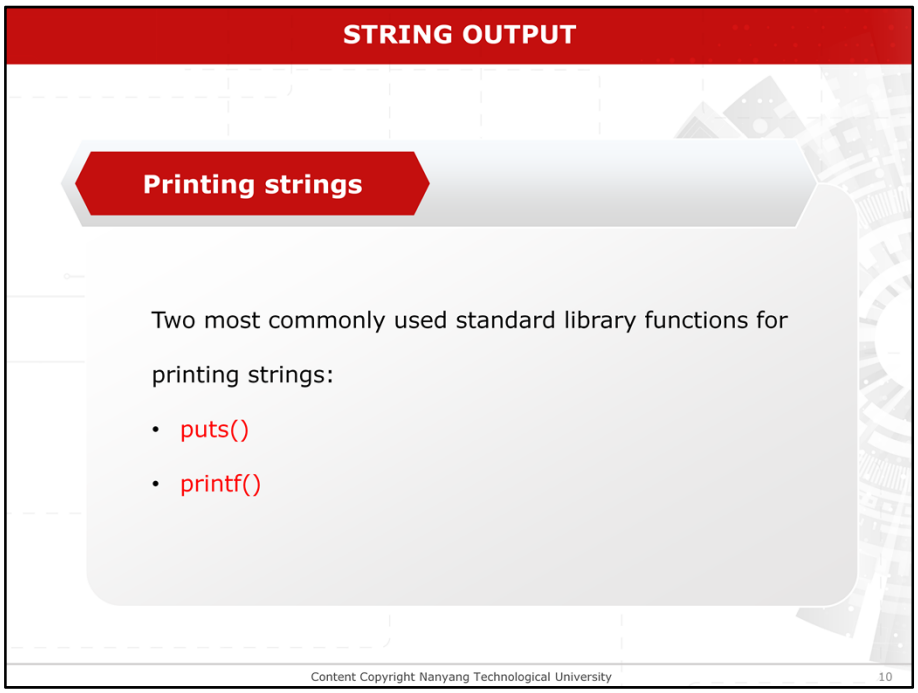
Apply print f() to print output



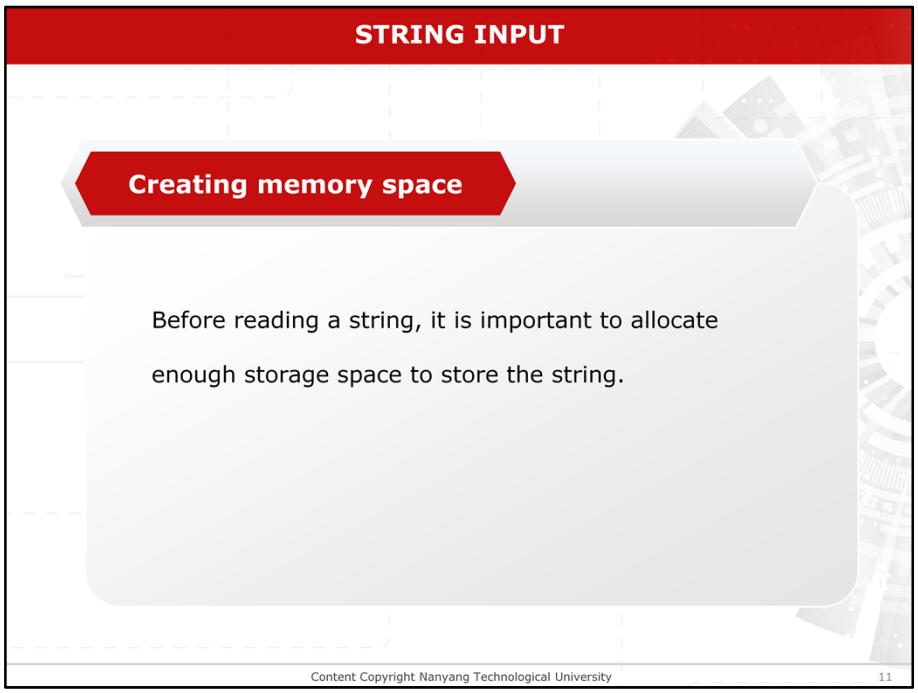
Compute string length



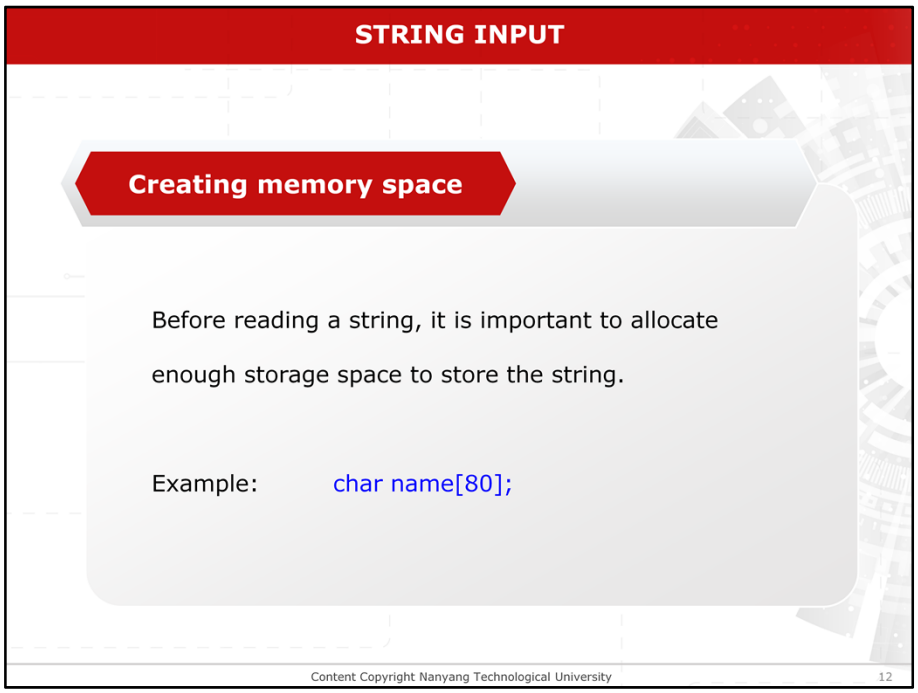
The two most commonly used standard library functions for reading strings are **get s()** and **scan f()**.



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

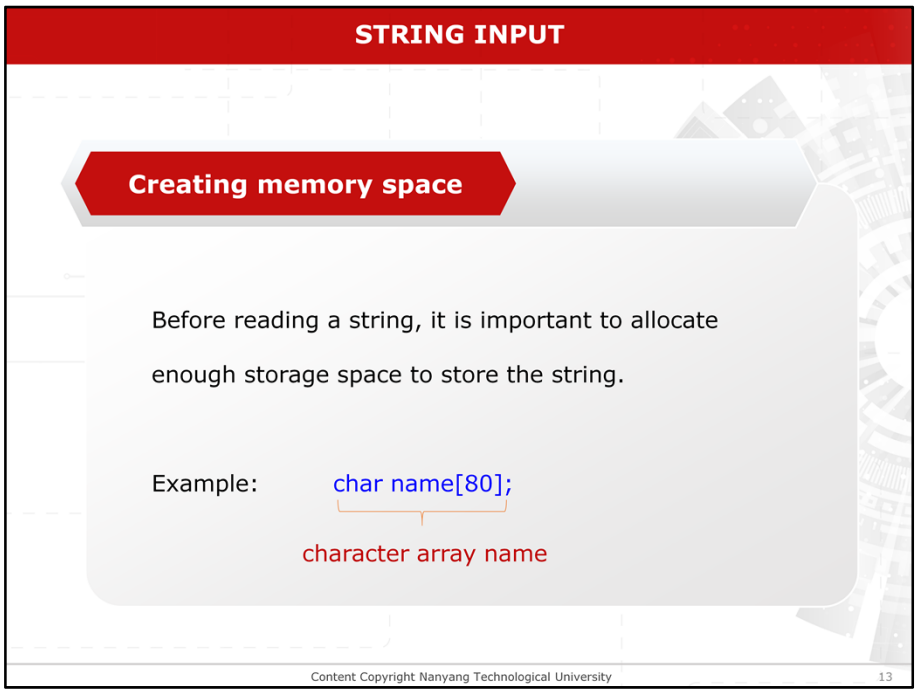


Before reading a string, it is important to allocate enough storage space to store the string.

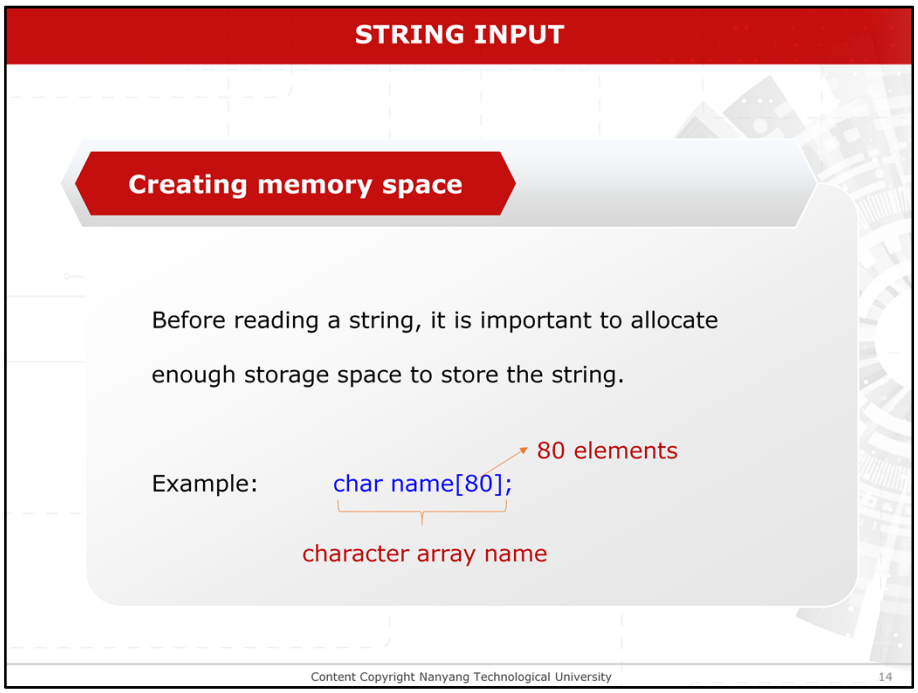


To create space for a string, we may include an explicit array size as shown in the following declaration:

character name [80];



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



of 80 elements.

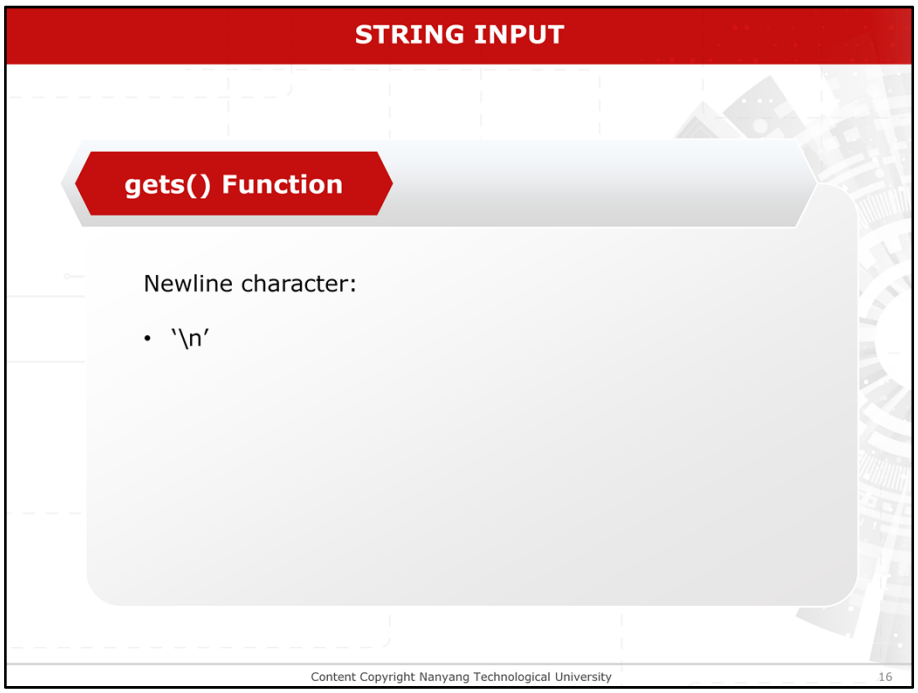
Once the storage space has been acquired, we can read in the string through the C library functions.

STRING INPUT

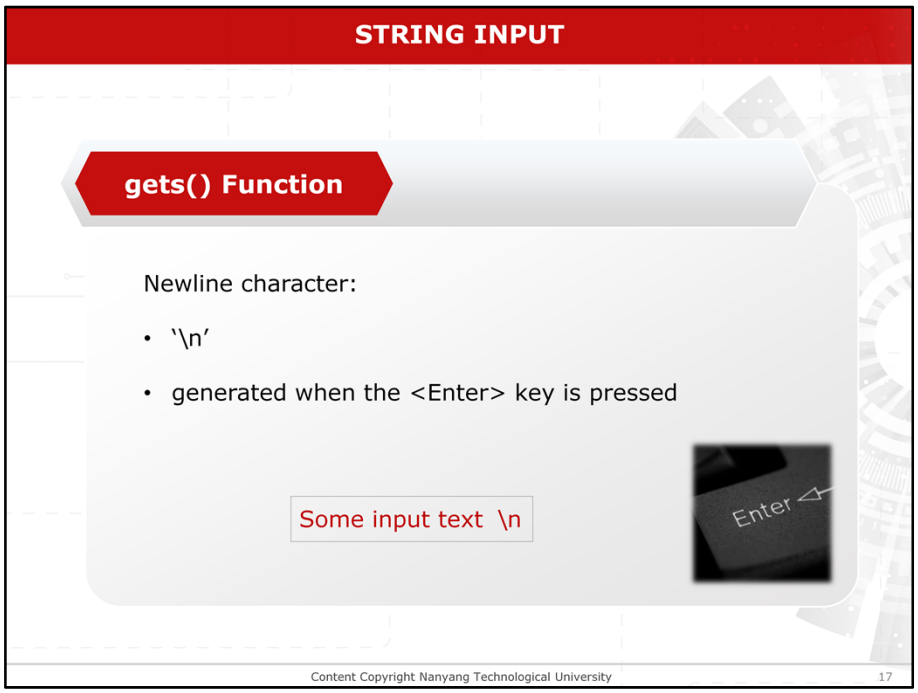
gets() Function

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

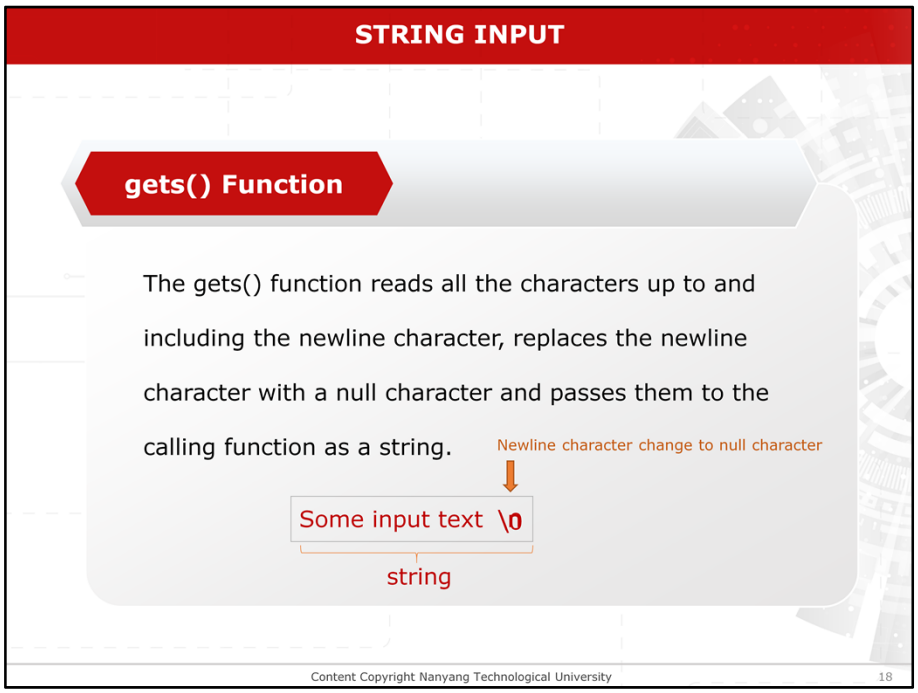
The get s() function gets a string from the standard input device.



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



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

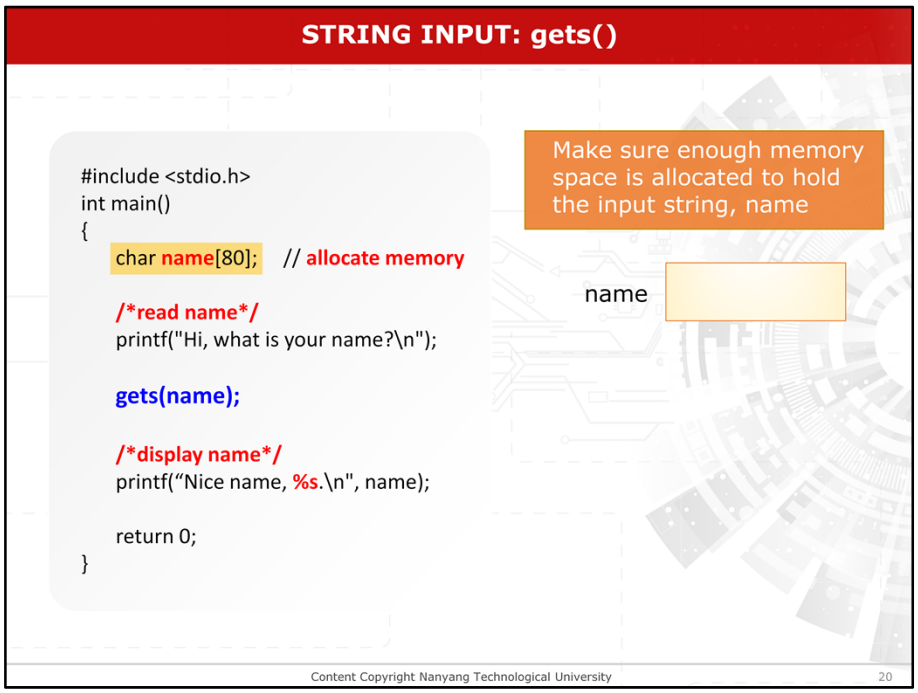


The **get s()** function reads all the characters up to and including the newline character, replaces the newline character with a null character and passes them to the calling function as a string.

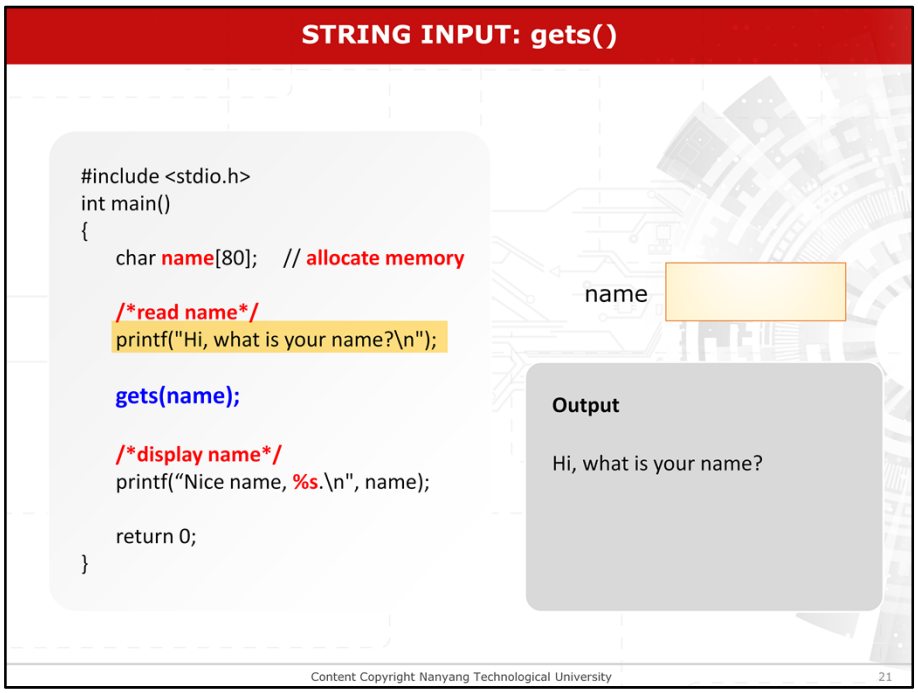
STRING INPUT: gets()

```
#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;
}
```

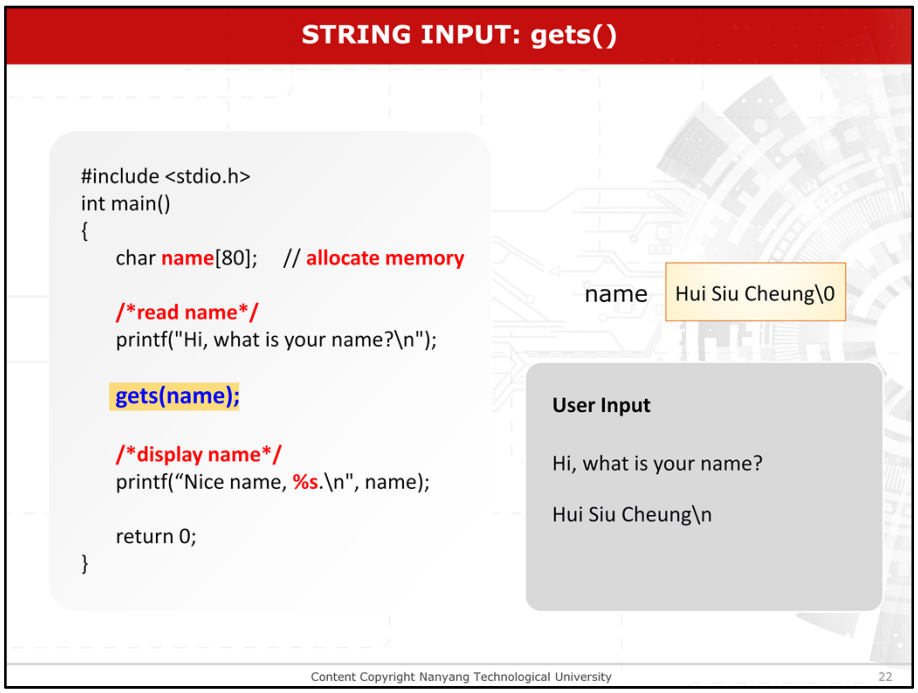

In this program, it will read a string and prints the string to the screen.



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.



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



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;
}
```

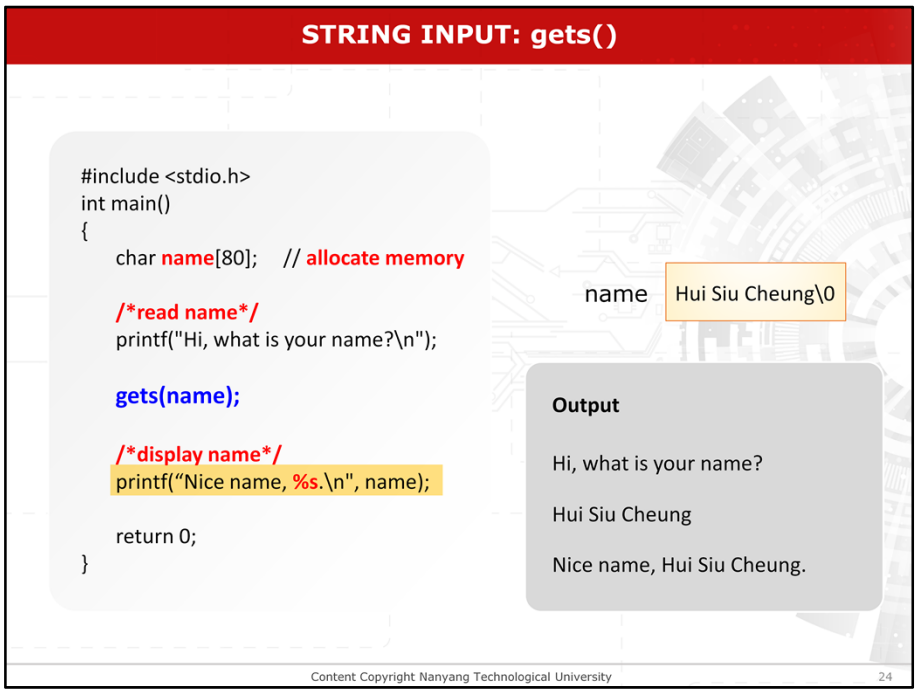
gets() returns Null if it fails, otherwise a pointer to the string is returned.

name | Hui Siu Cheung\0

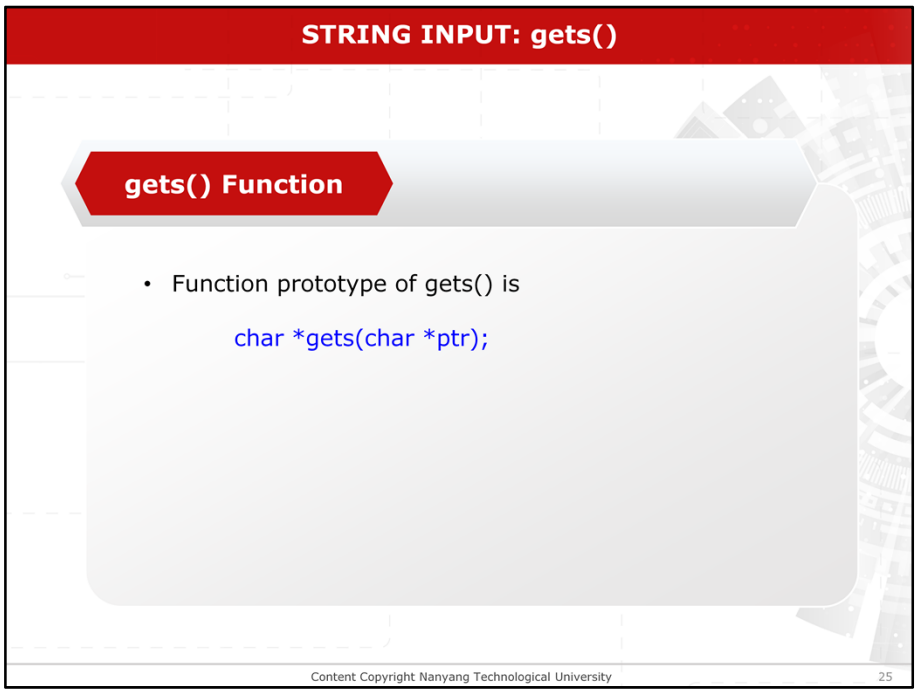
User Input

Hi, what is your name?

Hui Siu Cheung

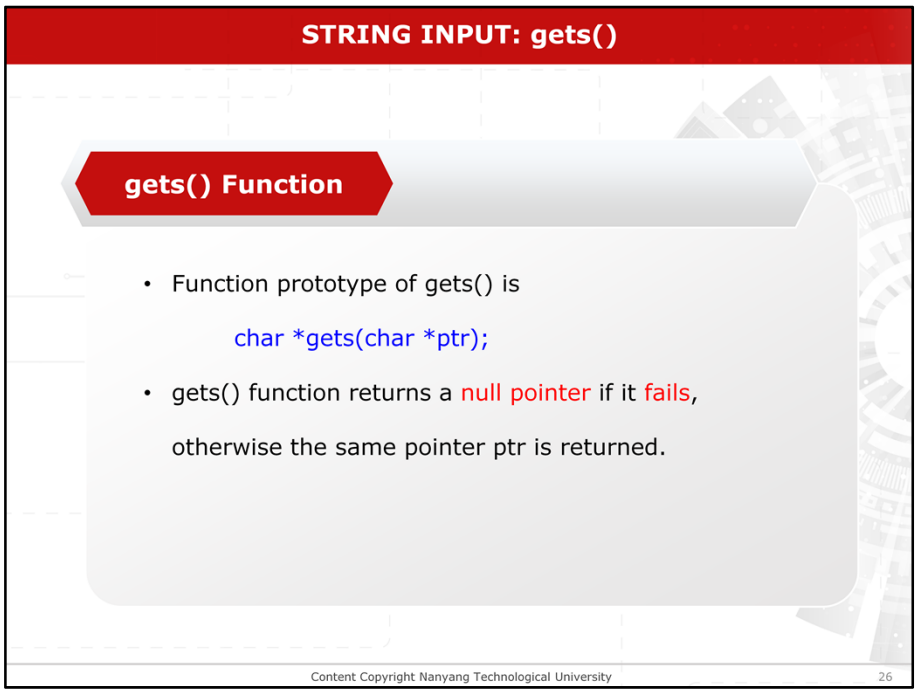


The string is then printed out.

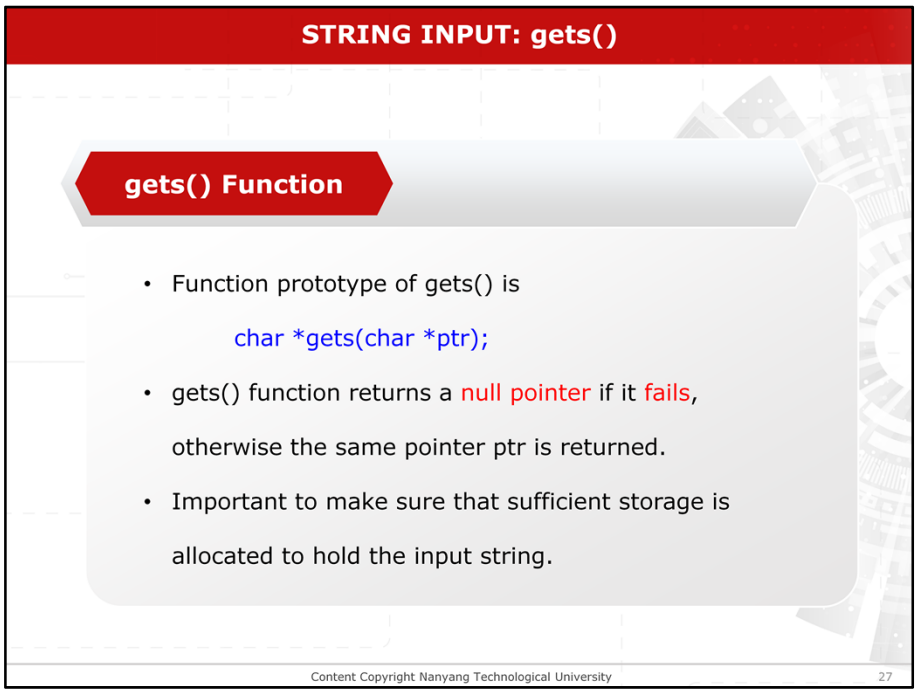


The function prototype of **get s()** is

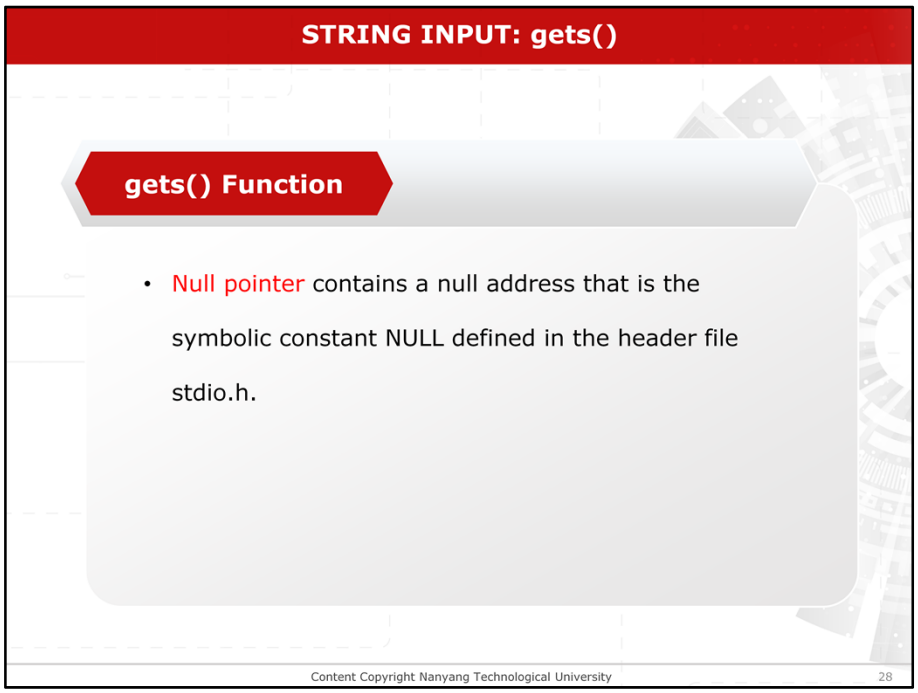
character asterisk get s (character asterisk pointer);



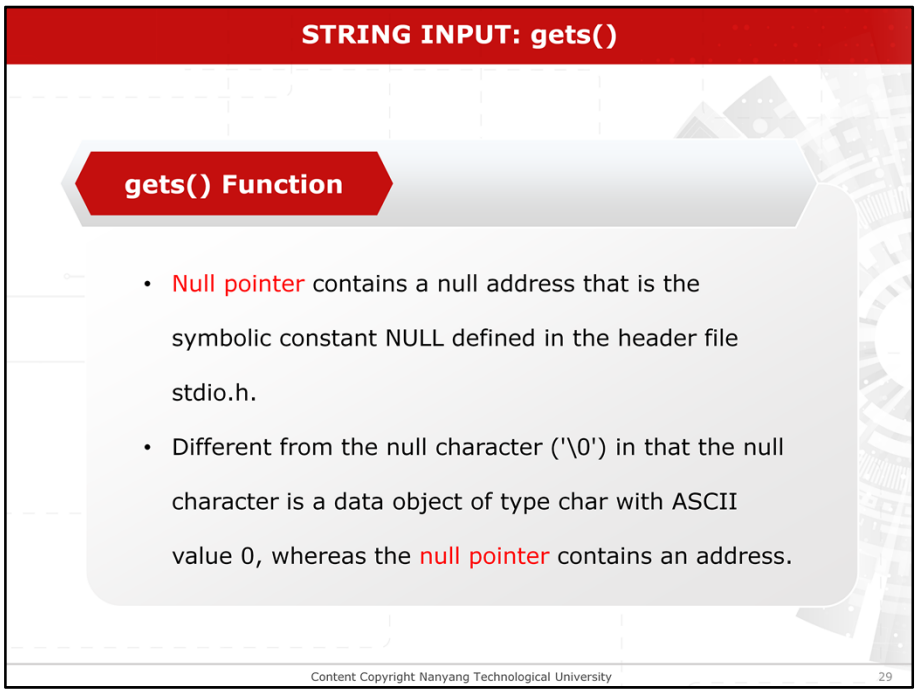
The **get s()** function returns a null pointer if it fails, otherwise the same pointer is returned.



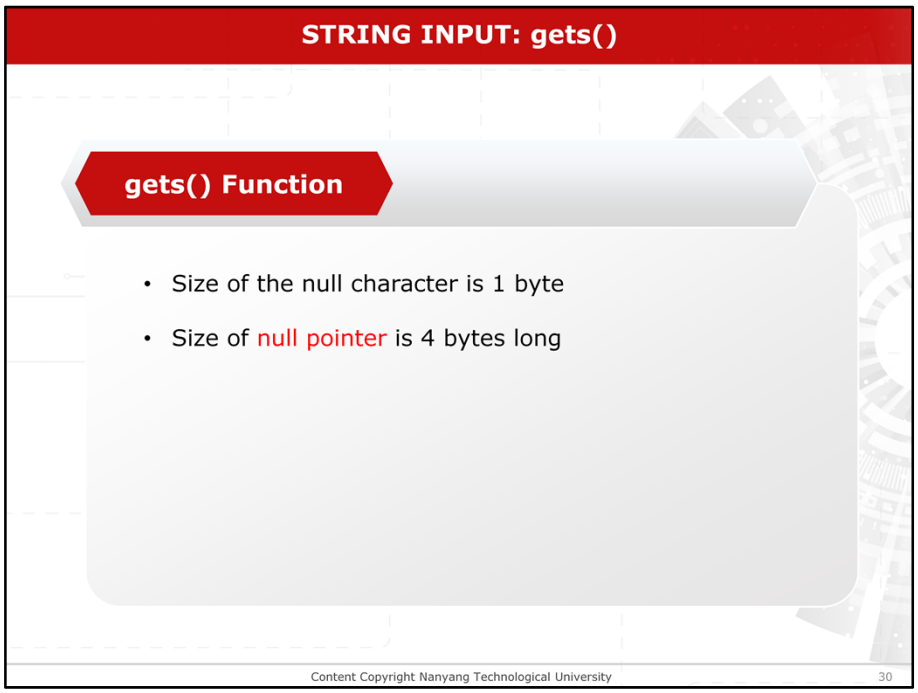
It is important to make sure that sufficient storage is allocated to hold the input string.



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



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.



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;
}
```


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

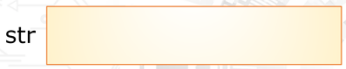
STRING OUTPUT: puts()

```
#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;
}
```

Array declared can hold up to 80 elements

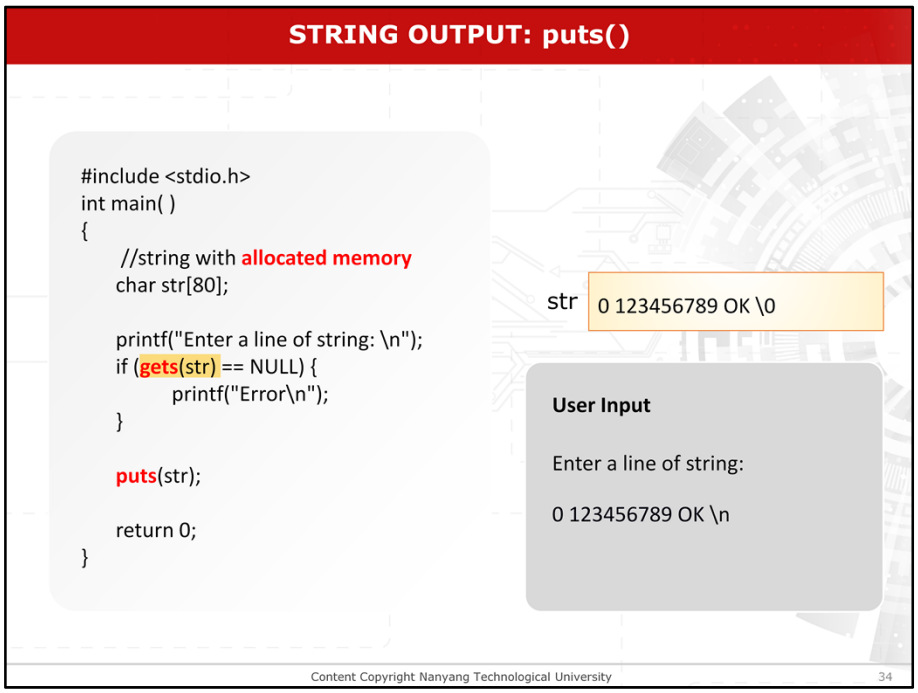
STRING OUTPUT: puts()

```
#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;
}
```



User Input

Enter a line of string:



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

STRING OUTPUT: puts()

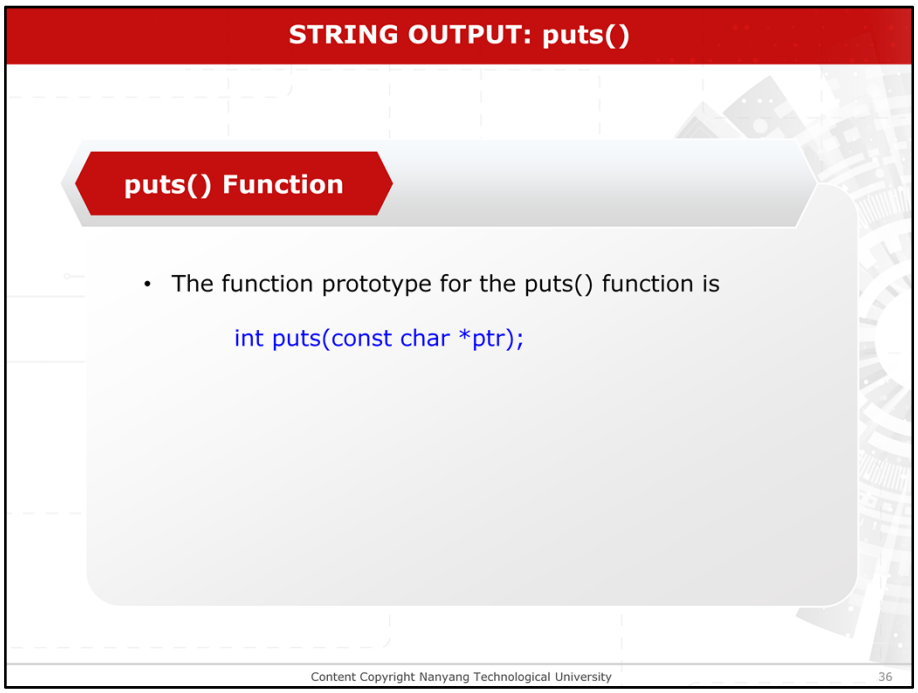
```
#include <stdio.h>
int main( )
{
      //string with allocated memory
     char str[80];

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

     puts(str);

     return 0;
}
```

str | 0 123456789 OK \0

Output

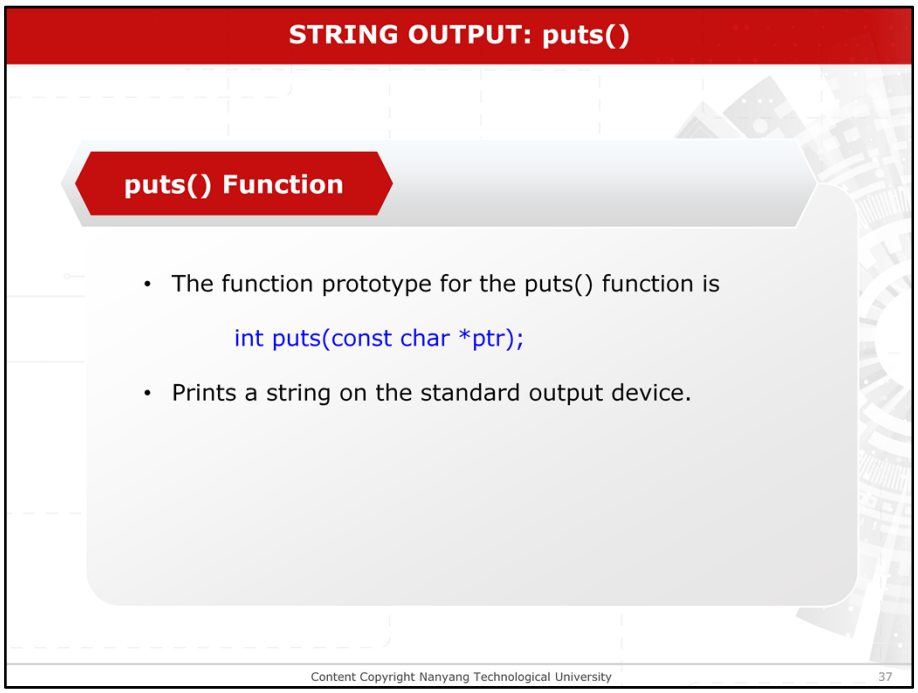
```
Enter a line of string:
0 123456789 OK
0 123456789 OK
```


prints a string on the standard output device.

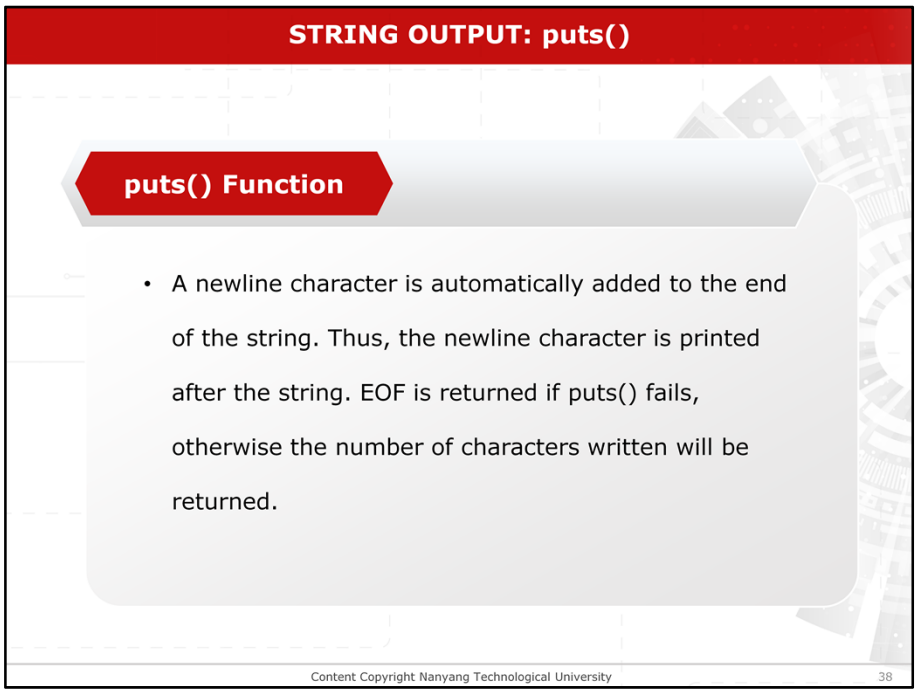


The function prototype for the puts() function is

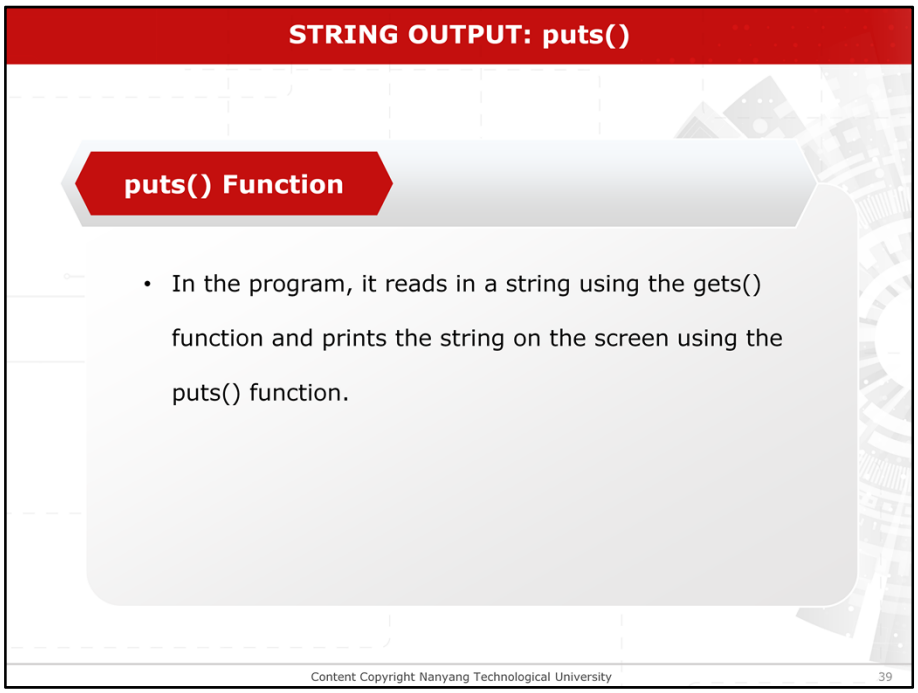
integer put s (constant character asterisk pointer);



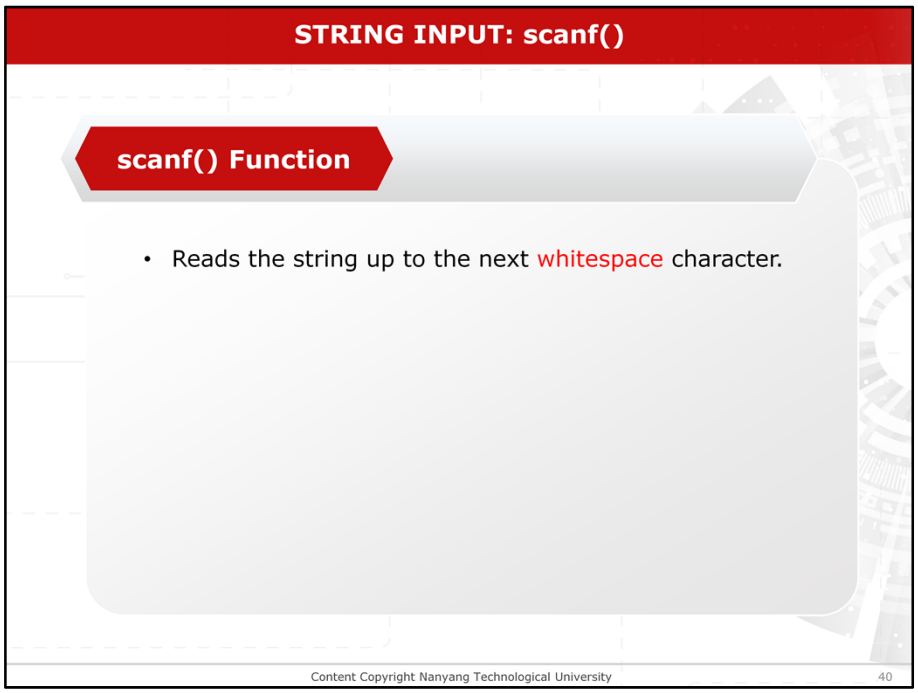
It prints a string on the standard output device.



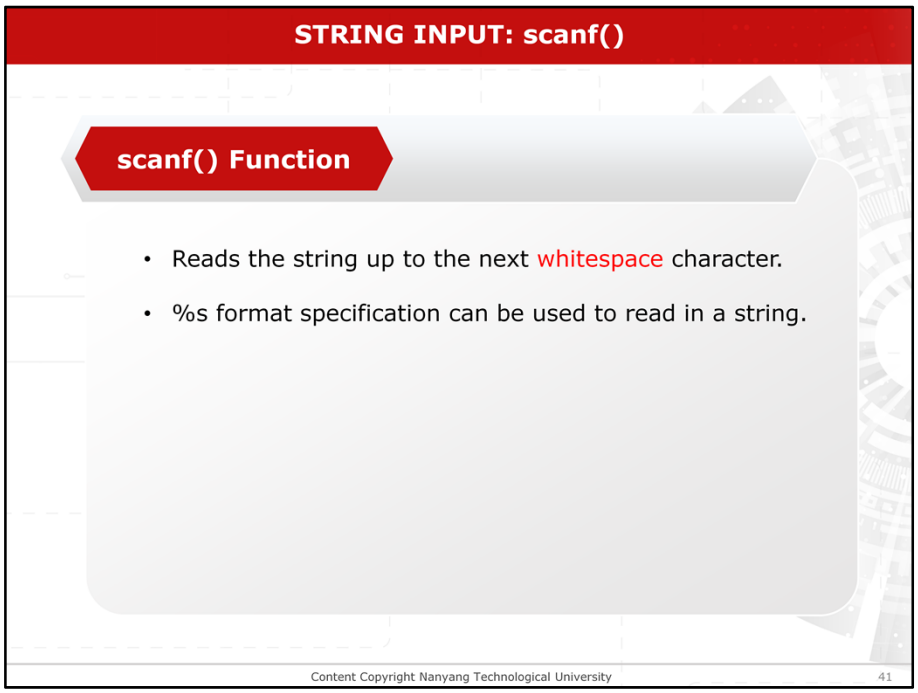
A newline character is automatically added to the end of the string. Thus, the *newline* character is printed after the string. **E O F** is returned if **put s()** fails, otherwise the number of characters written will be returned.



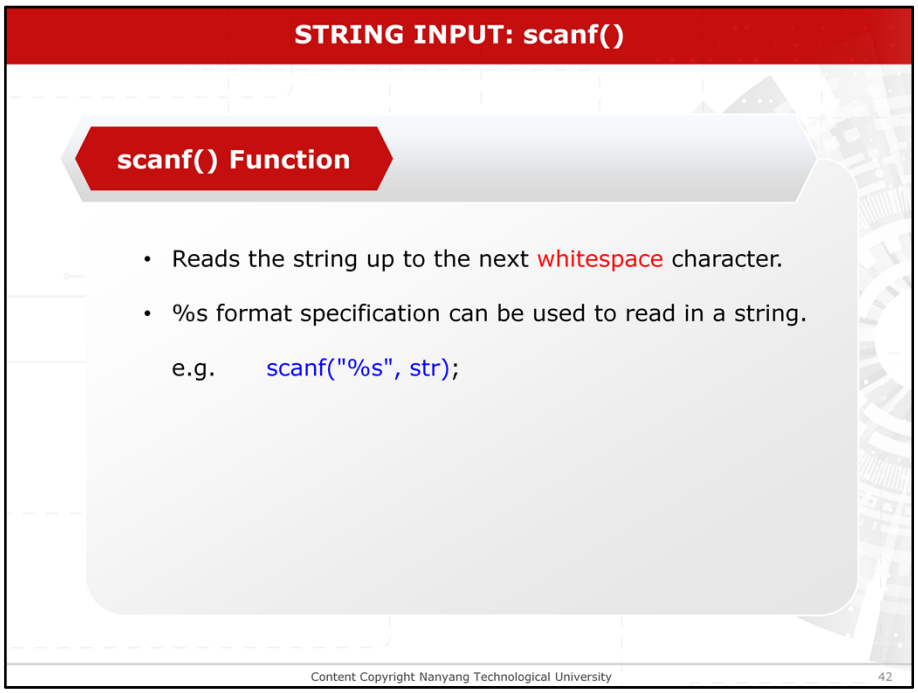
In the program, it reads in a string using the **get s()** function and prints the string on the screen using the **put s()** function.



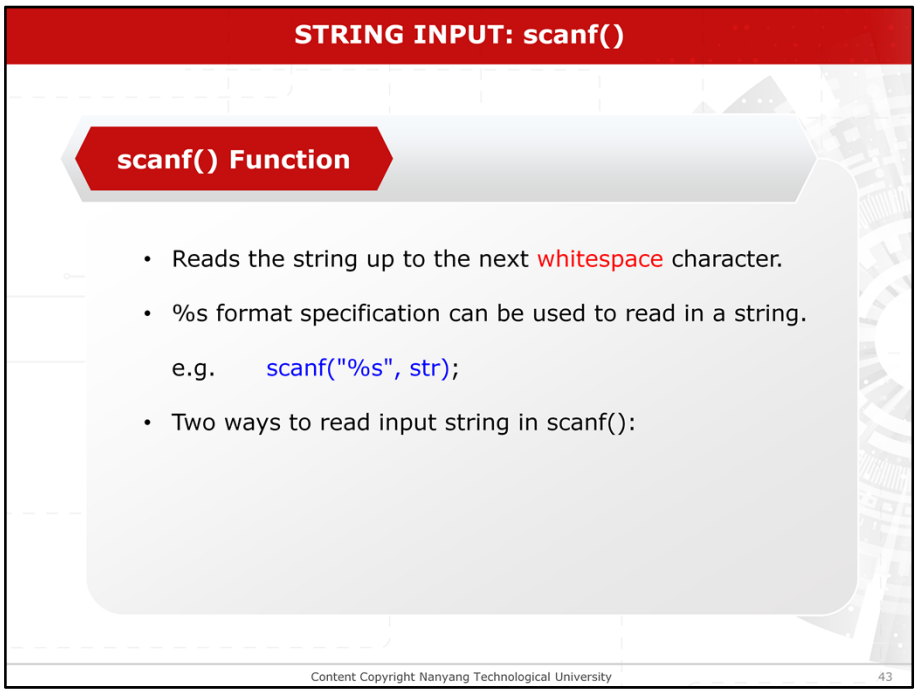
The scan f() function reads the string up to the next whitespace character.



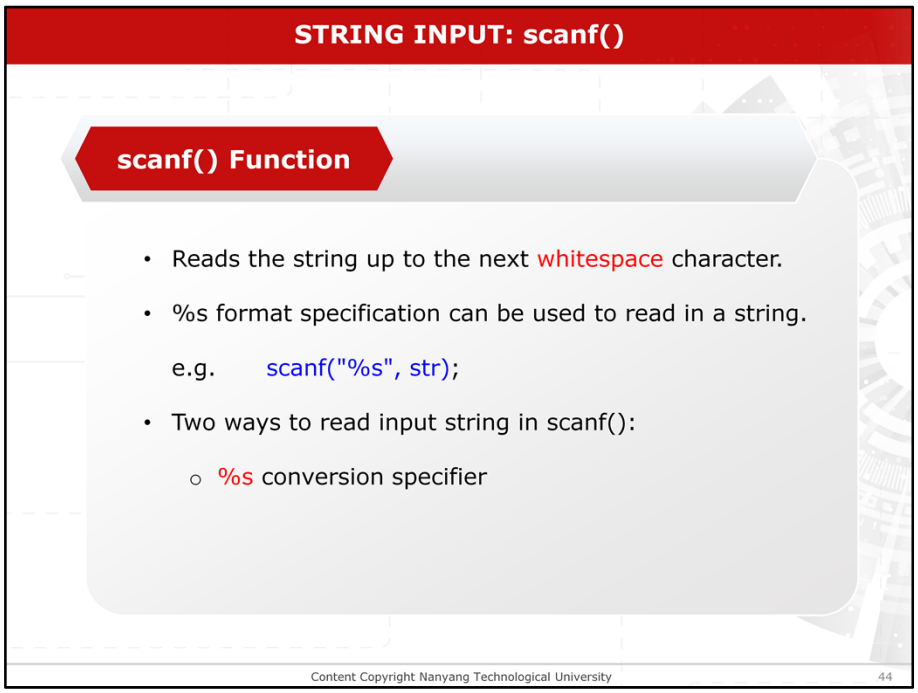
The **scan f()** function with the **percentage s** format specification can be used to read in a string.



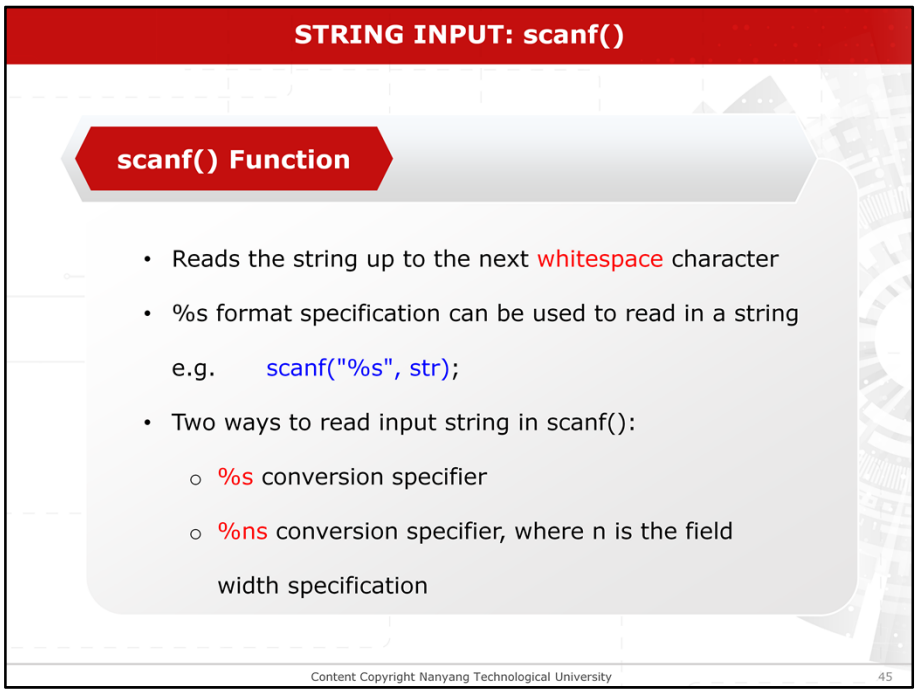
An example of the **scan f()** function to read in a string is given as follows:
scan f (double quote percentage s double quote, string);



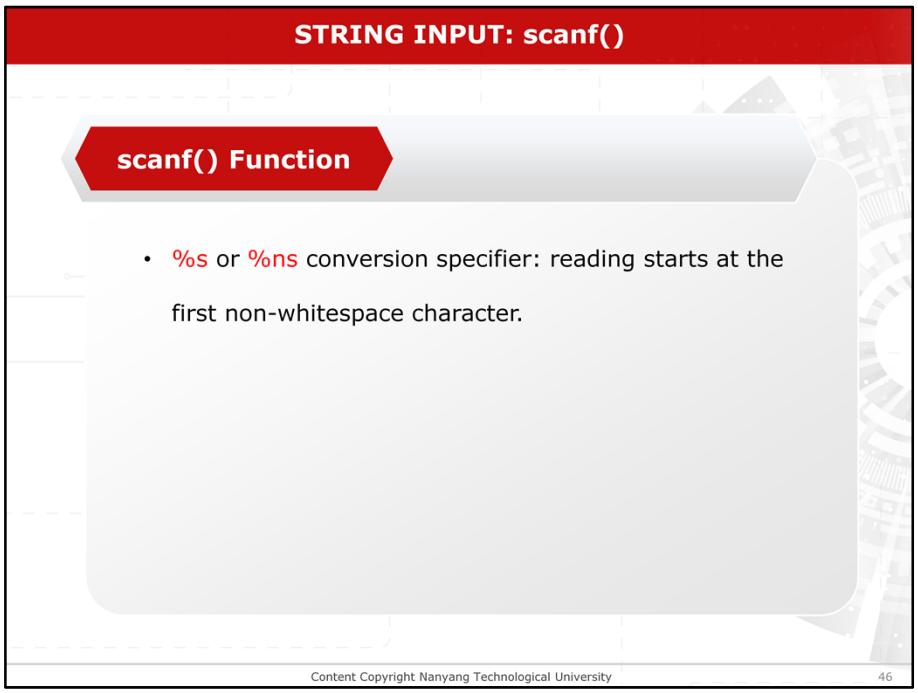
There are two ways to read input string in **scan f()**.



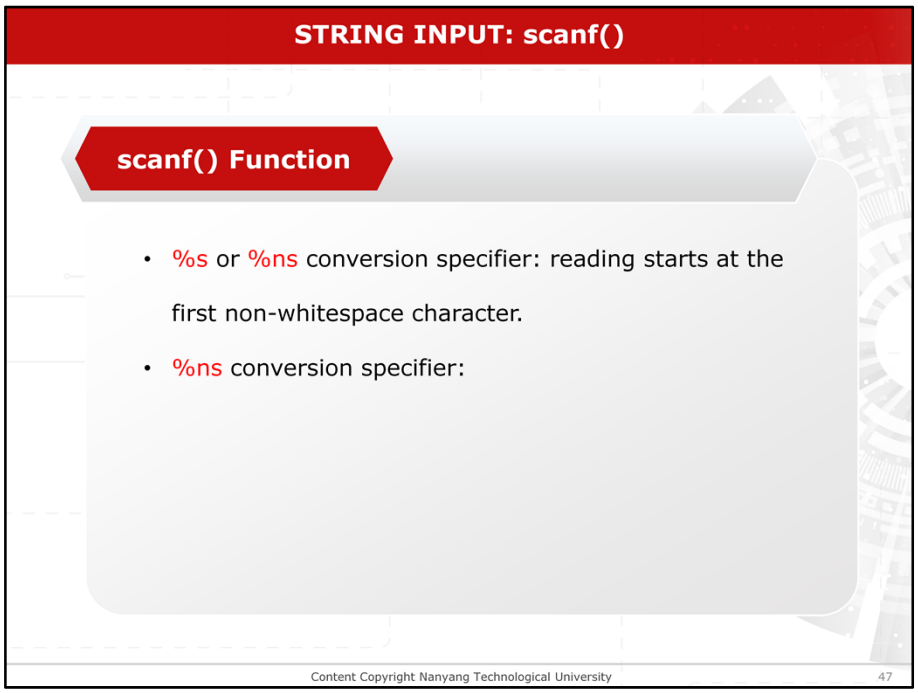
We may use either a **percentage s** conversion specifier



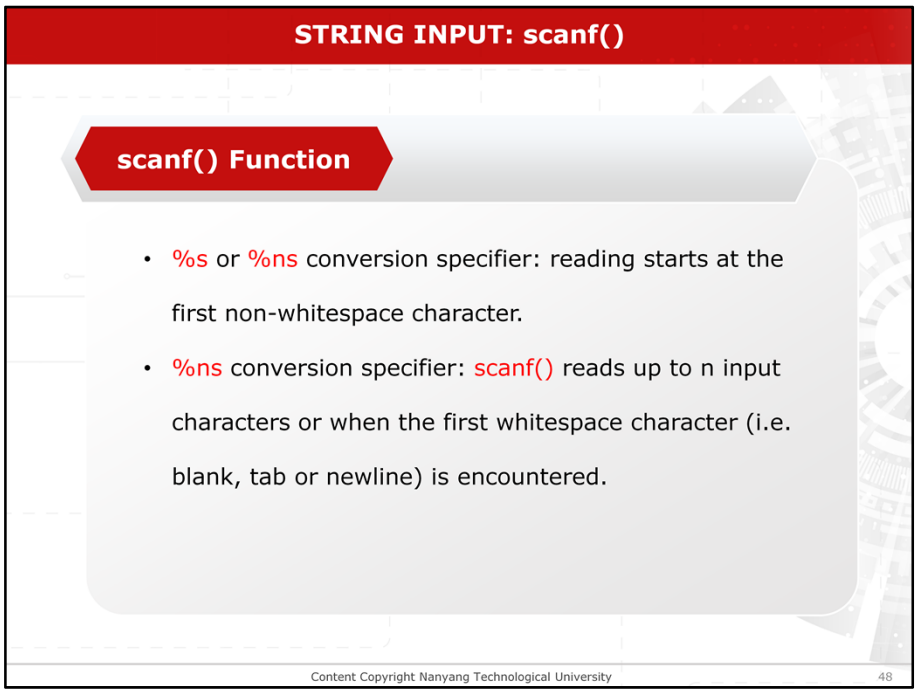
or **percentage n s** conversion specifier, where **n** is the field width specification.



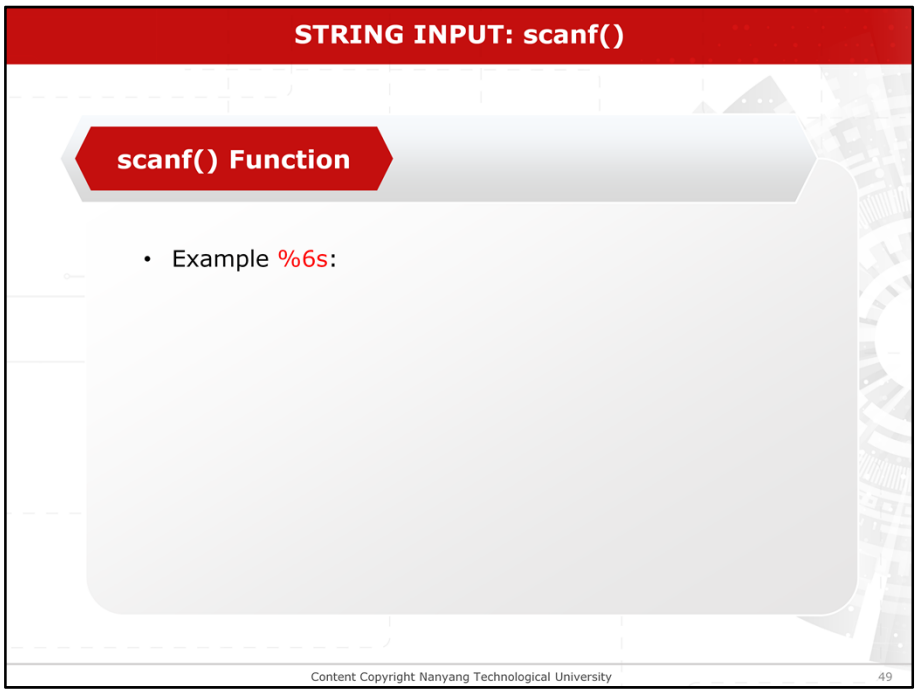
In both cases, reading starts at the first non-whitespace character.



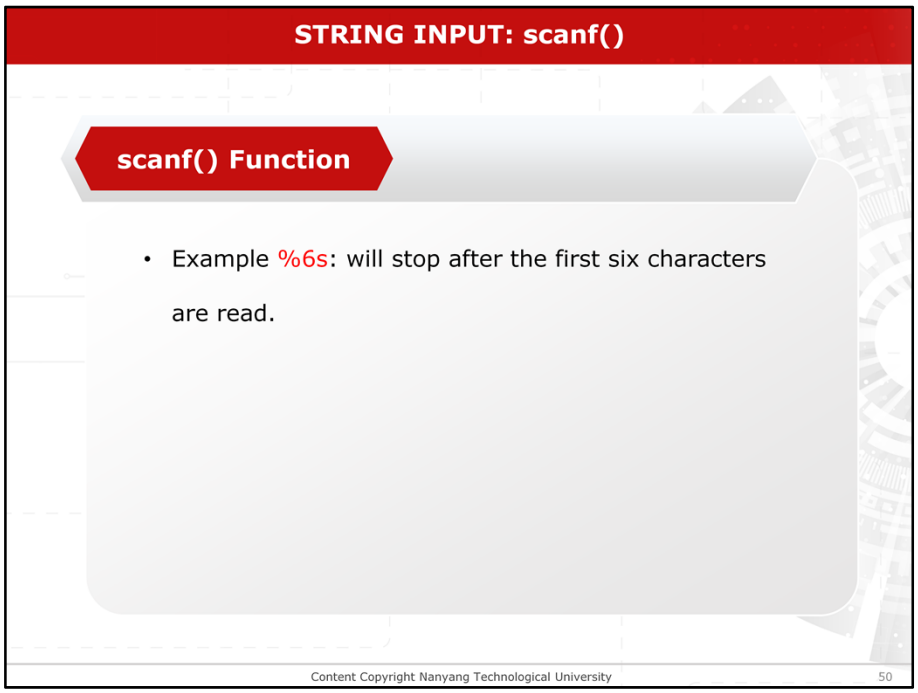
When **percentage n s** is used,



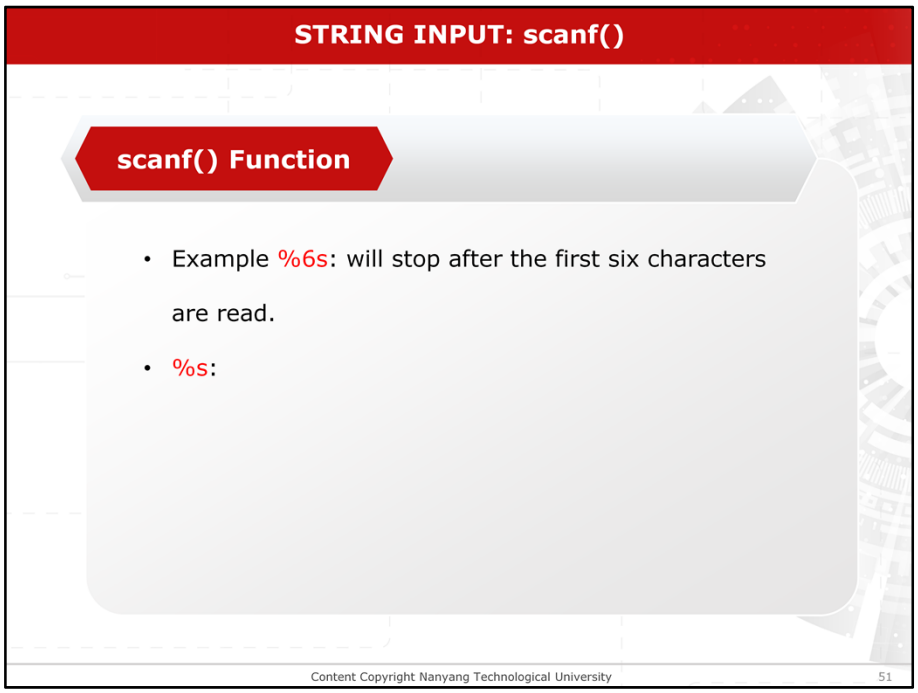
Scan f() reads up to **n** input characters or when the first whitespace character that is when blank, tab or new line is encountered.



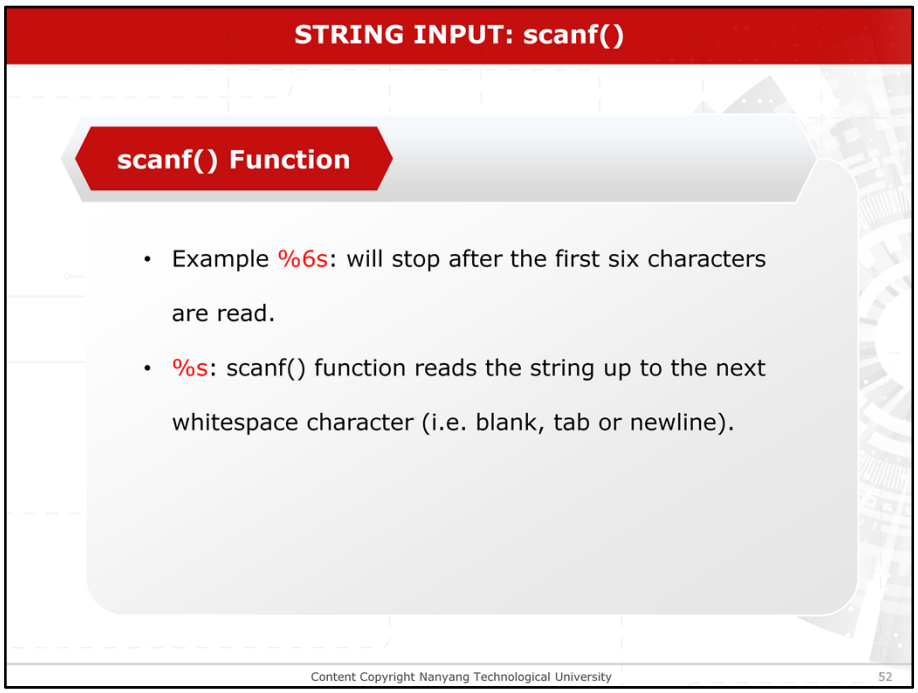
For example, **percentage 6 s**



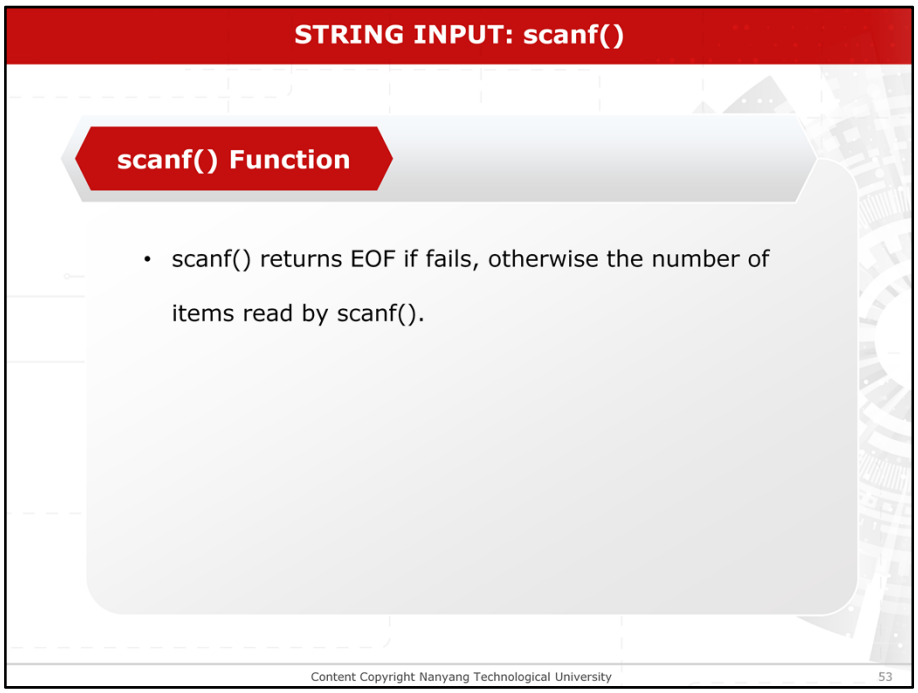
will stop after the first six characters are read



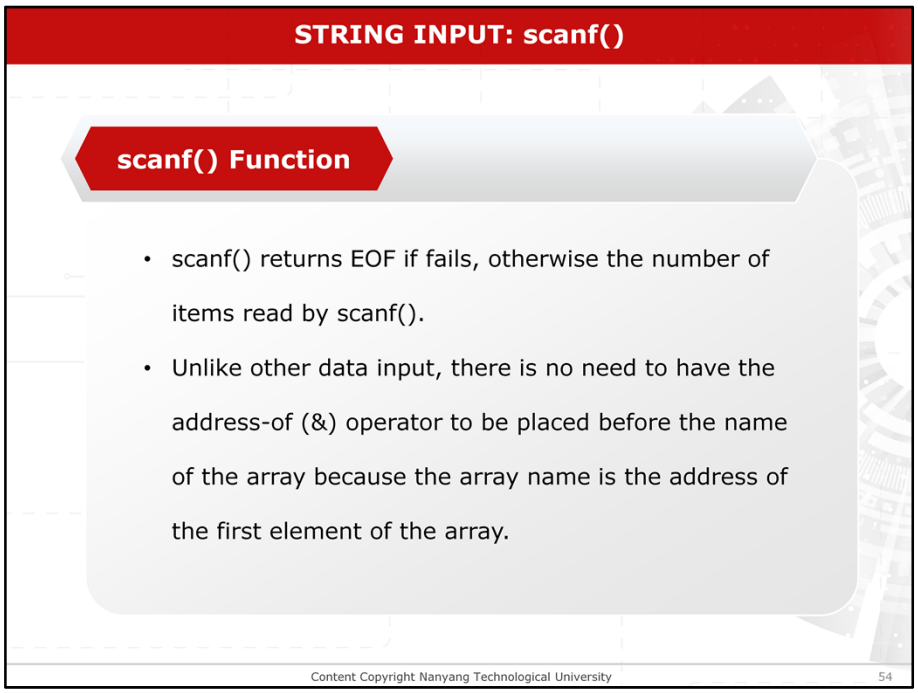
When **percentage s** is specified,



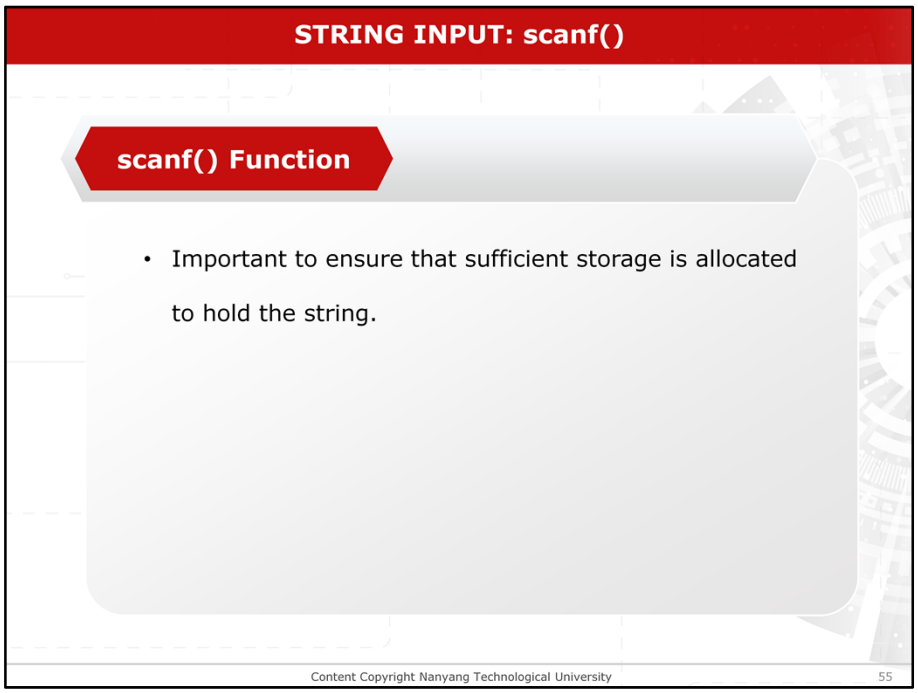
the **scan f()** function reads the string up to the next whitespace character.



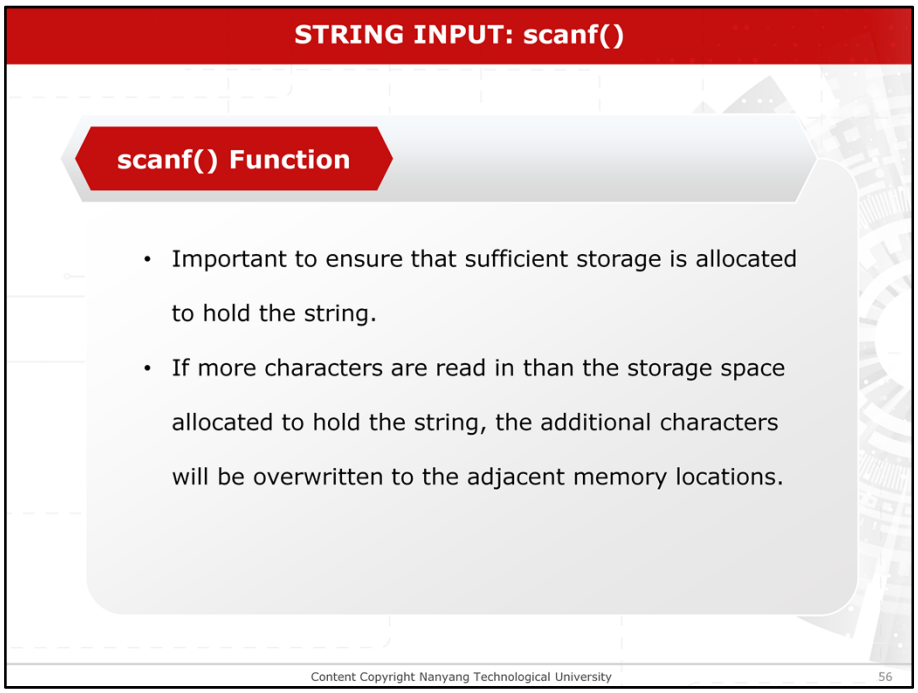
This function will return **E O F** if it fails, otherwise the number of items read by **scan f()** will be returned.



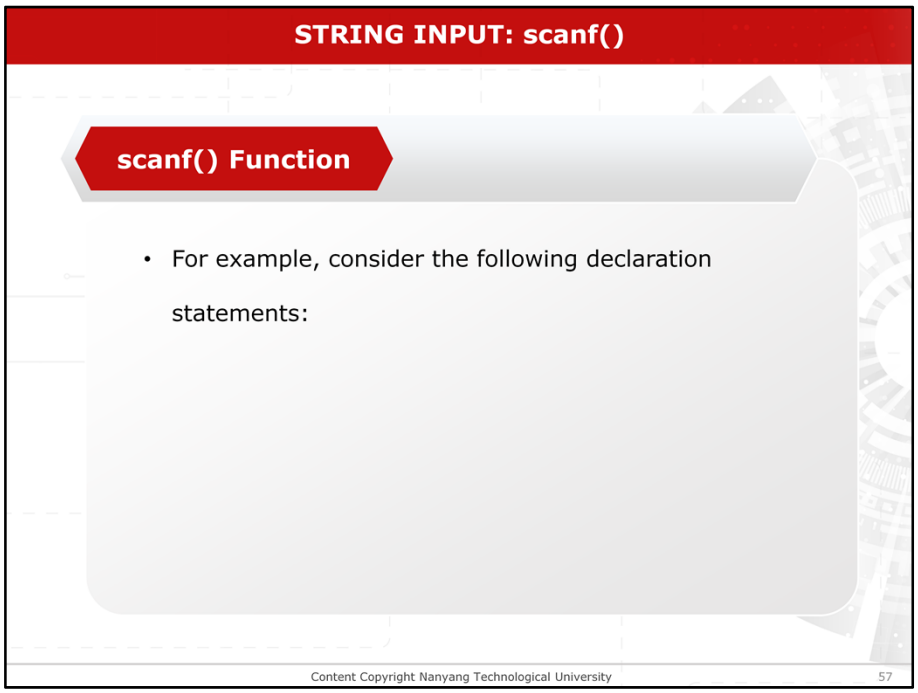
Unlike other data input, there is no need to have the address-of (**ampersand**) operator to be placed before the name of the array because the array name is the address of the first element of the array.



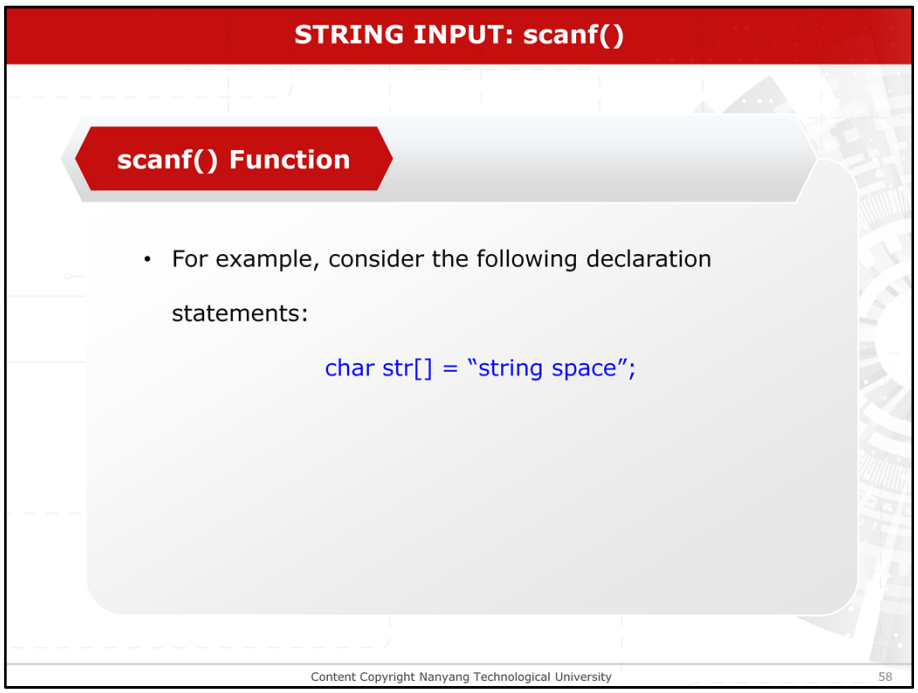
It is important to ensure that **sufficient storage** is allocated to hold the string.



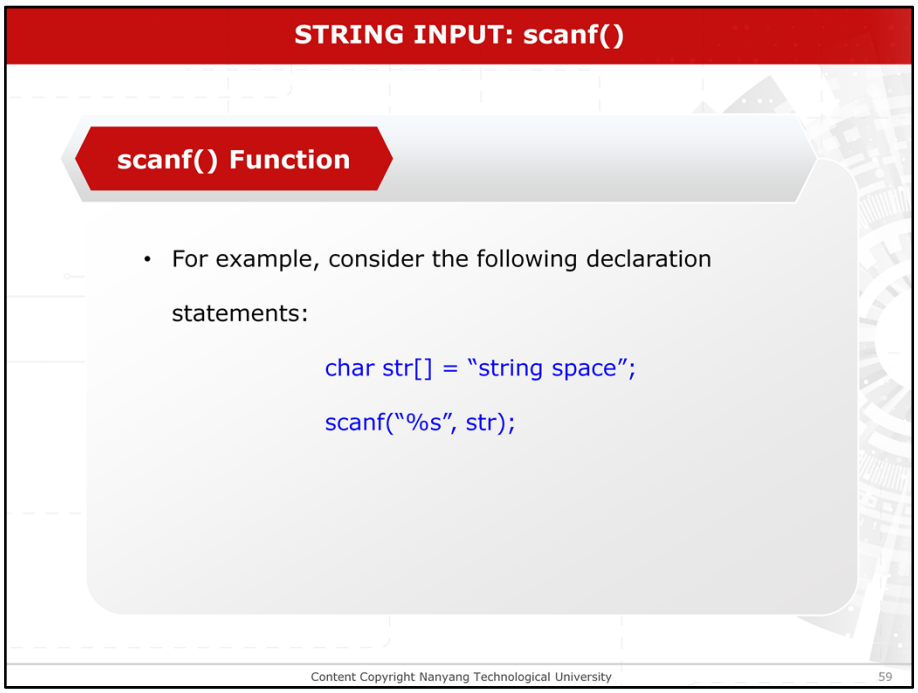
If more characters are read in than the storage space allocated to hold the string, the additional characters will be overwritten to the adjacent memory locations.



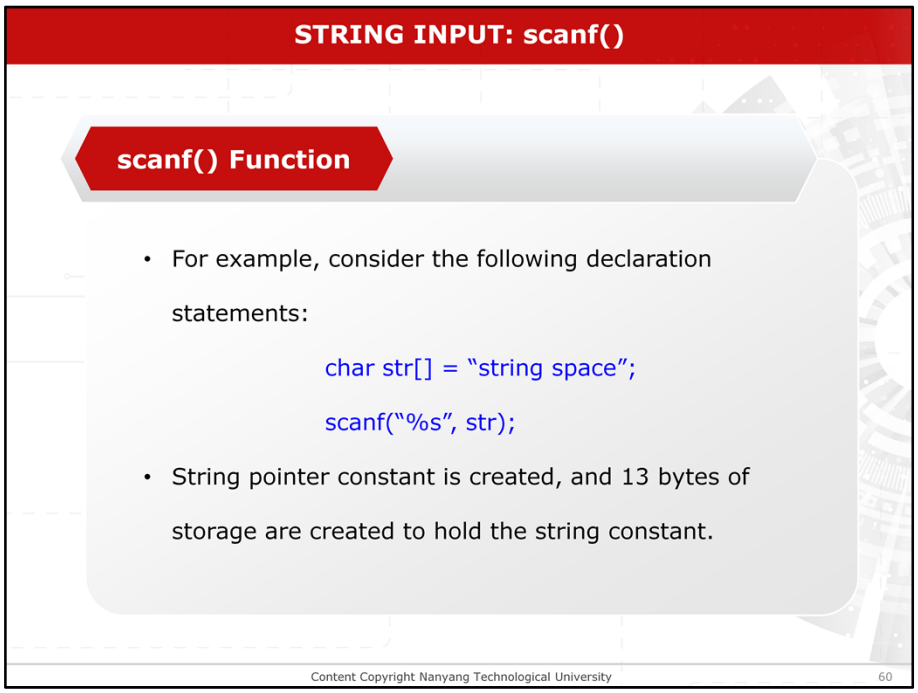
For example, consider the following declaration statements:



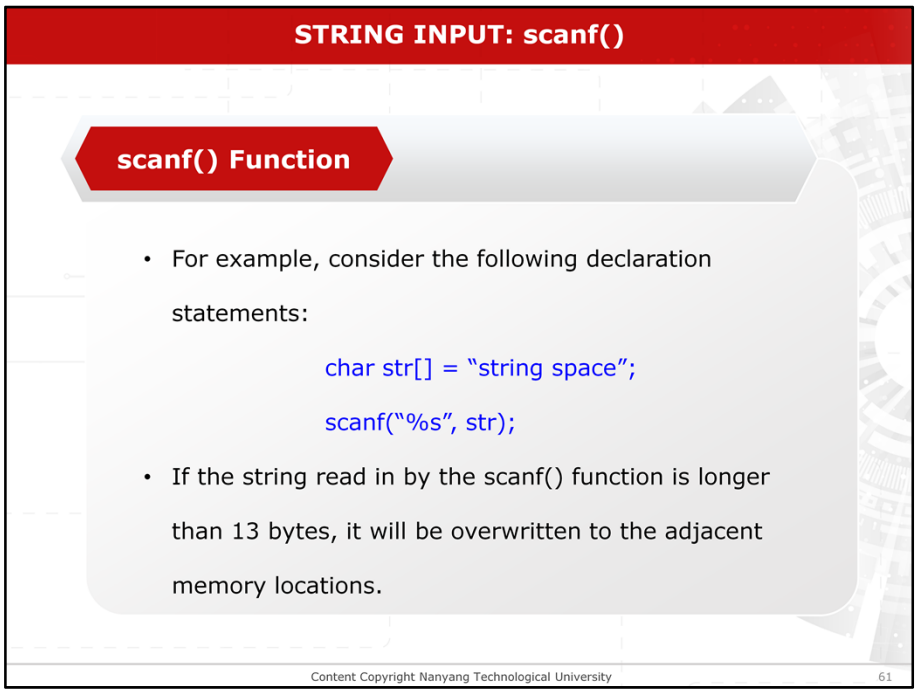
character string = double quote string space double quote;



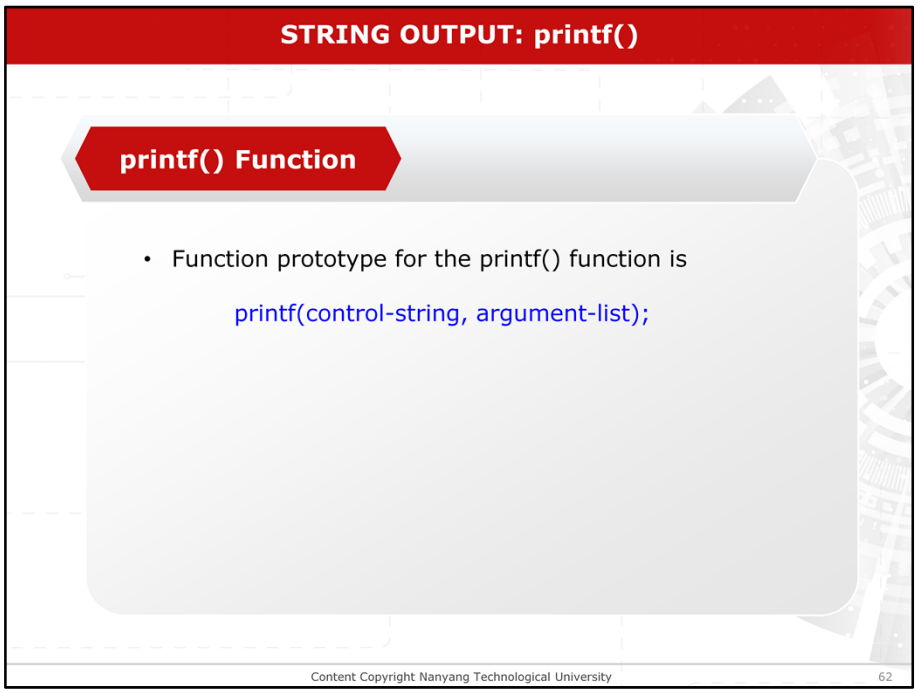
scan f (double quote percentage s double quote, string);



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

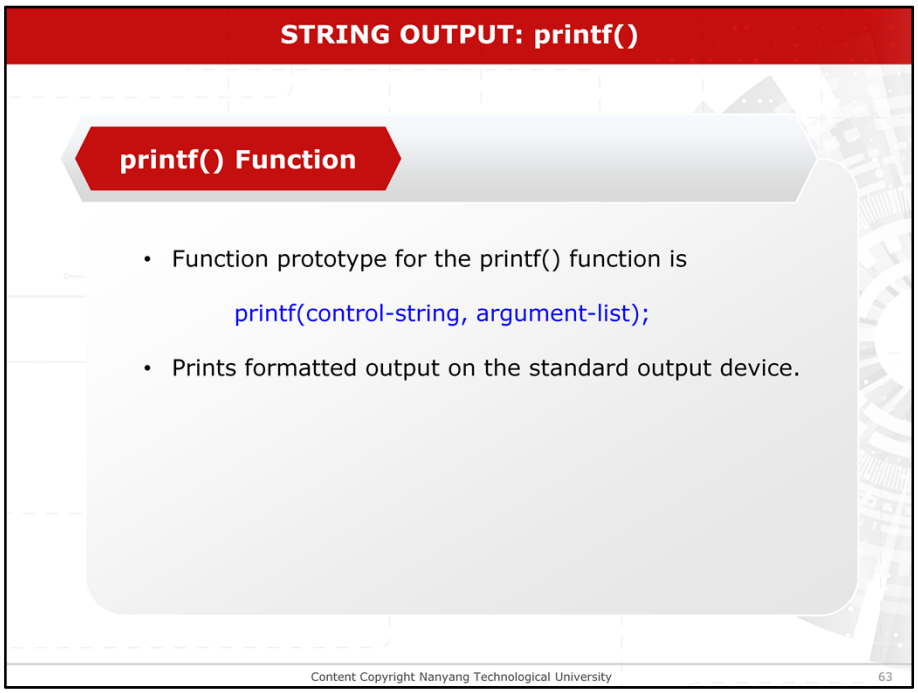


If the string read in by the **scan f()** function is longer than 13 bytes, it will be overwritten to the adjacent memory locations.

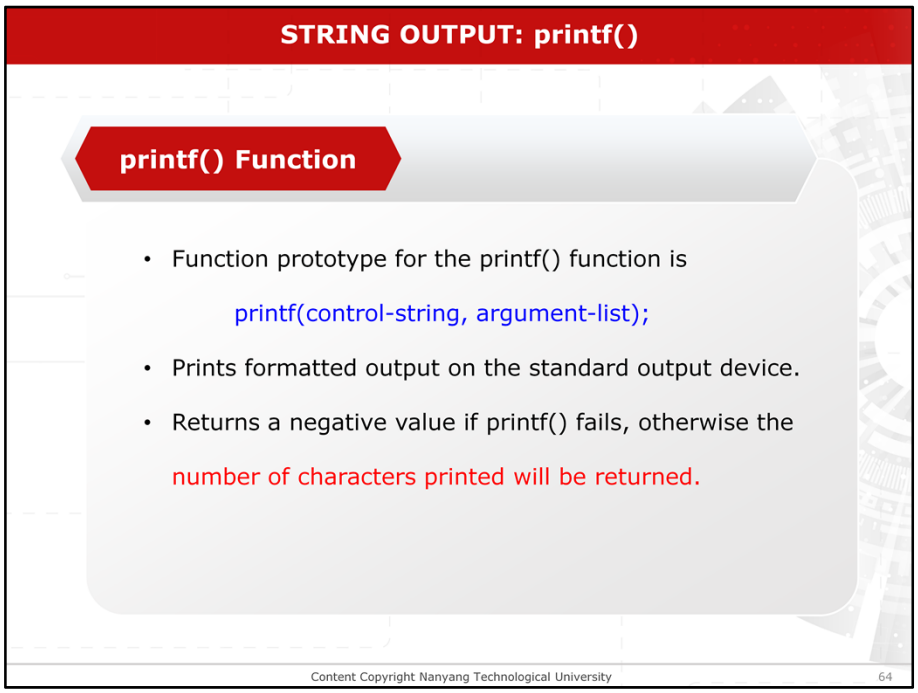


The function prototype for the **print f()** function is

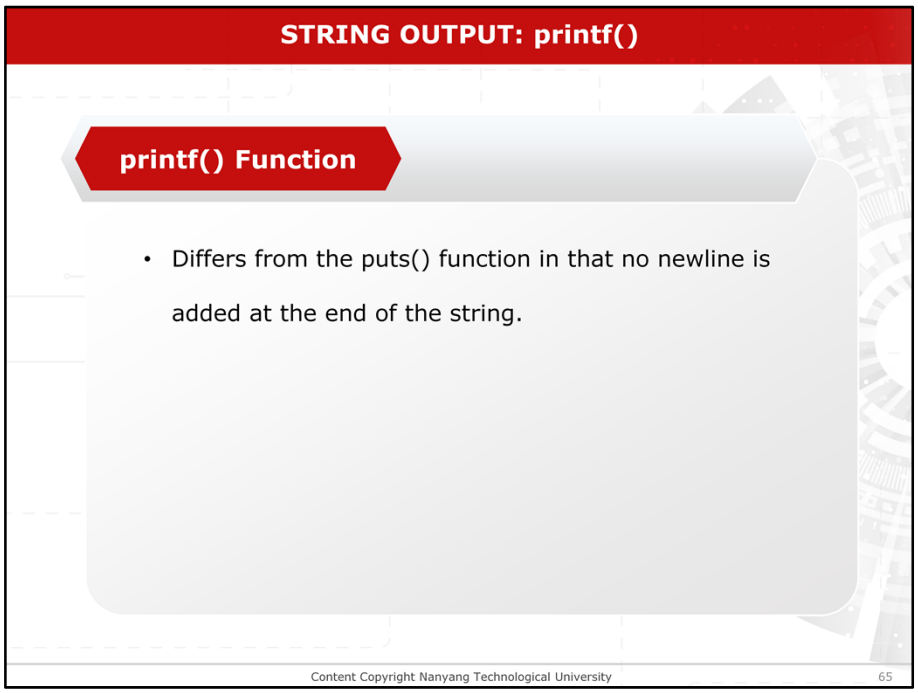
print f (control-string, argument-list);



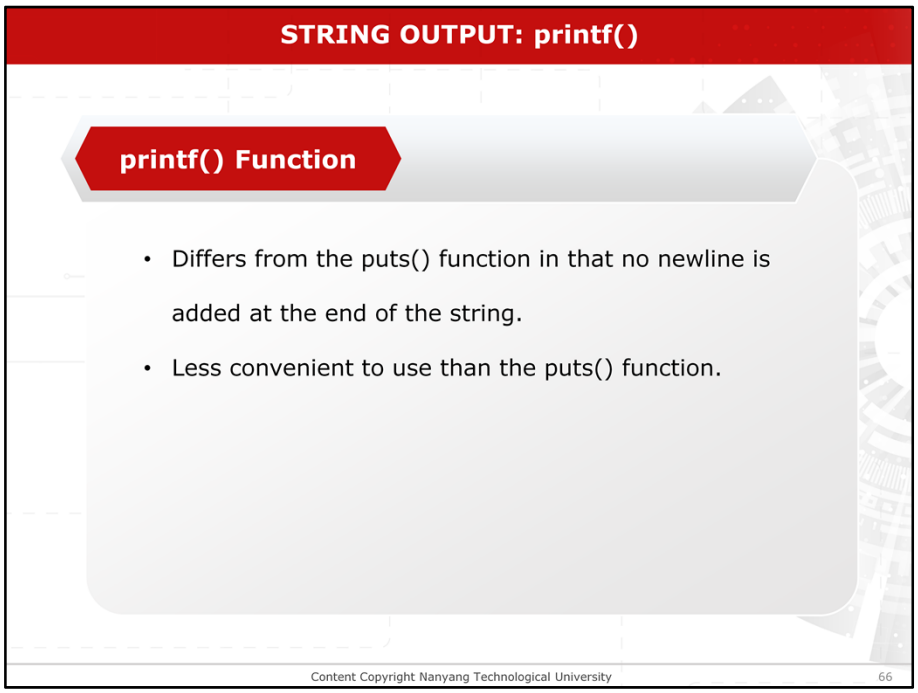
It prints formatted output on the standard output device.



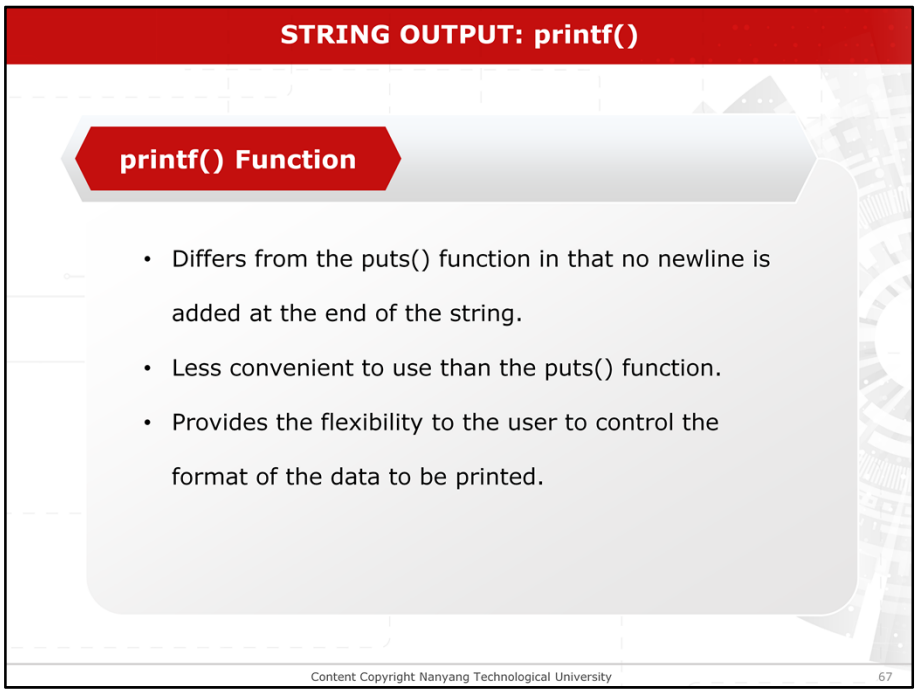
It returns a negative value if print f() fails, otherwise the number of characters printed will be returned.



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



The **print f()** function is less convenient to use than the **put s()** function



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

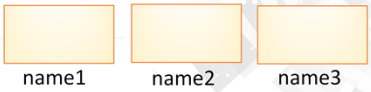
scanf() AND printf(): EXAMPLE

```
#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;
}
```


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;
}
```



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;
}
```

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.

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

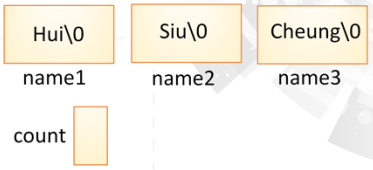
scanf() AND printf(): EXAMPLE

```
#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



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

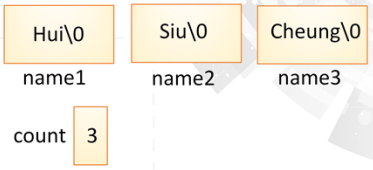
scanf() AND printf(): EXAMPLE

```
#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



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

scanf() AND printf(): EXAMPLE

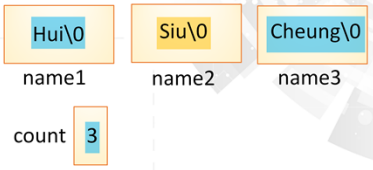
```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    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.



The final print f give the output as shown.

scanf() AND printf(): EXAMPLE

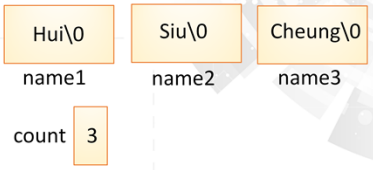
```
#include <stdio.h>
int main( )
{
    char name1[20], name2[20], name3[20];
    int count;

    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

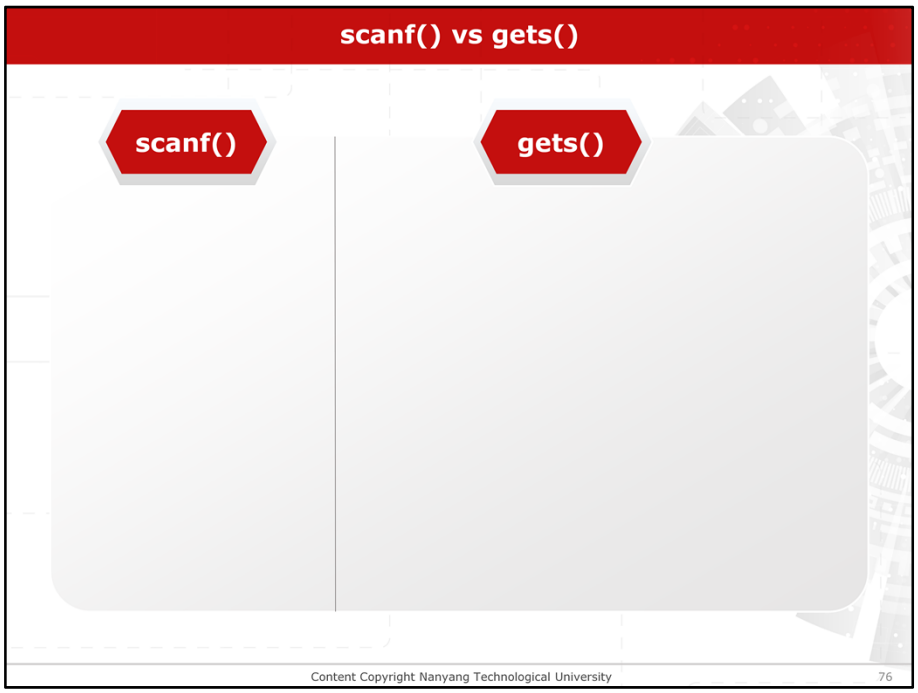
Please enter your strings.

Hui Siu Cheung

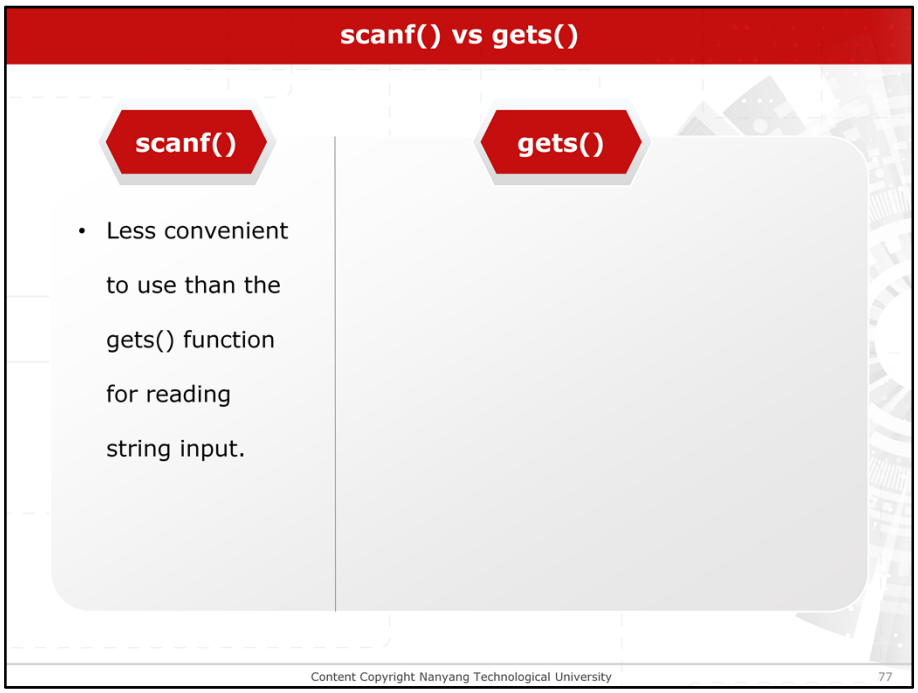
I read the 3 strings: Hui Siu Cheung.



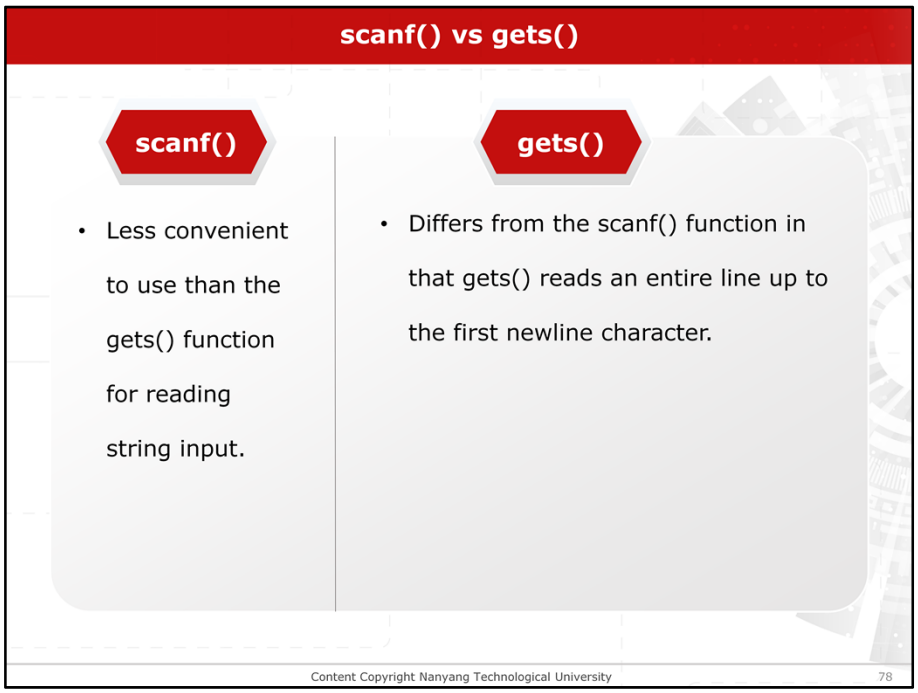
[audio pause]



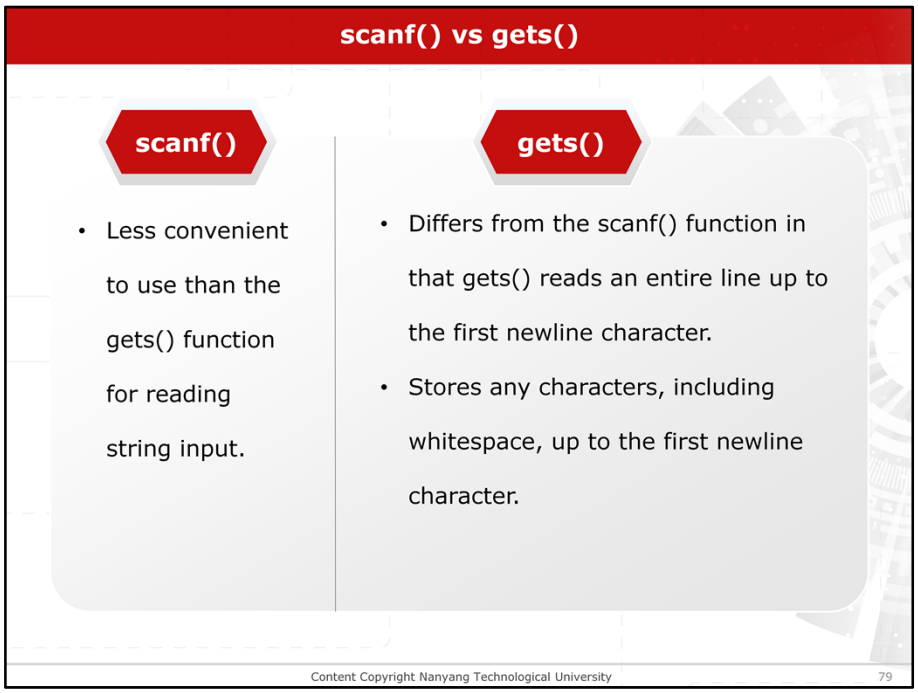
Scan f() vs get s()



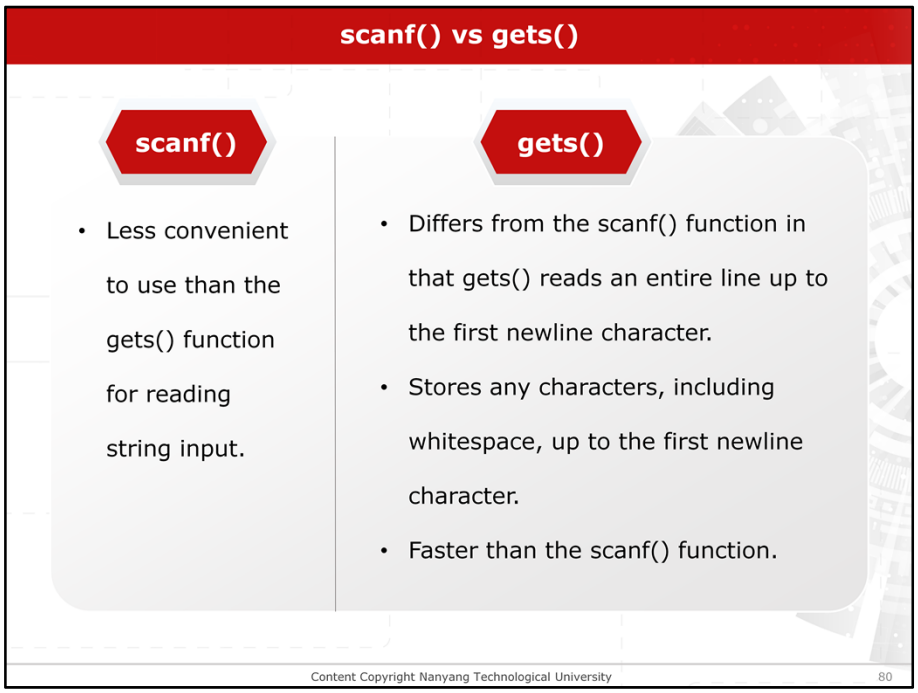
The **scan f()** function is less convenient to use than the **get s()** function for reading string input.



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



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



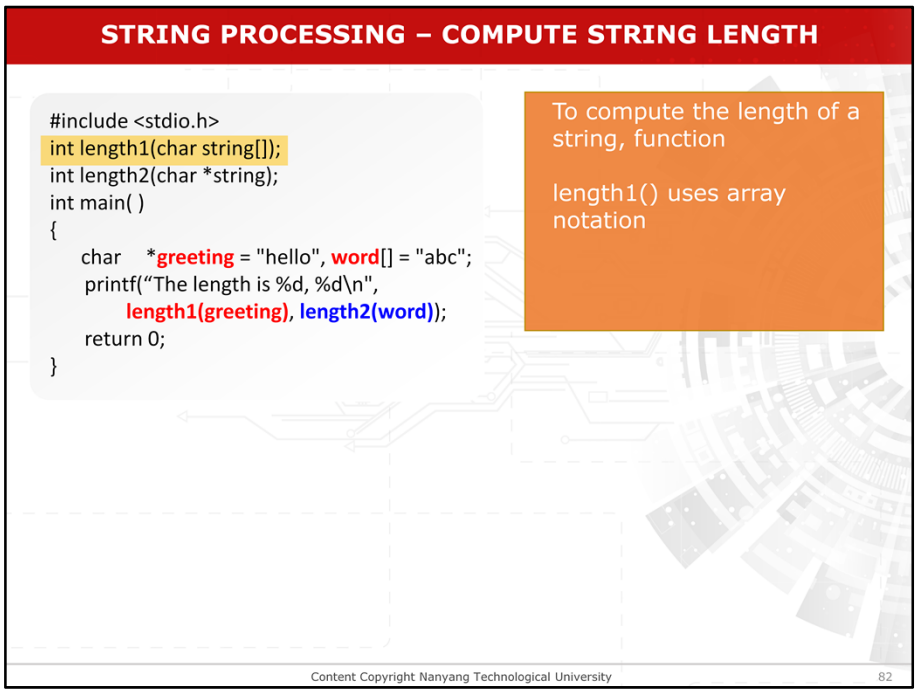
The **get s()** function is also faster than the **scan f()** function.

STRING PROCESSING – COMPUTE STRING LENGTH

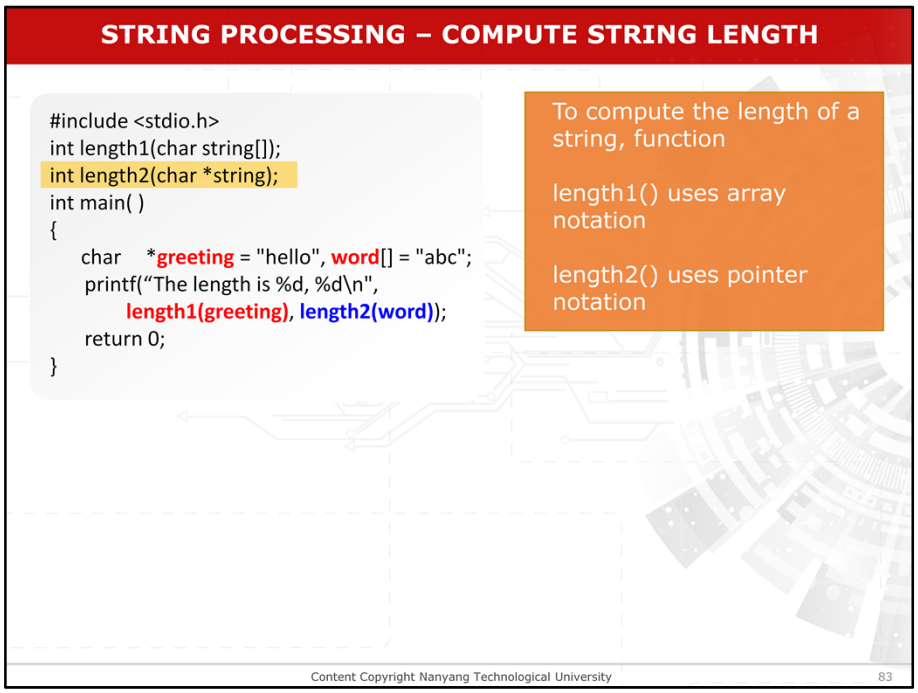
```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char    *greeting = "hello", word[] = "abc";
     printf("The length is %d, %d\n",
          length1(greeting), length2(word));
     return 0;
}
```


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

Two functions are implemented in the program.



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



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

STRING PROCESSING – COMPUTE STRING LENGTH

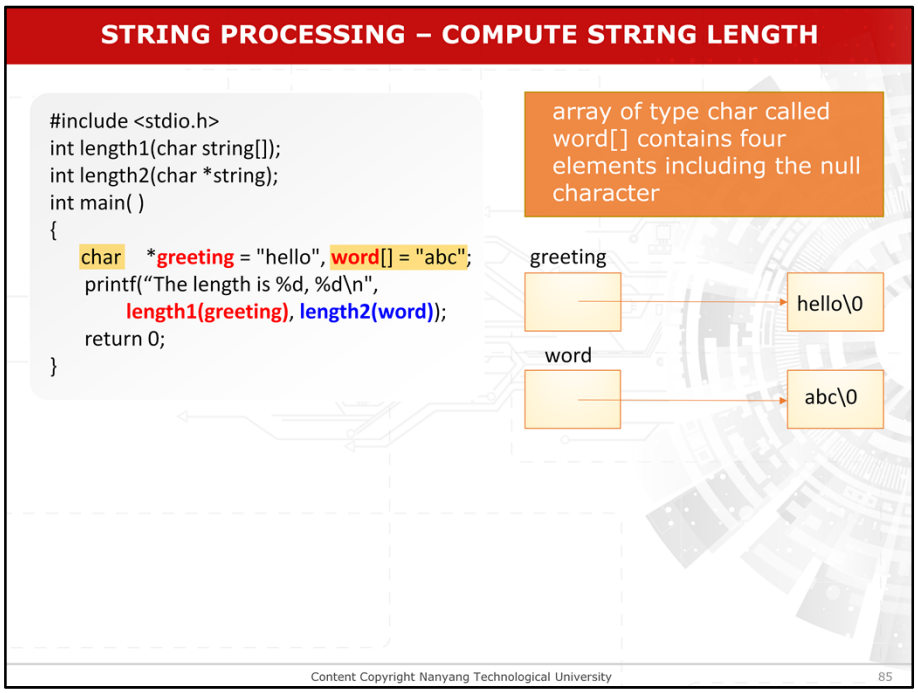
```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char    *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
          length1(greeting), length2(word));
    return 0;
}
```

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

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

character asterisk greeting = double quote hello double quote;

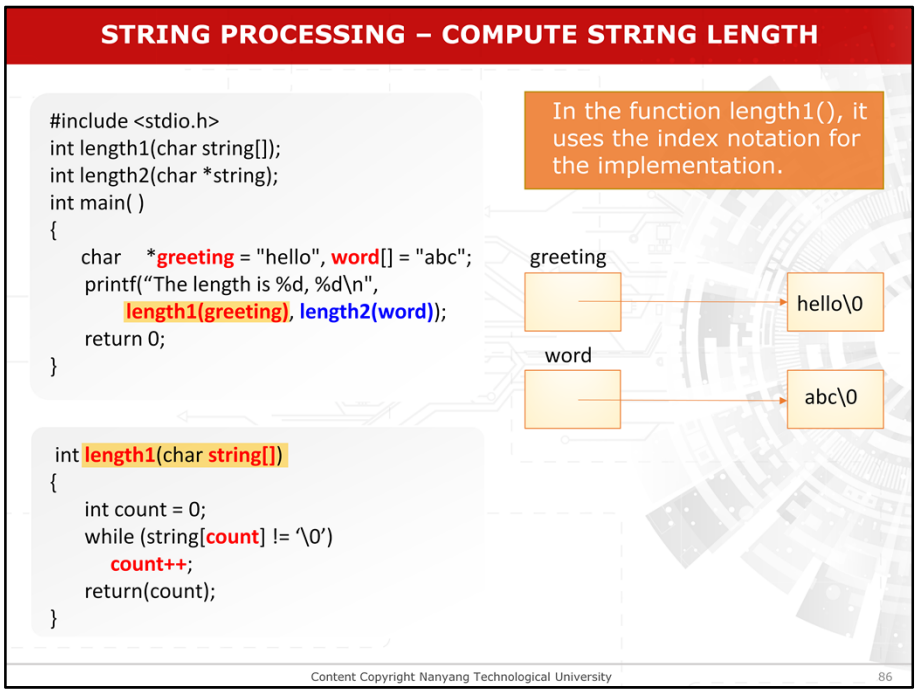
creates a pointer variable called **greeting** that points to the first character of the string "**hello**".



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.



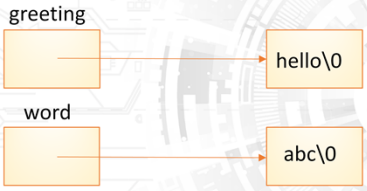
In the function **length 1()**, it uses the index notation for the implementation.

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;
}
```

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

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



The function **length 1()** uses the statement

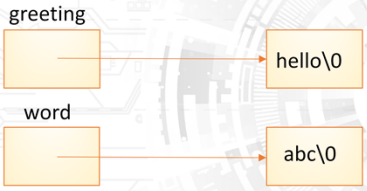
while (string[count] not = null character)

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);
}
```



Output

The length is 5,

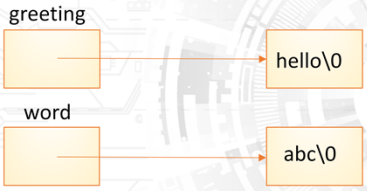
[audio pause]

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;
}
```

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

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

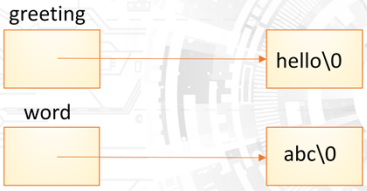


In the function **length 2()**, it uses the pointer notation for the implementation.

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);
}
```

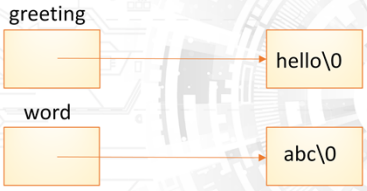

The function **length 2()** uses the statement
while (asterisk(string + count) not = null character**)**
to check for terminating null character.

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);
}
```

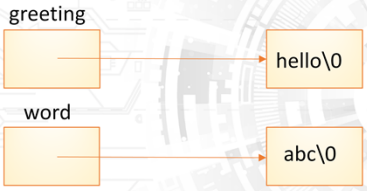
The expression (*(string + count)) first gets the character stored at the address location contained in (string + count).



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;
}
```



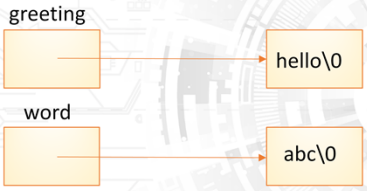

```
int length2(char *string)
{
      int count = 0;
      while ( *(string+count) != '\0')
          count++;
      return(count);
}
```


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

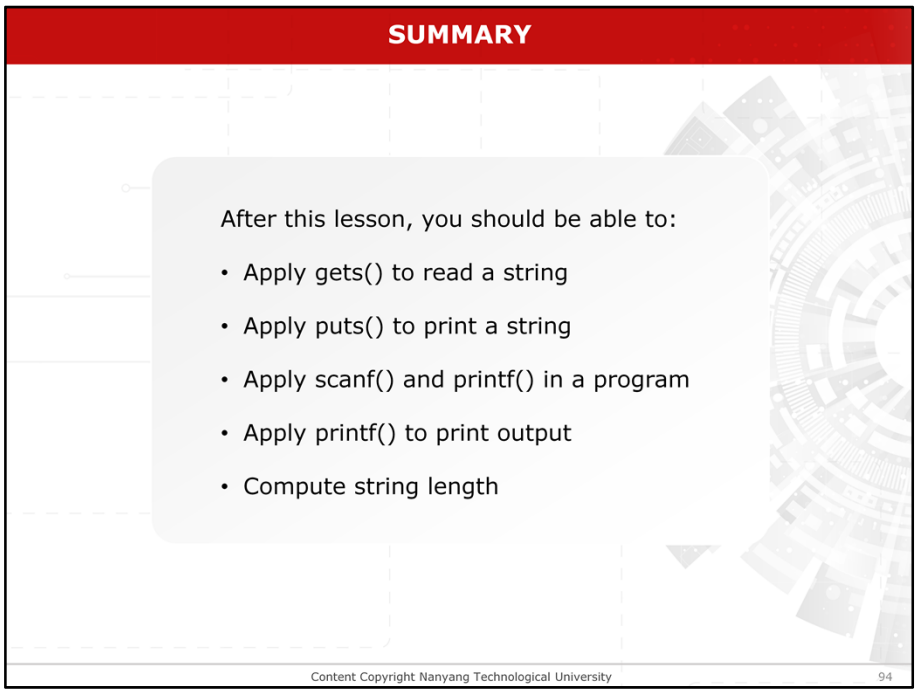
```
#include <stdio.h>
int length1(char string[]);
int length2(char *string);
int main( )
{
    char    *greeting = "hello", word[] = "abc";
    printf("The length is %d, %d\n",
          length1(greeting), length2(word));
    return 0;
}
```

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



Output

The length is 5, 3



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