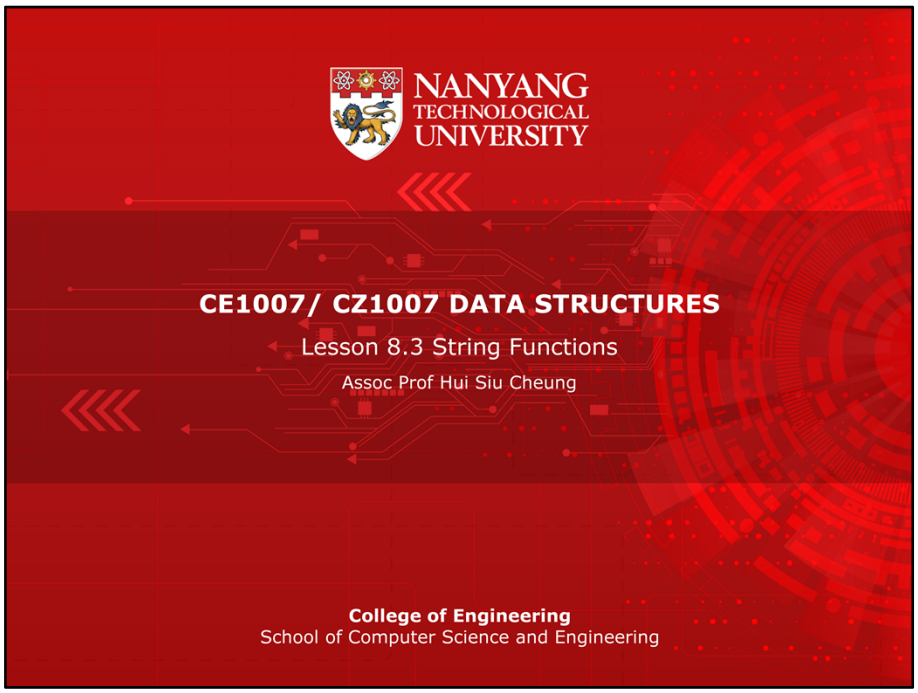
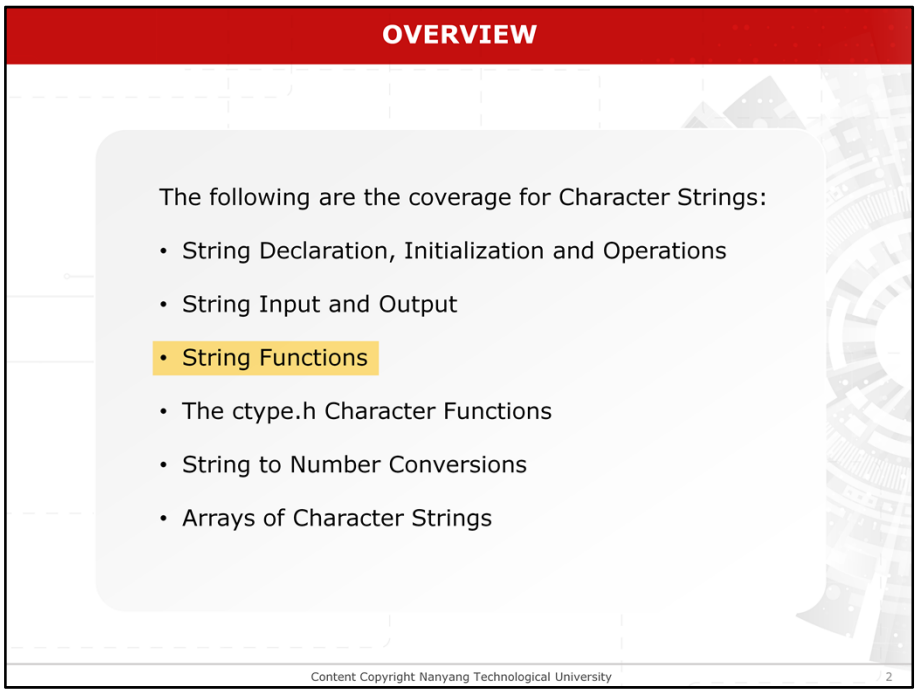
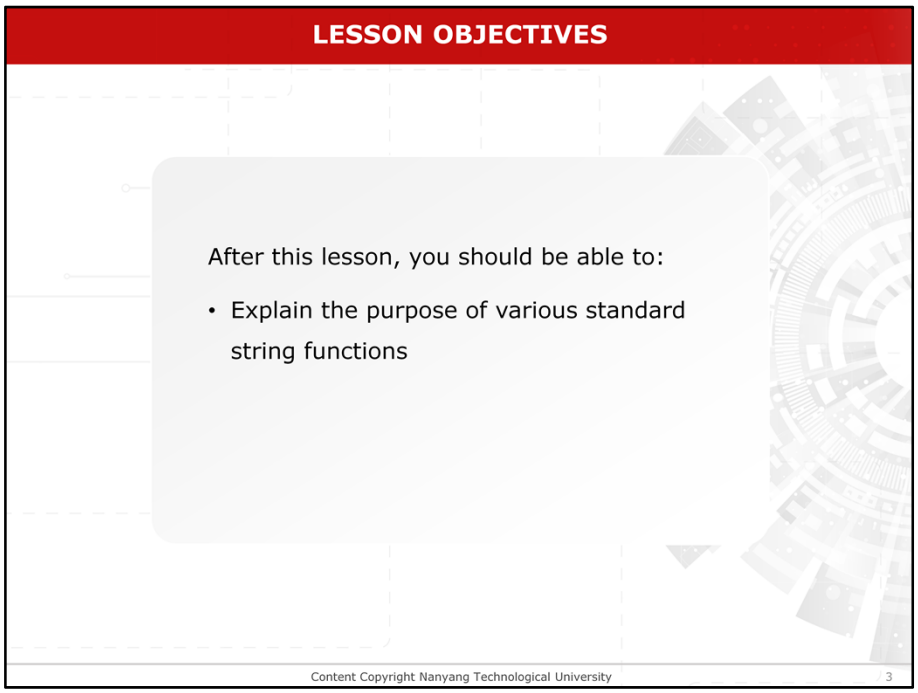

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 8.3 String Functions
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering

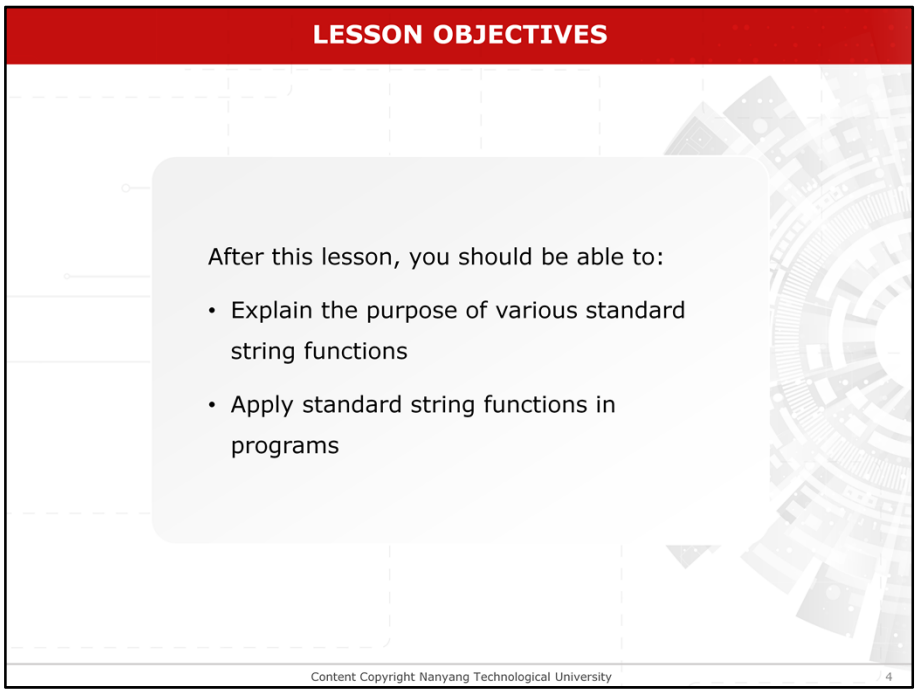


There are 6 main sections to cover for Character Strings as shown. This video lesson focuses on the third part on String Functions.

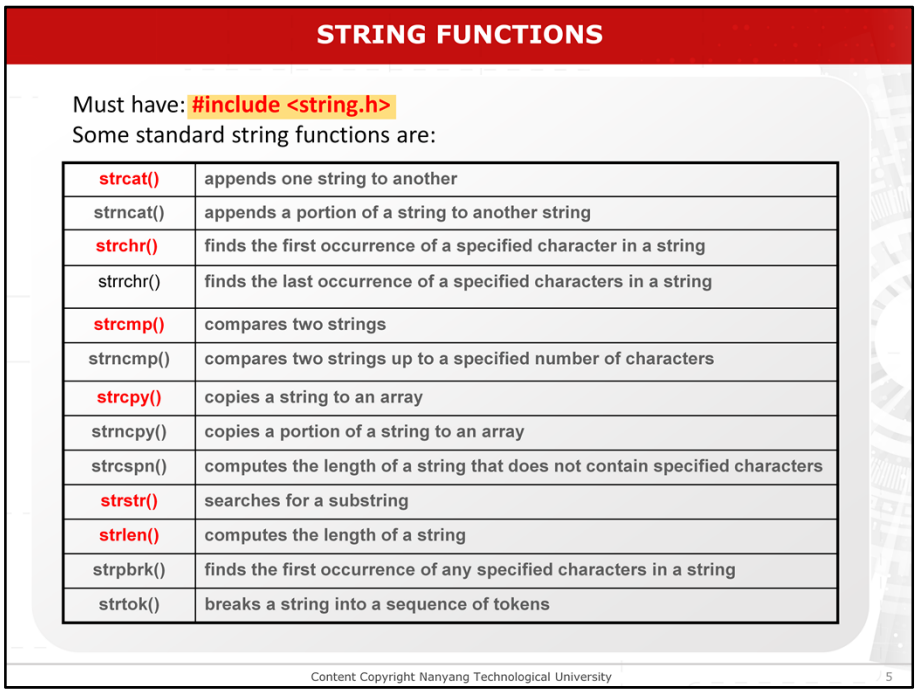


After this lesson, you should be able to:

Explain the purpose of various standard string functions



Apply standard string functions in programs



The standard C library provides many functions that perform string operations. To use any of these functions, we must include the header file *string dot h* in the program:

pound include string dot h

String functions are very useful for writing programs that involve the manipulation of strings. Some examples of string functions include finding the length of strings, combining two strings, comparing two strings, and searching a string for a specific character. Programmers are encouraged to use these functions instead of developing their own functions. In the table, it lists some of the more commonly used string functions provided in *string dot h*

In this lesson, we describe some of the most useful C string handling functions.

STRING FUNCTIONS

Must have: **#include <string.h>**
Some standard string functions are:

strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

The **string length** function computes the length of a string.

STRING FUNCTIONS

Must have: **#include <string.h>**
Some standard string functions are:

strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

The **string cat** function appends one string to another.

STRING FUNCTIONS

Must have: **#include <string.h>**
Some standard string functions are:

strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

The **string copy** function copies a string to an array.

STRING FUNCTIONS

Must have: **#include <string.h>**
Some standard string functions are:

strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

The **string compare** function compares two strings.

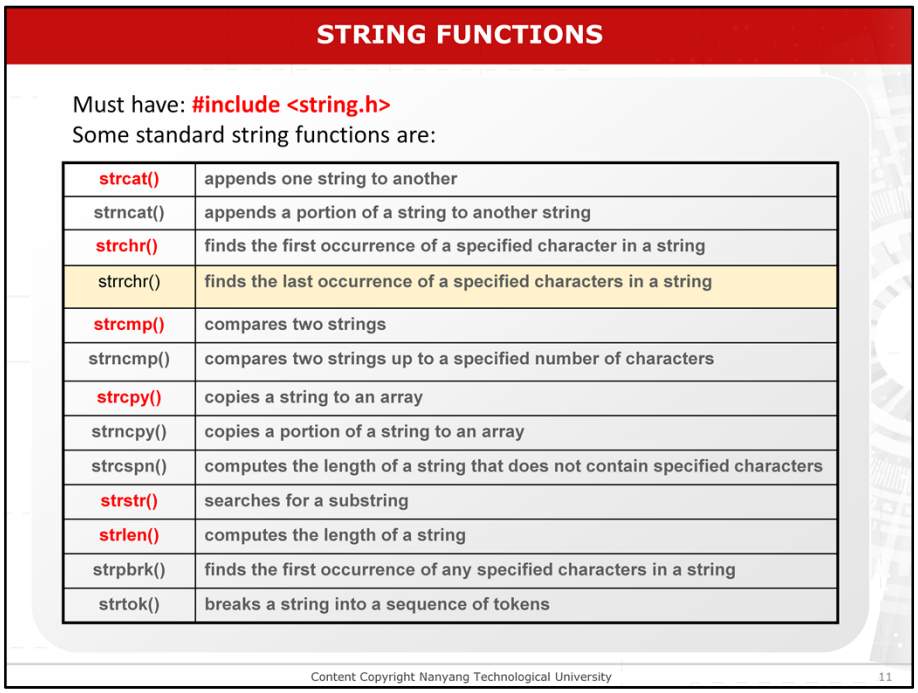
STRING FUNCTIONS

Must have: **#include <string.h>**
Some standard string functions are:

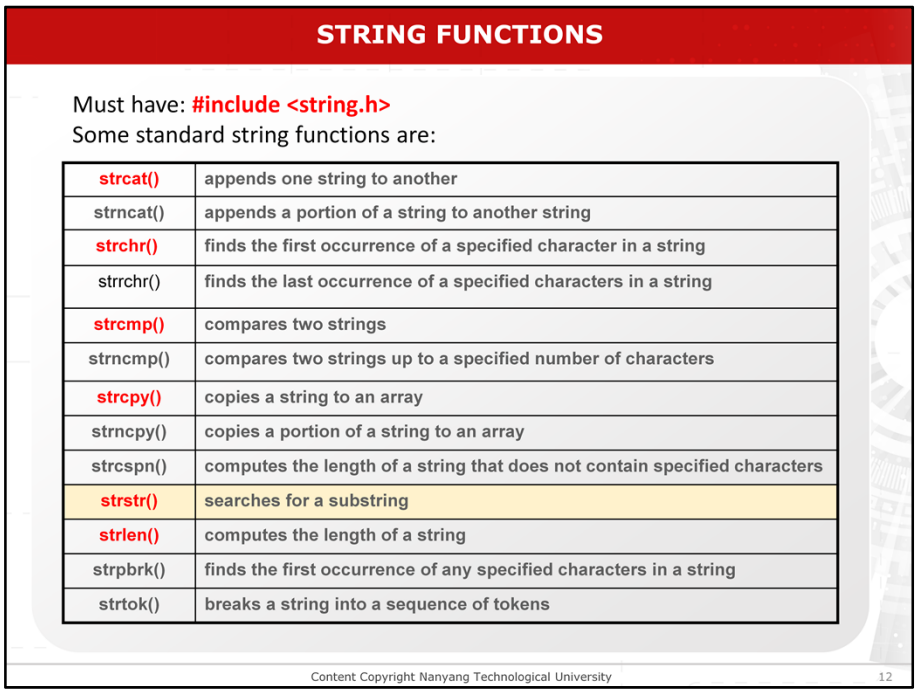
strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

The **string character** function searches the first occurrence of the leftmost character in the string, and returns a pointer to that occurrence.

A null pointer will be returned if it does not find the character.



The **string r character** function works similarly to **string character** except that it searches for the last rightmost occurrence of the character in the string, and returns a pointer to that occurrence. Similarly, a null pointer will be returned if it does not find the character in the string.



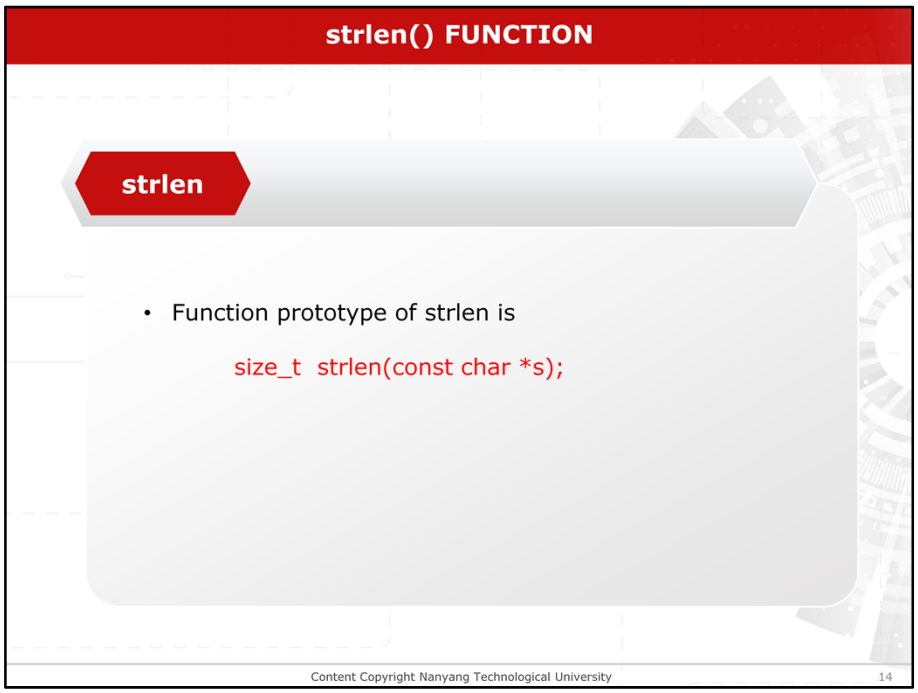
The **string string** function searches for a substring.

STRING FUNCTIONS

Must have: **#include <string.h>**
Some standard string functions are:

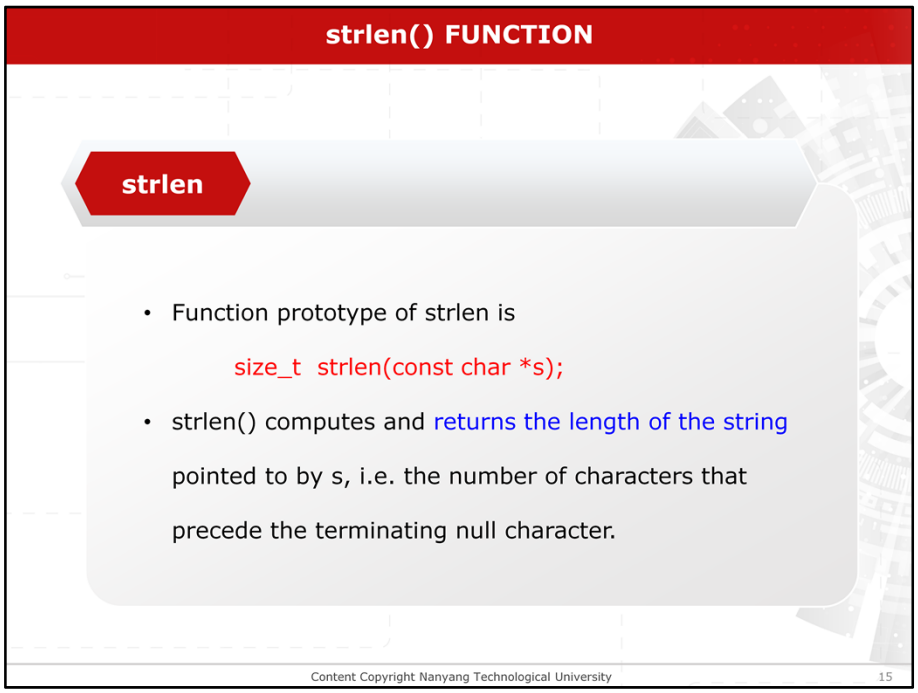
strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array
strcspn()	computes the length of a string that does not contain specified characters
strstr()	searches for a substring
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

In the subsequent slides, we will discuss the 4 most common functions: **string length()**, **string cat()**, **string copy()** and **string compare()**.

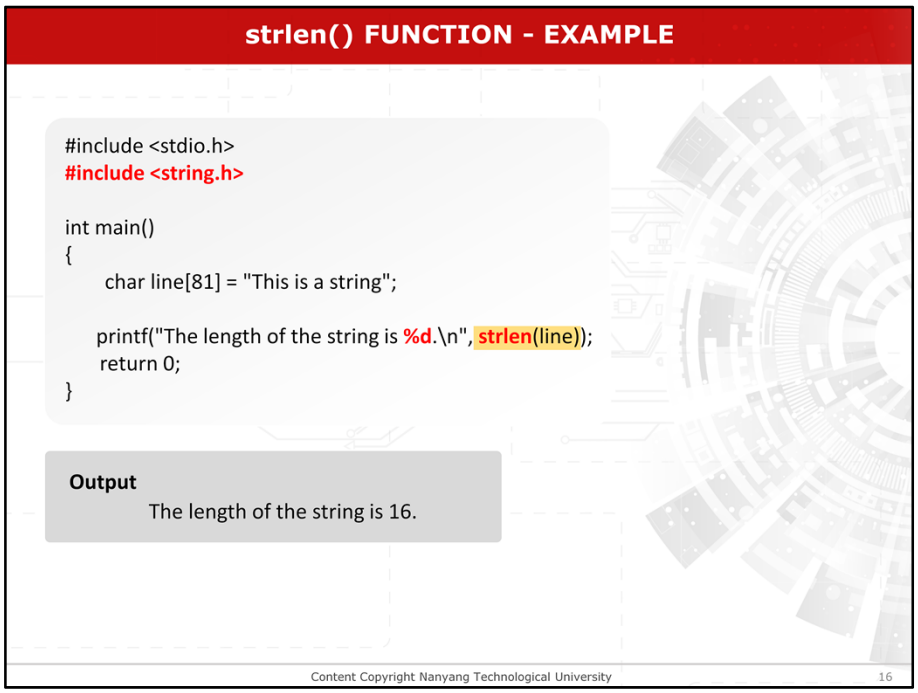


The **string length()** function finds the length of a string. The function prototype of **string length()** is

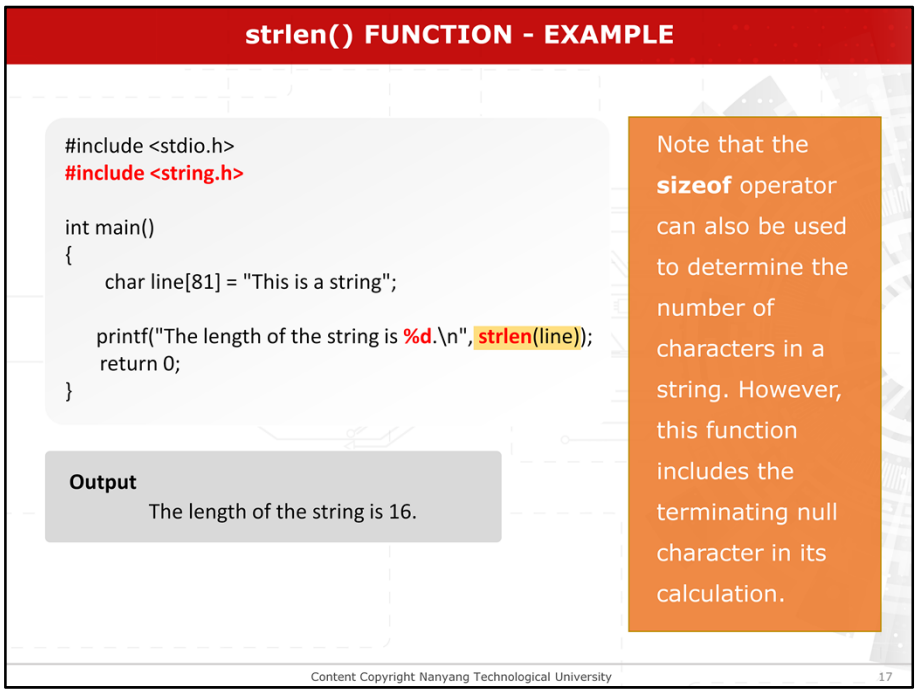
unsigned string length constant character asterisk string



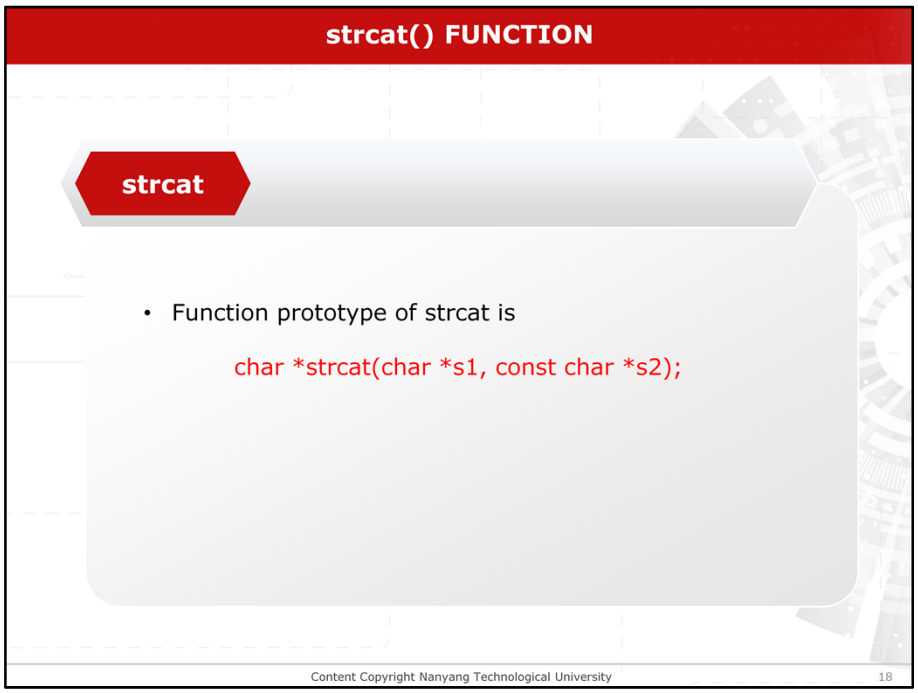
String length() computes the length of the string pointed to by **string**. It returns the number of characters that precede the terminating null character. The null character is excluded in the calculation.



In the program, it uses the **string length()** function to obtain the length of the string variable **line**. The program creates a character array called **line[]** to store the string. The character array **line[]** is initialized to the character string constant "**This is a string**". The length of this string is then calculated using the **string length()** function, and printed on the display.

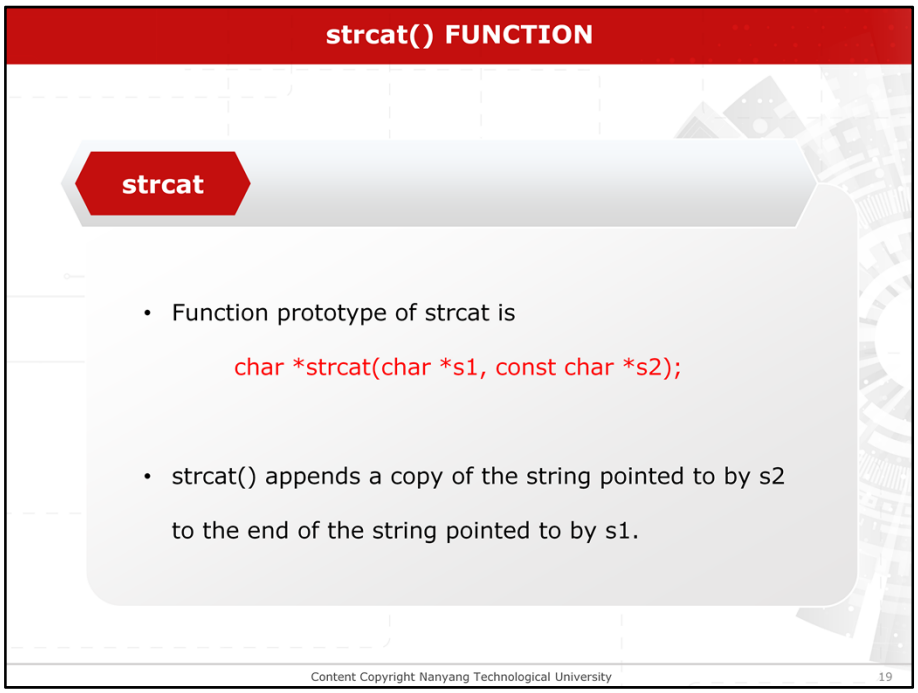


Note that the **size of** operator can also be used to determine the number of characters in a string. However, this function includes the terminating null character in its calculation.

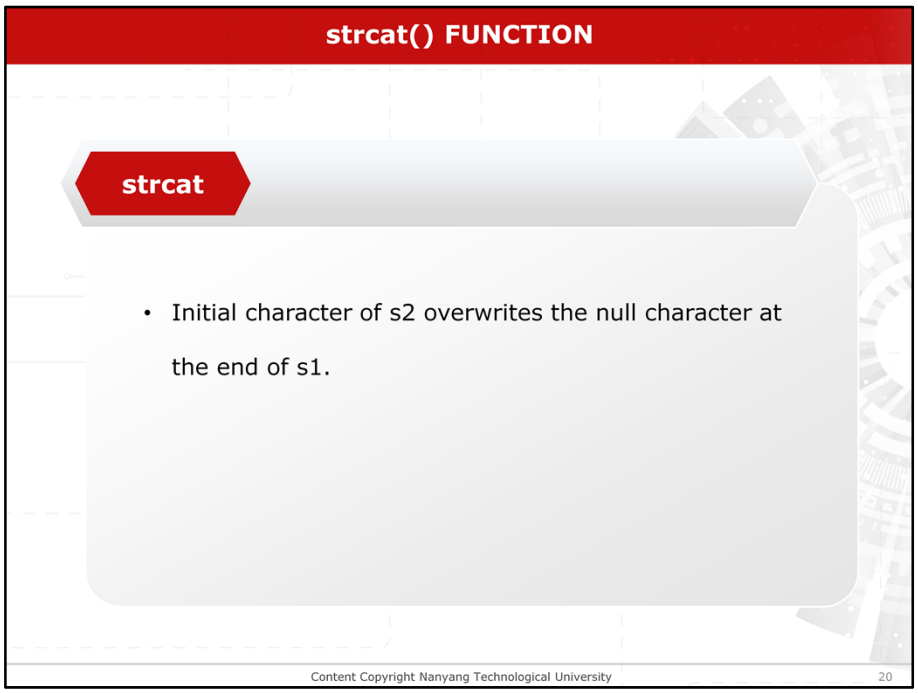


The **string cat()** function concatenates two strings to form a new string. The function prototype of **string cat()** is

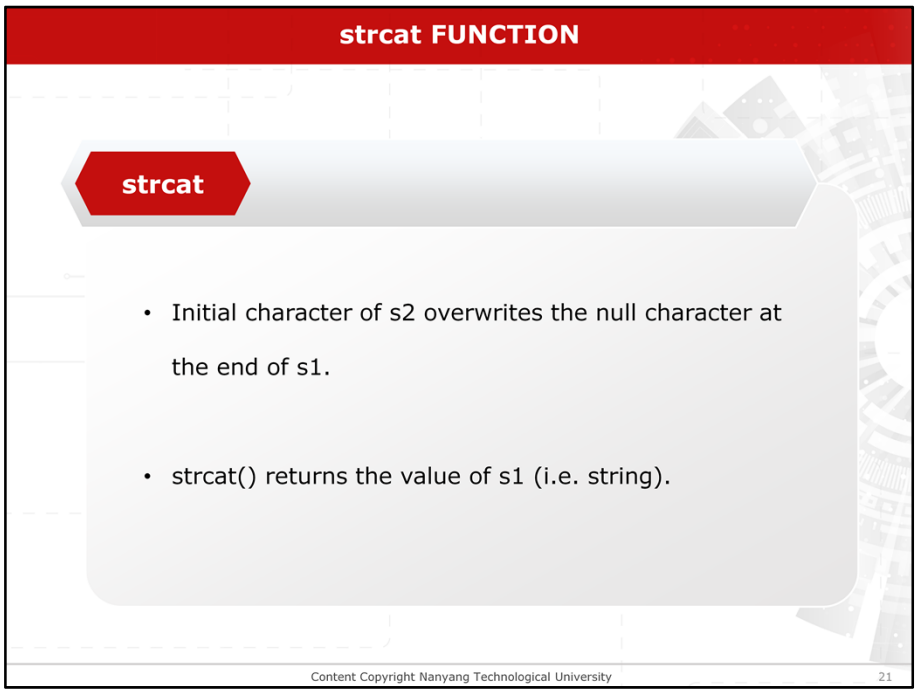
character asterisk string cat(character asterisk string 1, constant character asterisk string 2



String cat() appends a copy of the string pointed to by **string2** to the end of the string pointed to by **string1**.



The initial character of **string2** overwrites the null character at the end of **string1**.



The initial character of **string2** overwrites the null character at the end of **string1**.
String cat() returns the address value of **string 1**.
And **string 2** is unchanged.

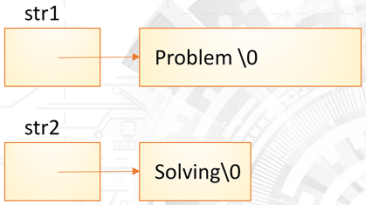
strcat() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[40] = "Problem ";
    char *str2 = "Solving";

    printf("The first string: %s\n", str1);
    printf("The second string: %s\n", str2);
    strcat(str1, str2);
    printf("The combined string: %s\n", str1);
    return 0;
}
```


In the program, it uses the **string cat()** function to concatenate two strings.

strcat() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[40] = "Problem ";
    char *str2 = "Solving";

    printf("The first string: %s\n", str1);
    printf("The second string: %s\n", str2);
    strcat(str1, str2);
    printf("The combined string: %s\n", str1);
    return 0;
}
```

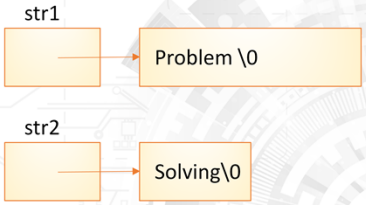


The two strings are declared and initialized as follows:

character array string1 of size 40 = double quote Problem double quote
character asterisk string 2 = double quote Solving double quote

strcat() FUNCTION - EXAMPLE

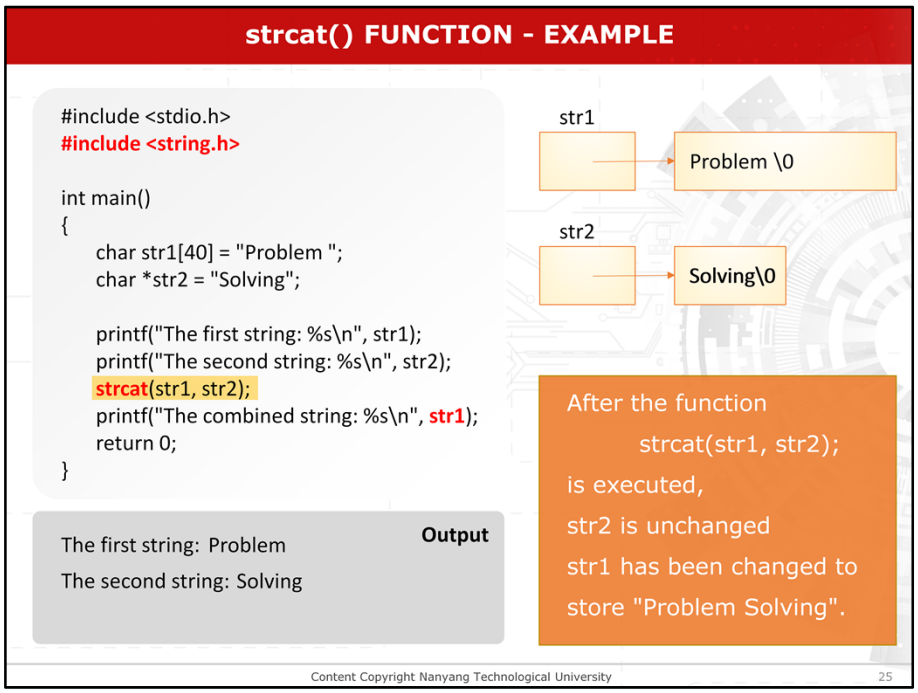
```
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[40] = "Problem ";
    char *str2 = "Solving";

    printf("The first string: %s\n", str1);
    printf("The second string: %s\n", str2);
    strcat(str1, str2);
    printf("The combined string: %s\n", str1);
    return 0;
}
```



Output

```
The first string:  Problem
The second string: Solving
```


[audio pause]



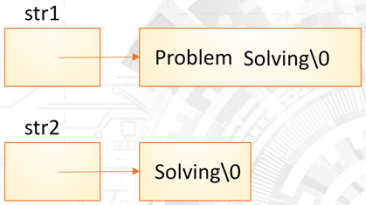
After the function

string cat(string1, string2);

is executed, **string2** is unchanged and **string1** has been changed to store **"Problem Solving"**.

strcat() FUNCTION - EXAMPLE

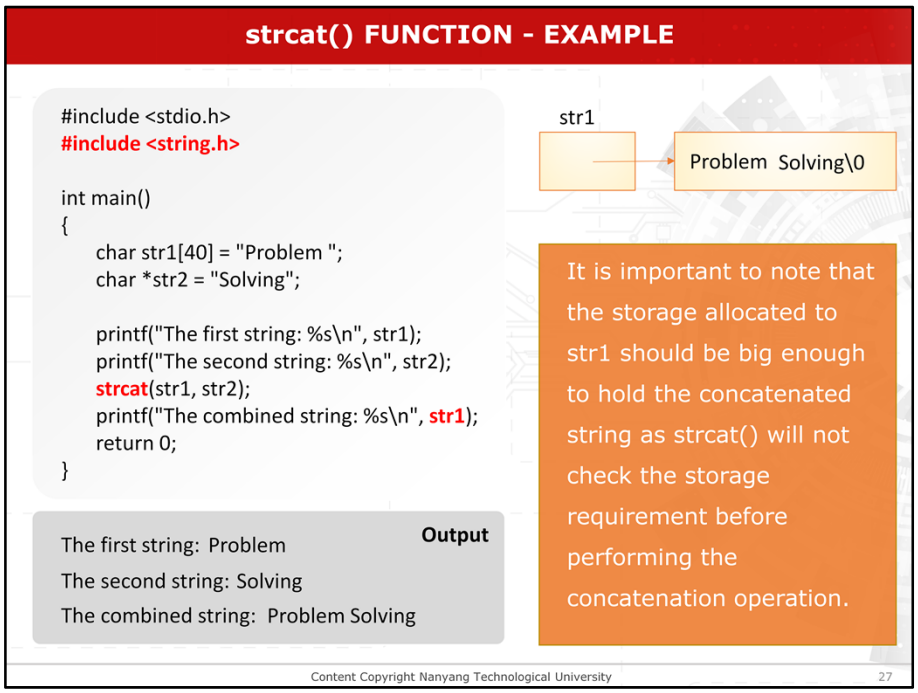
```
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[40] = "Problem ";
    char *str2 = "Solving";

    printf("The first string: %s\n", str1);
    printf("The second string: %s\n", str2);
    strcat(str1, str2);
    printf("The combined string: %s\n", str1);
    return 0;
}
```



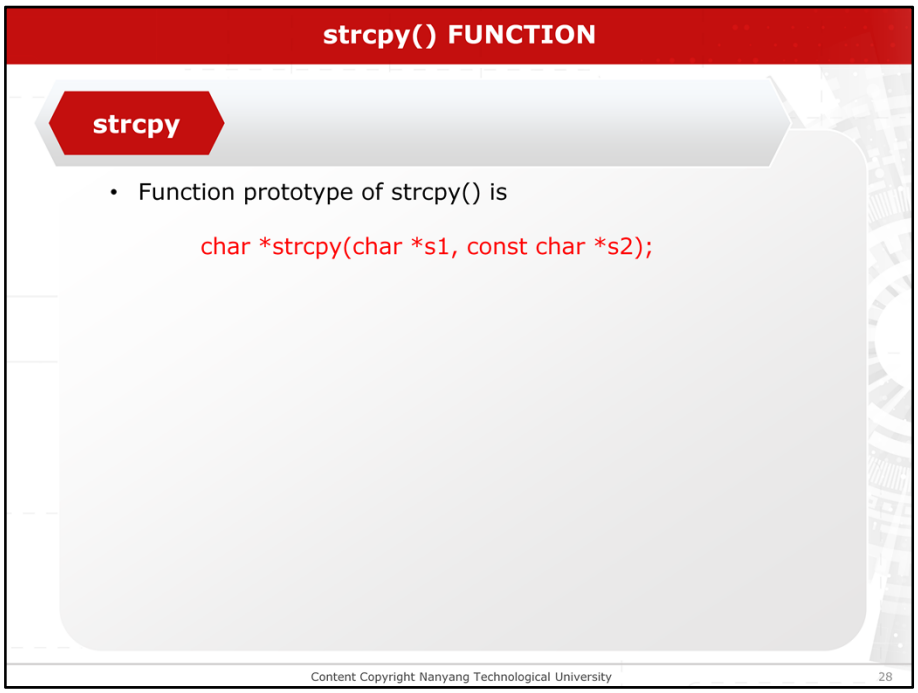
Output

```
The first string:  Problem
The second string: Solving
The combined string:  Problem Solving
```


[audio pause]

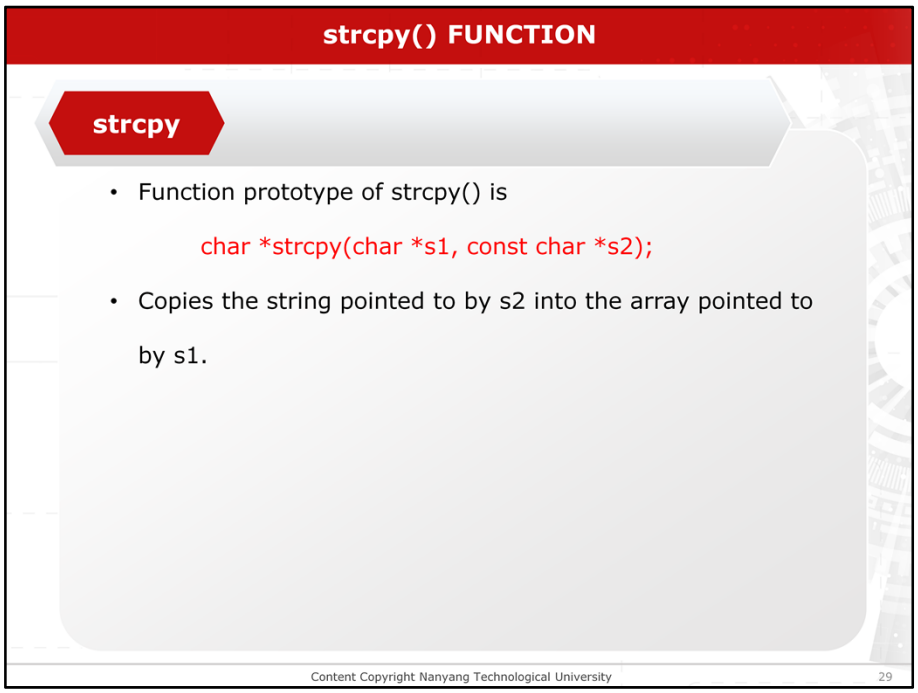


It is important to note that the storage allocated to **string1** should be big enough to hold the concatenated string as **string cat()** will not check the storage requirement before performing the concatenation operation.

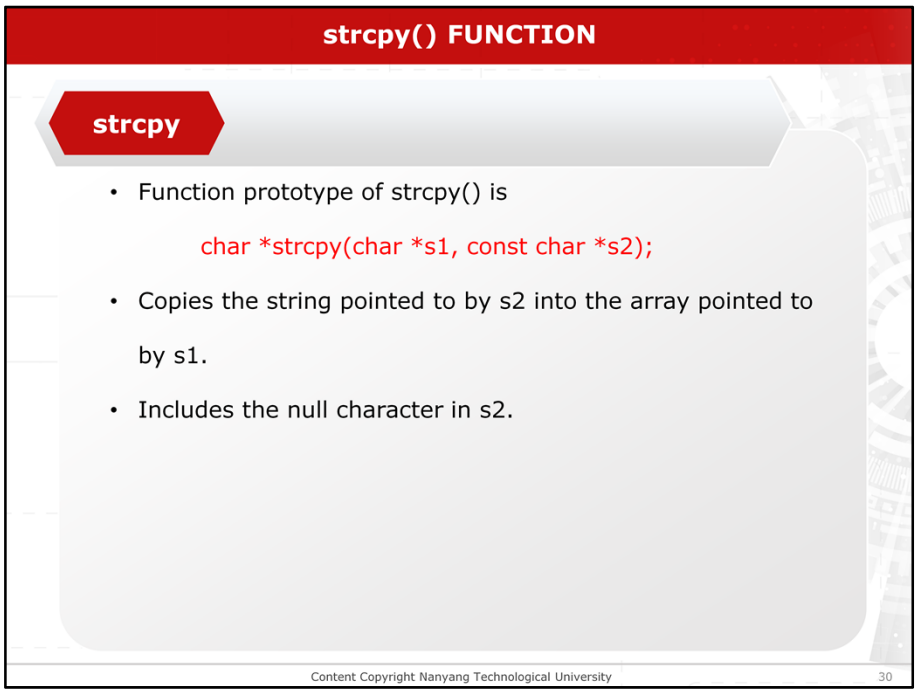


The **string copy()** function copies one string to another. The function prototype of **string copy()** is

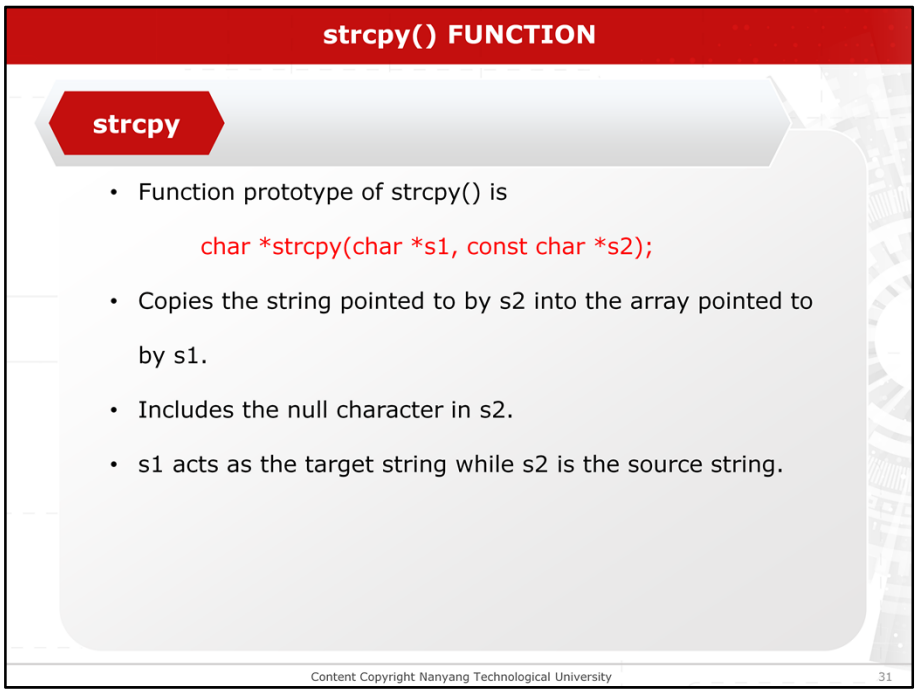
character asterisk string copy(character asterisk string 1, constant character asterisk string 2



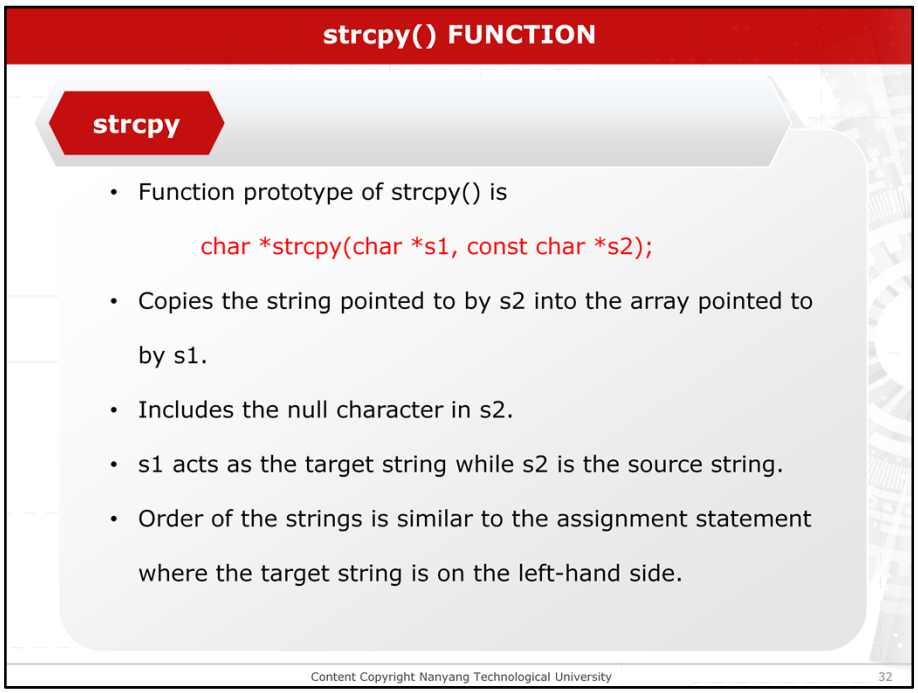
The function copies all the characters in the string pointed to by **string2** into the array pointed to by **string1**.



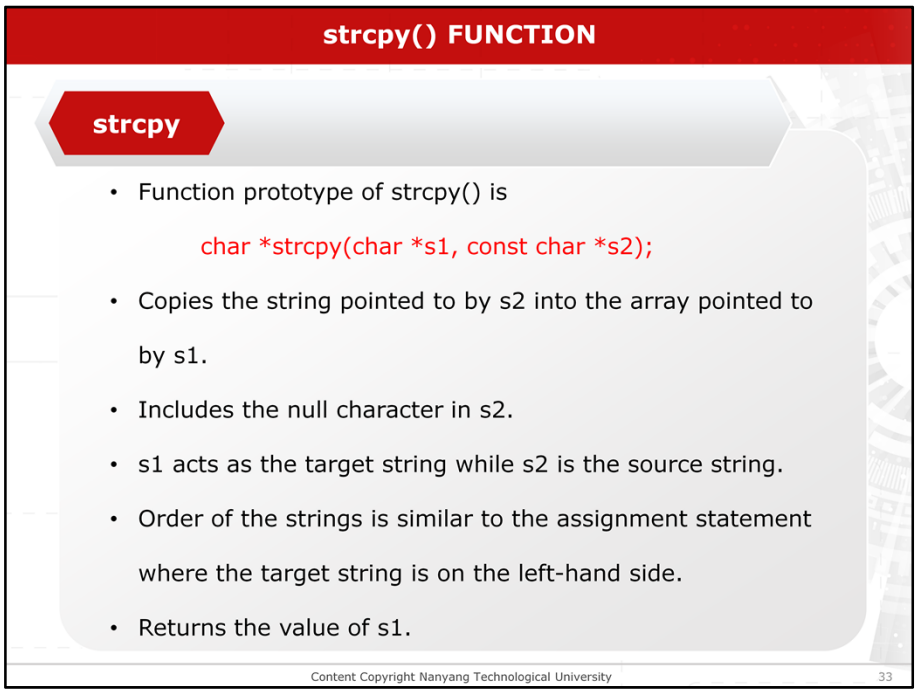
The copy operation includes the null character in **string2**.



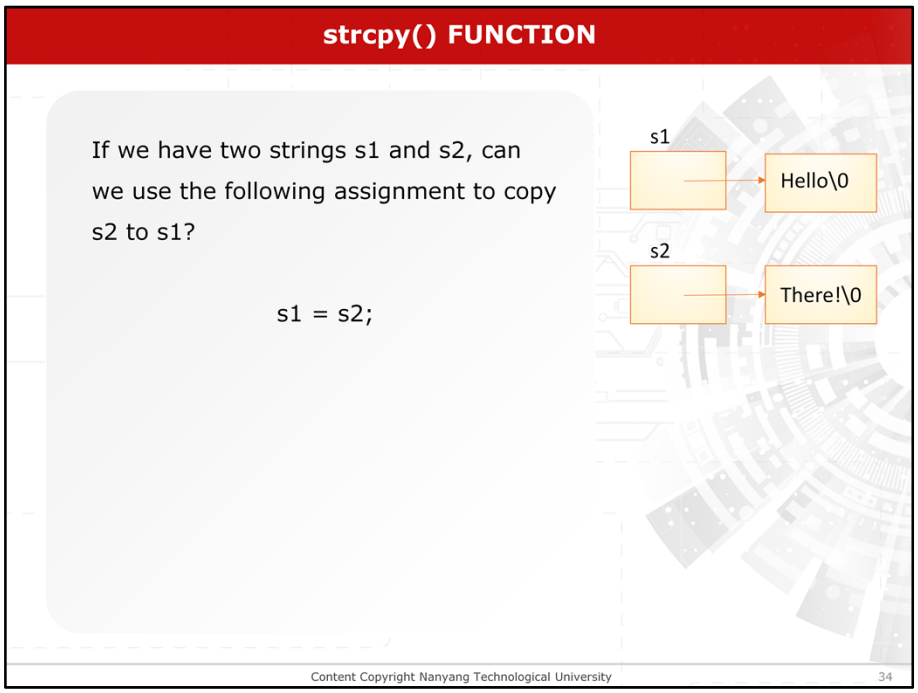
string1 acts as the target string while **string2** is the source string.



The order of the strings is similar to the assignment statement where the target string is on the left-hand side.



The **string copy()** function returns the value of **string1**.



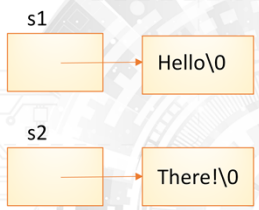
If we have two strings s1 and s2, can we use the assignment s1 = s2 to copy s2 to s1?

strcpy() FUNCTION

If we have two strings s1 and s2, can we use the following assignment to copy s2 to s1?

```
s1 = s2;
```

The answer is NO.



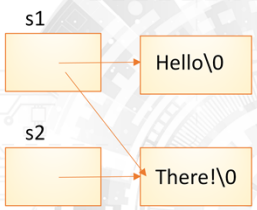
The answer is NO.

strcpy() FUNCTION

If we have two strings s1 and s2, can we use the following assignment to copy s2 to s1?

```
s1 = s2;
```

The answer is NO. The assignment operation will only assign pointer s2 to pointer s1, not updating the contents pointed to by s1.



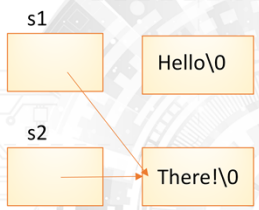
The assignment statement will only assign pointer **s2** to pointer **s1**, it does not update the contents pointed to by **s1**.

strcpy() FUNCTION

If we have two strings s1 and s2, can we use the following assignment to copy s2 to s1?

```
s1 = s2;
```

The answer is NO. The assignment operation will only assign pointer s2 to pointer s1, not updating the contents pointed to by s1.



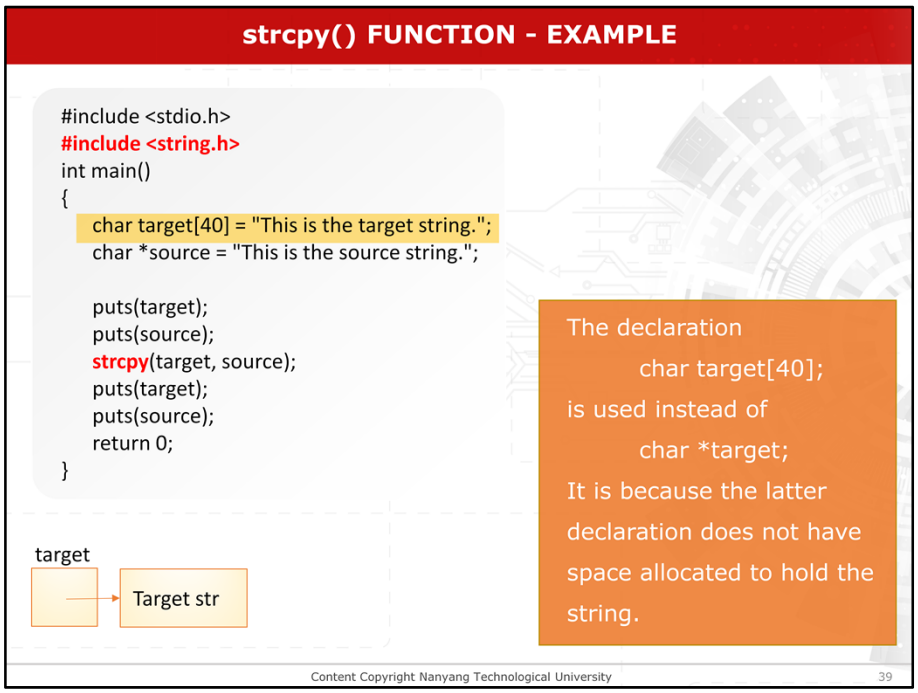
Use the strcpy() function when you want to copy the content of a string to another string.

Therefore, you need to use the **string copy()** function when you want to copy the content of a string to another string.

strcpy() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main()
{
    char target[40] = "This is the target string.";
    char *source = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);
    return 0;
}
```


In the program, it use **string copy()** function to copy a string to another string.



It is important to note that the target array must have enough space to hold the updated string contents.
The declaration

character array target of size 40;

is used instead of

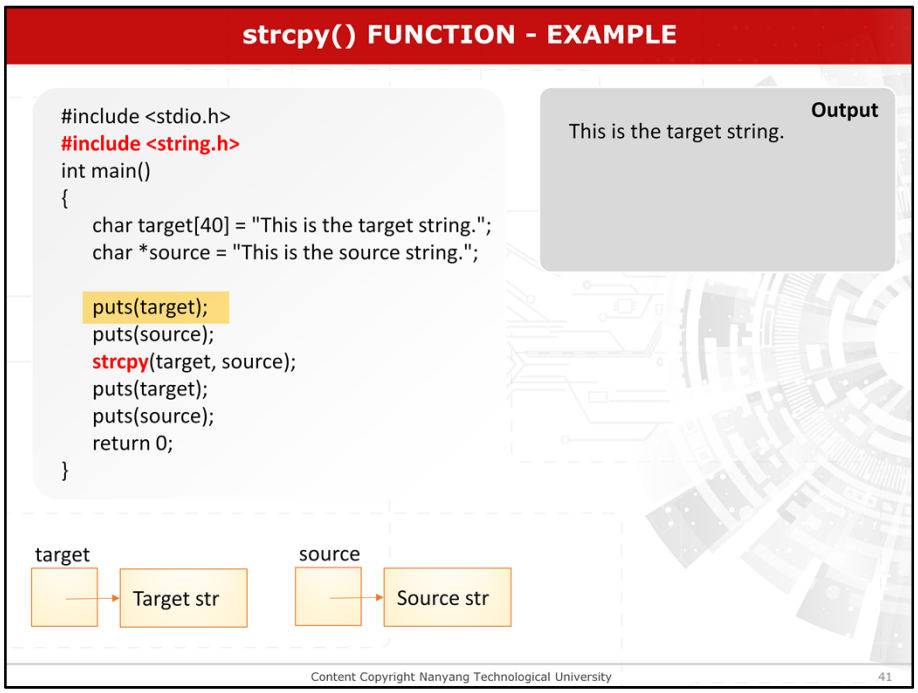
character asterisk target;

It is because the latter declaration does not have space allocated to hold the string.

strcpy() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main()
{
    char target[40] = "This is the target string.";
    char *source = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);
    return 0;
}
```

[audio pause]


strcpy() FUNCTION - EXAMPLE
#include <stdio.h>
#include <string.h>
int main()
{
char target[40] = "This is the target string.";
char *source = "This is the source string.";
puts(target);
puts(source);
strcpy(target, source);
puts(target);
puts(source);
return 0;
}
Output
This is the target string.
target
Target str
source
Source str
Content Copyright Nanyang Technological University
41

[audio pause]

strcpy() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main()
{
    char target[40] = "This is the target string.";
    char *source = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);
    return 0;
}
```

Output

```
This is the target string.
This is the source string.
```

[audio pause]

strcpy() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main()
{
    char target[40] = "This is the target string.";
    char *source = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);
    return 0;
}
```

Output

```
This is the target string.
This is the source string.
```

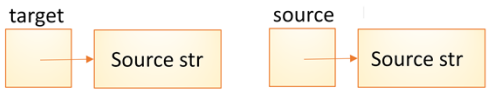
The string copy() function copy the source string to the target string

strcpy() FUNCTION - EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main()
{
    char target[40] = "This is the target string.";
    char *source = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);
    return 0;
}
```

Output

```
This is the target string.
This is the source string.
This is the source string.
```



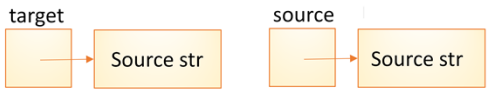
So the target now is “This is the source string”.

strcpy() FUNCTION - EXAMPLE

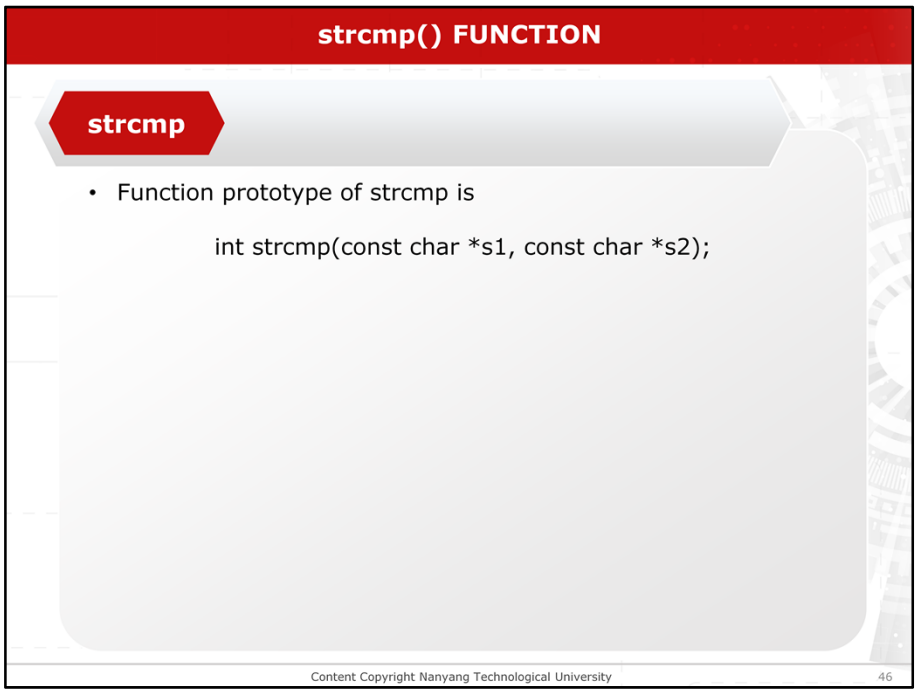
```
#include <stdio.h>
#include <string.h>
int main()
{
    char target[40] = "This is the target string.";
    char *source = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);
    return 0;
}
```

Output

```
This is the target string.
This is the source string.
This is the source string.
This is the source string.
```

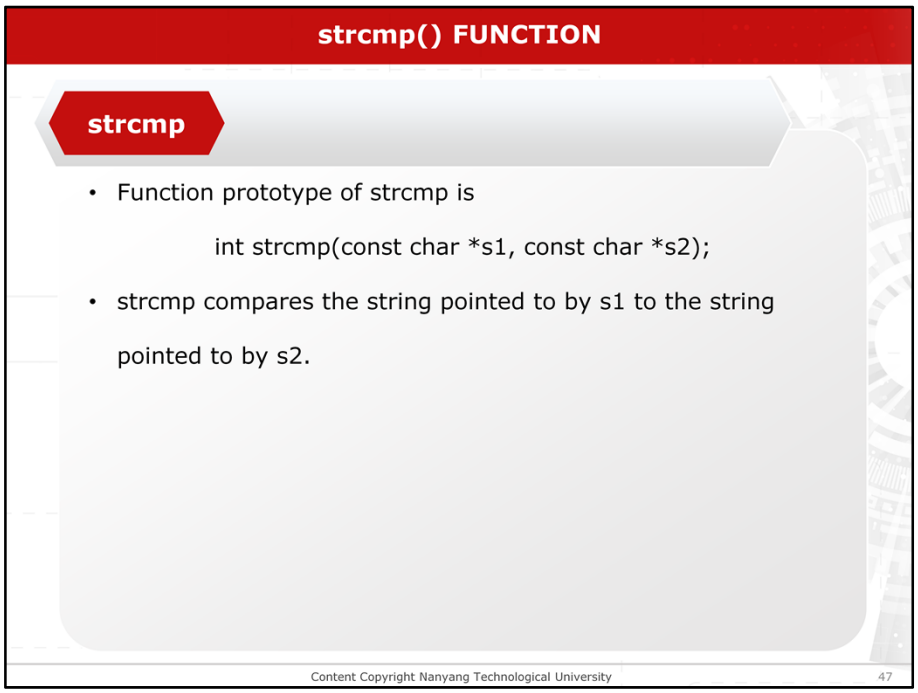


And the source remains as “This is the source string”.

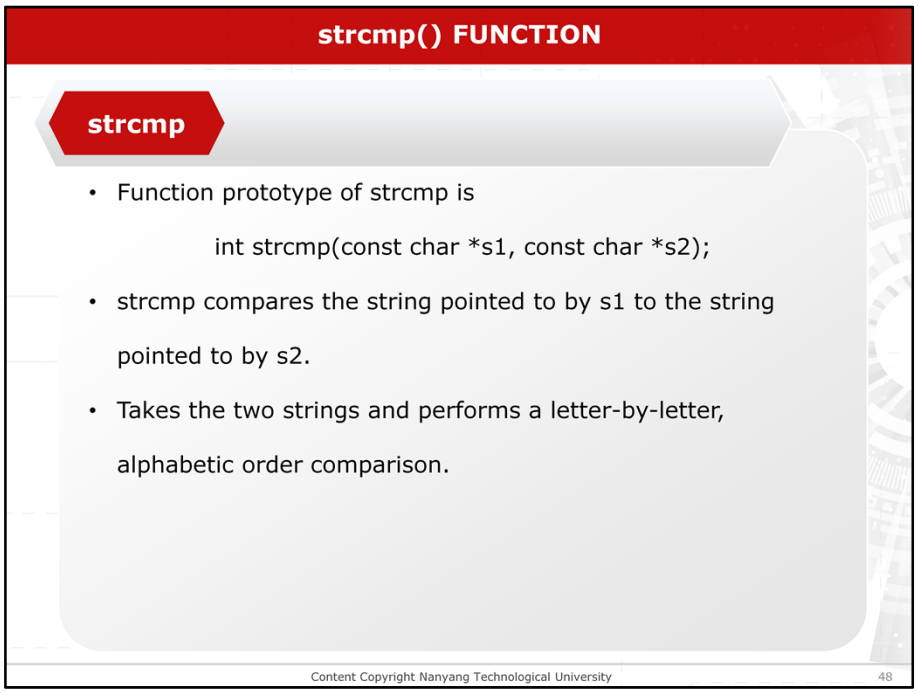


The **string compare()** function compares the contents of two strings. The prototype of **string compare()** is

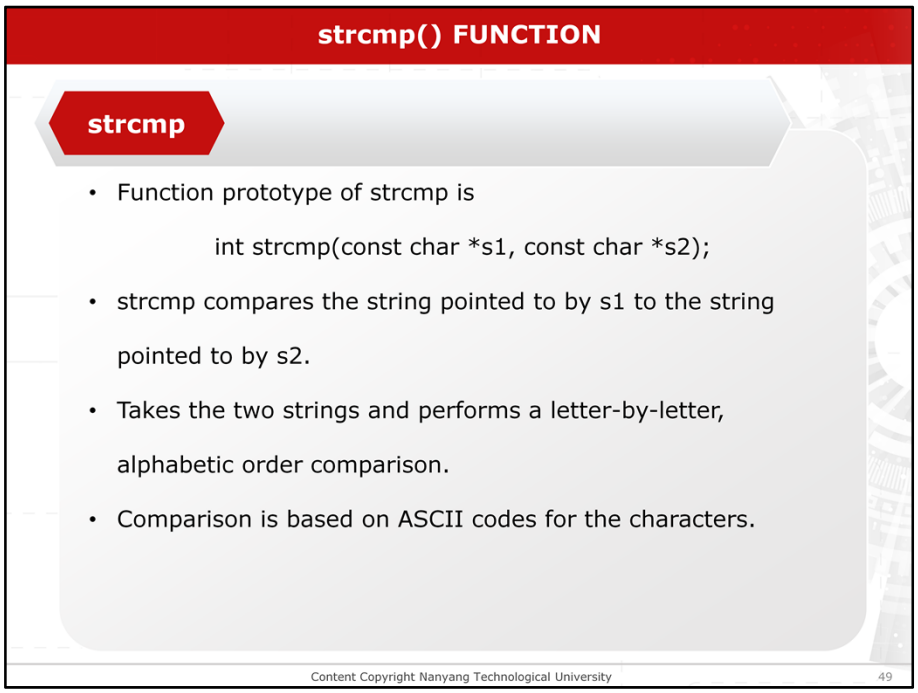
integer string compare (constant character asterisk string 1, constant character asterisk string 2



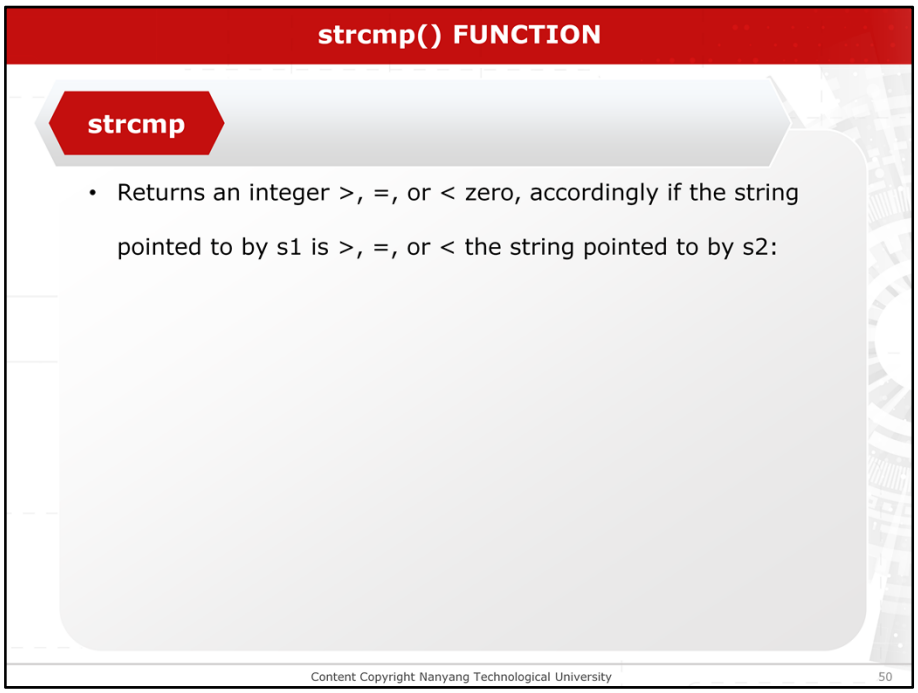
It compares the string pointed to by **string1** to the string pointed to by **string2**.



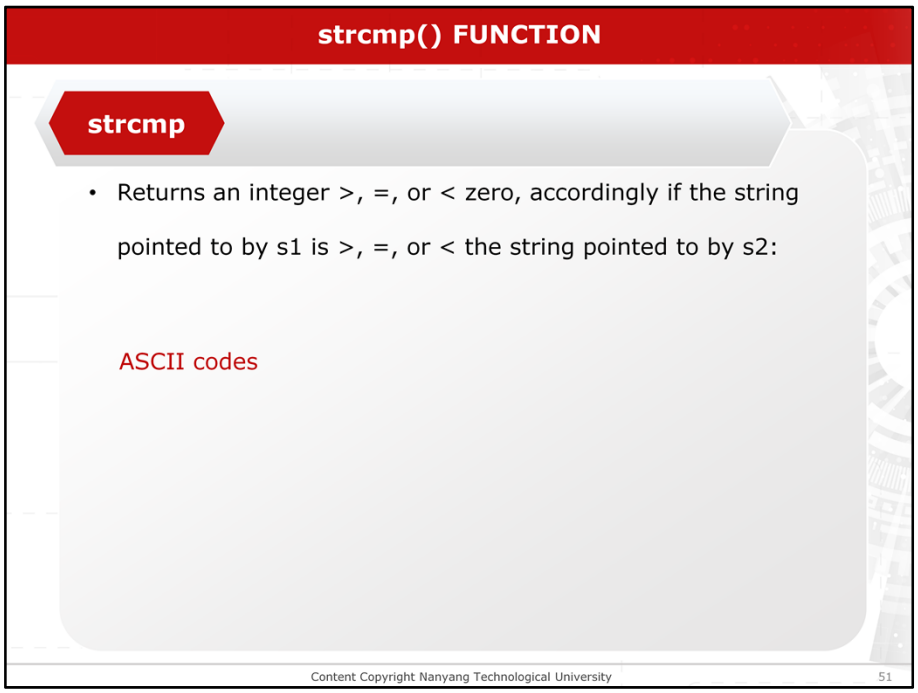
It takes the two strings and performs a letter-by-letter, alphabetic order comparison.



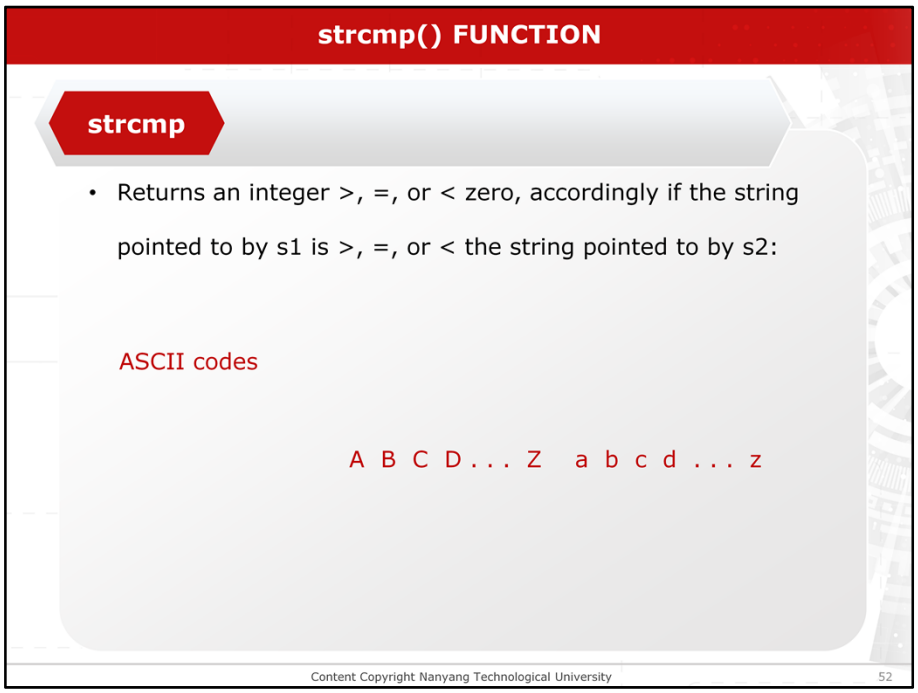
The comparison is based on ASCII codes for the characters



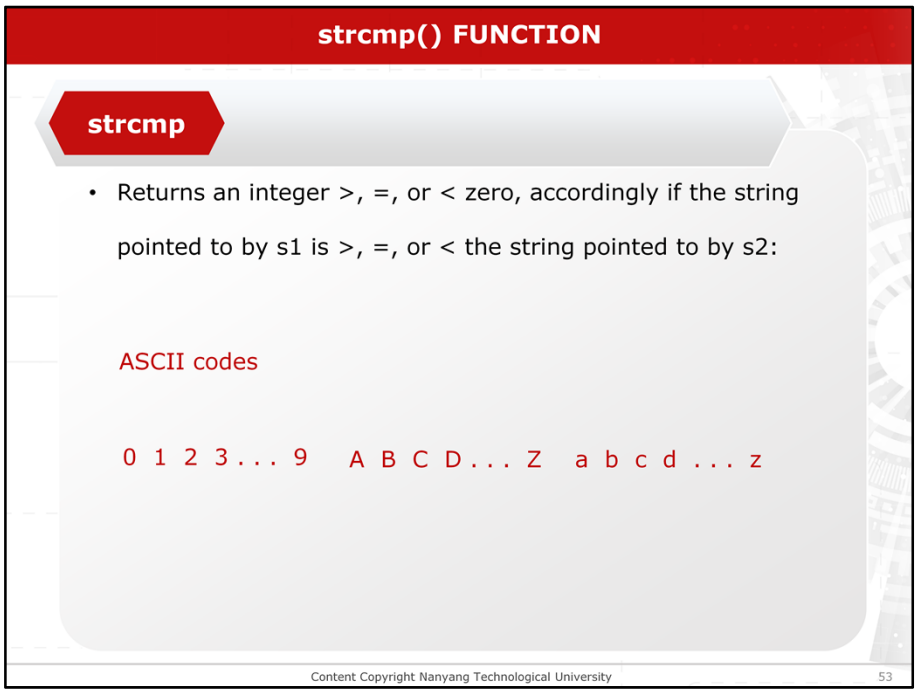
It returns an integer greater than, equal to or less than zero, according to whether the string pointed to by **string1** is greater than, equal to or less than the string pointed to by **string2** respectively.



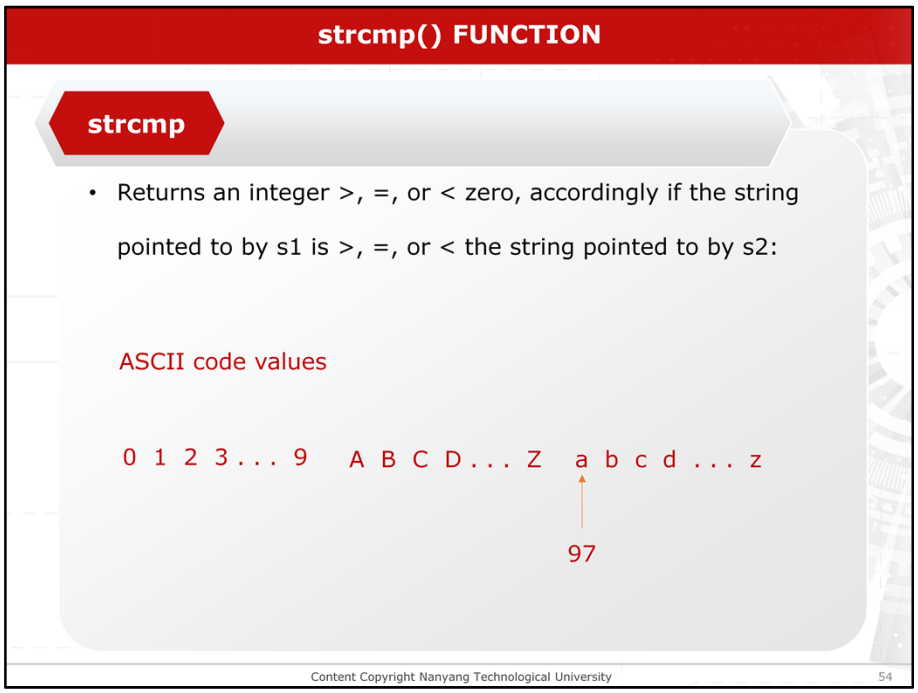
It is interesting to note that in ASCII codes,



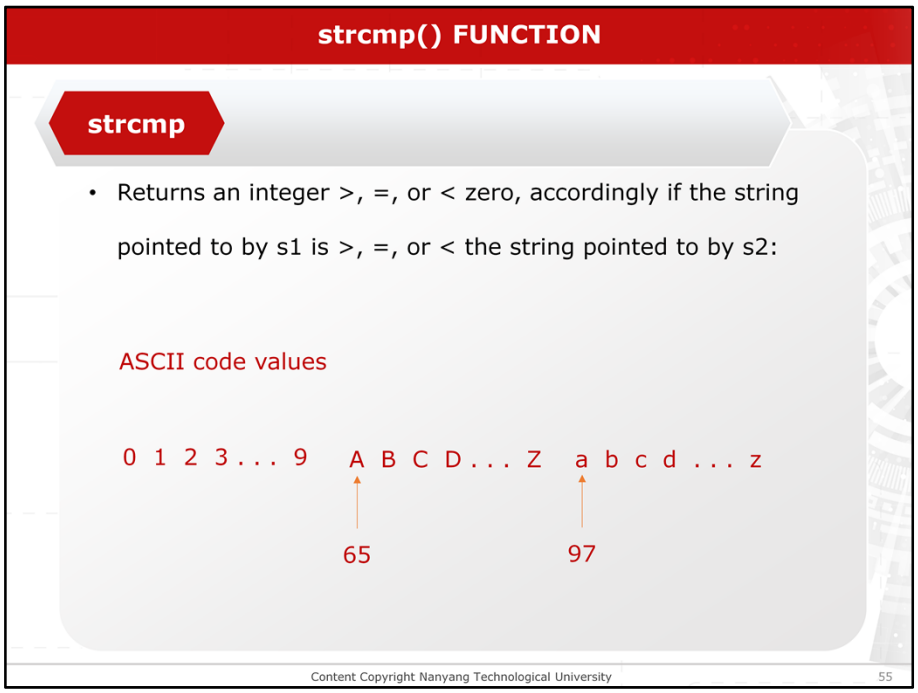
uppercase characters come before lowercase characters



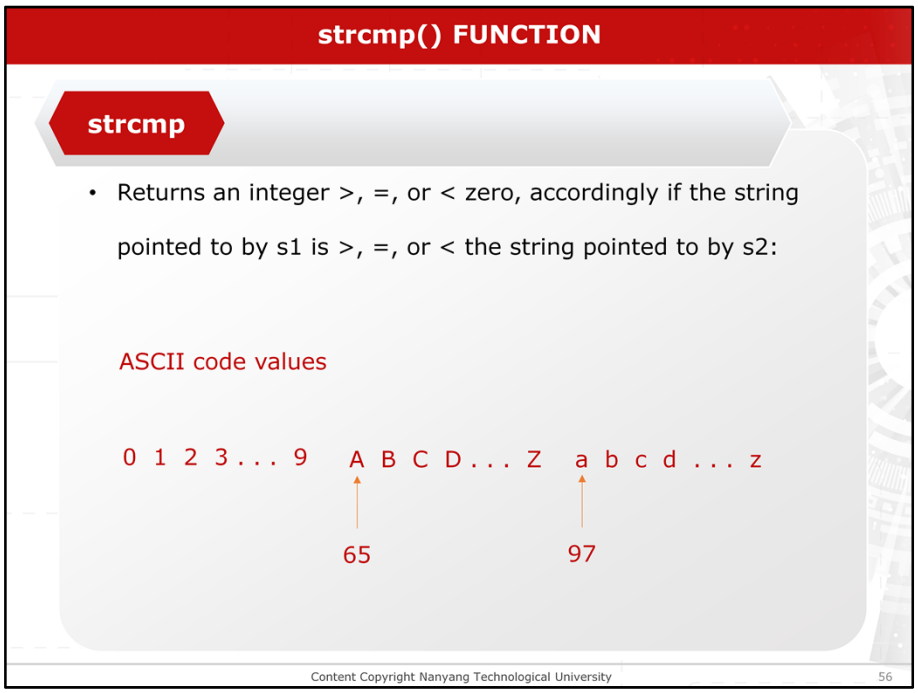
and digits come before the letters



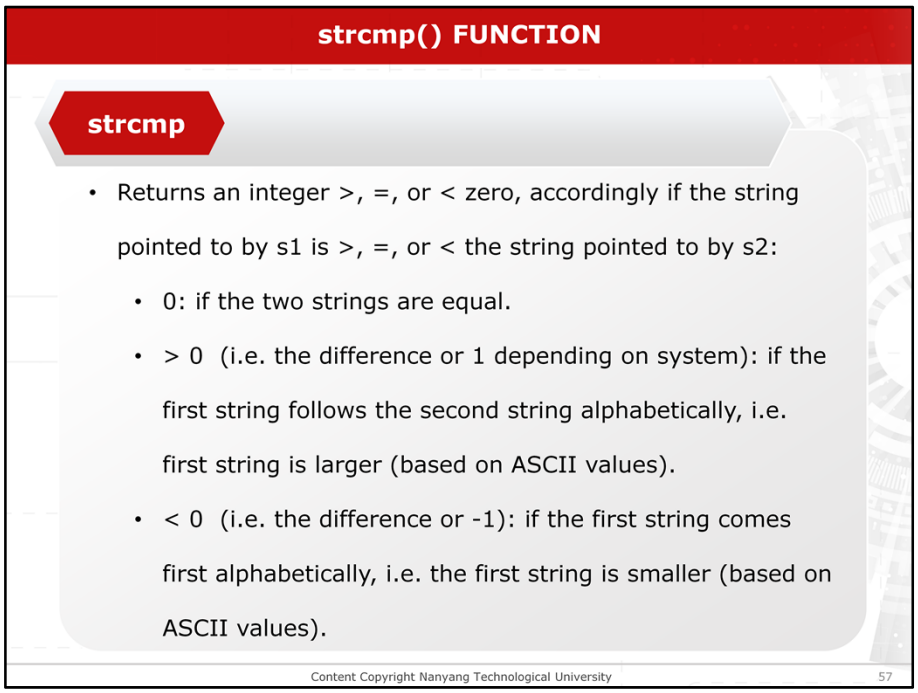
. For example, the lowercase '**a**' has the ASCII code value of 97,



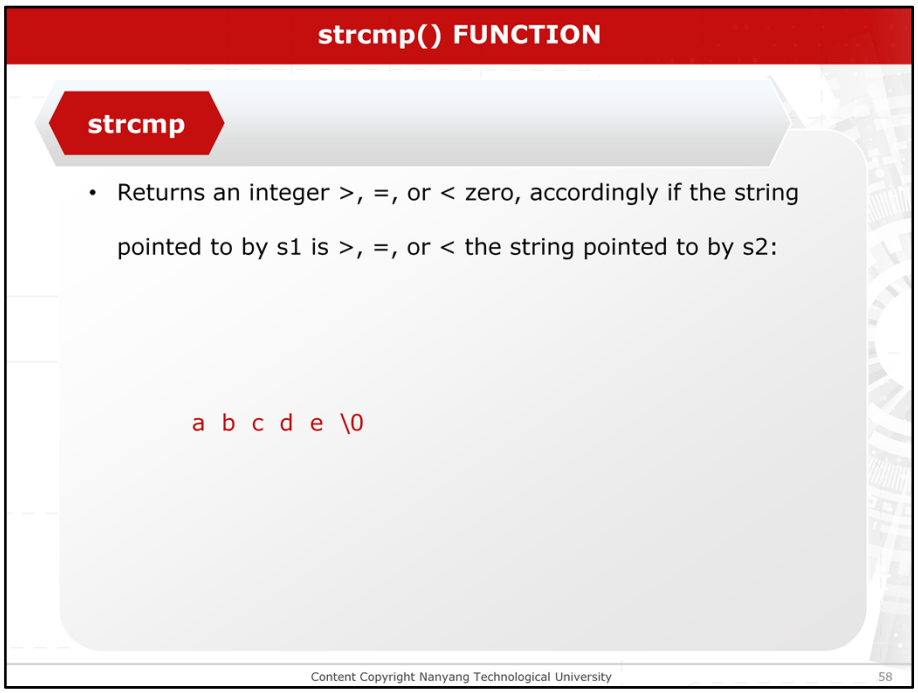
while the uppercase '**A**' has 65



If the initial characters are the same, then the **string compare()** function moves along the string until it finds the first pair of different characters, and returns the comparison result.



It returns an integer greater than, equal to or less than zero, according to whether the string pointed to by **string1** is greater than, equal to or less than the string pointed to by **string2** respectively.



For example, when comparing **a b c d e**

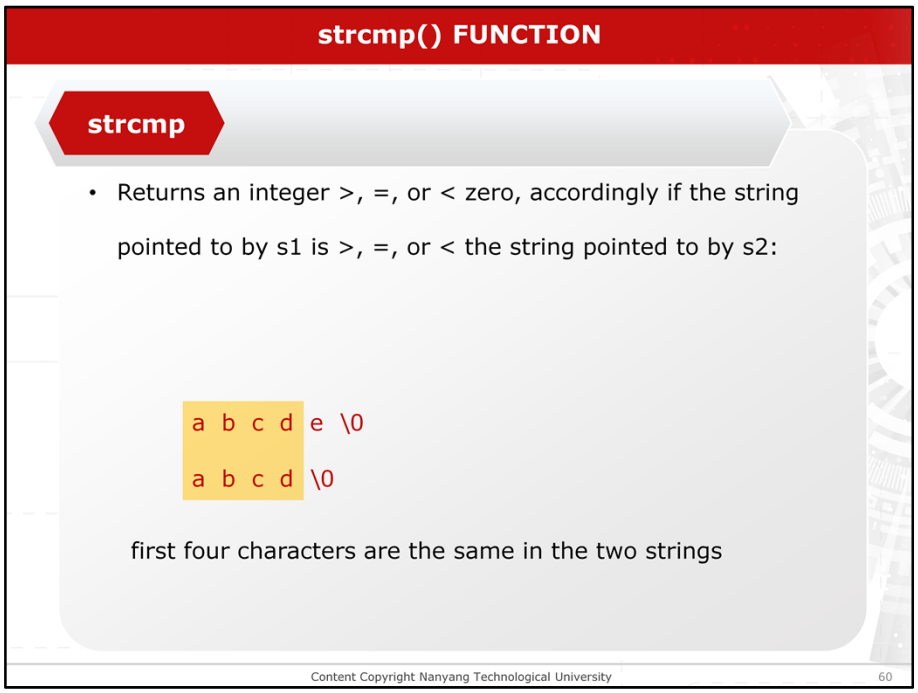
strcmp() FUNCTION

strcmp

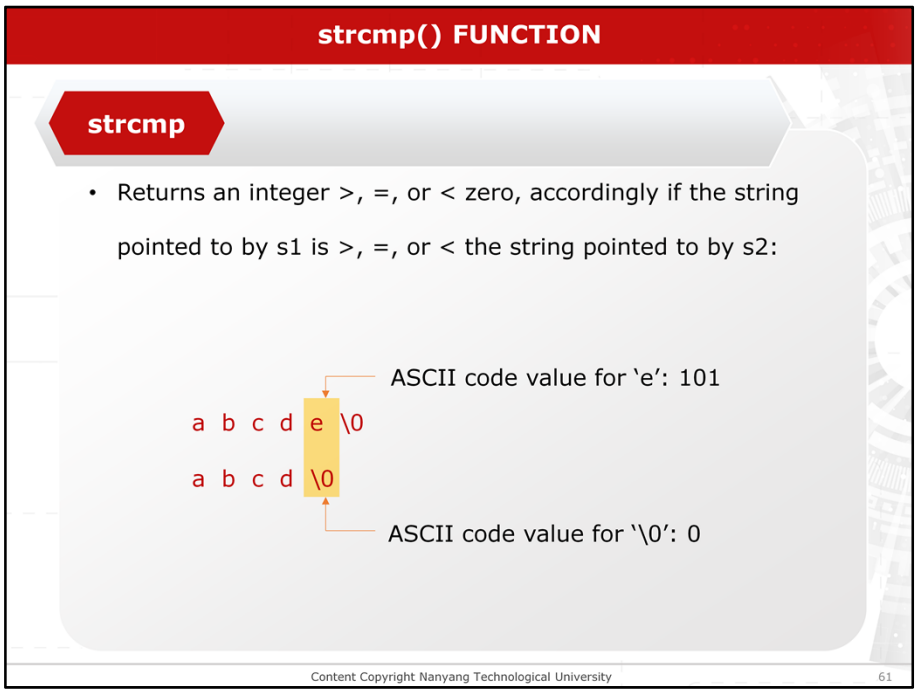
- Returns an integer >, =, or < zero, accordingly if the string pointed to by s1 is >, =, or < the string pointed to by s2:

```
a b c d e \0
a b c d \0
```

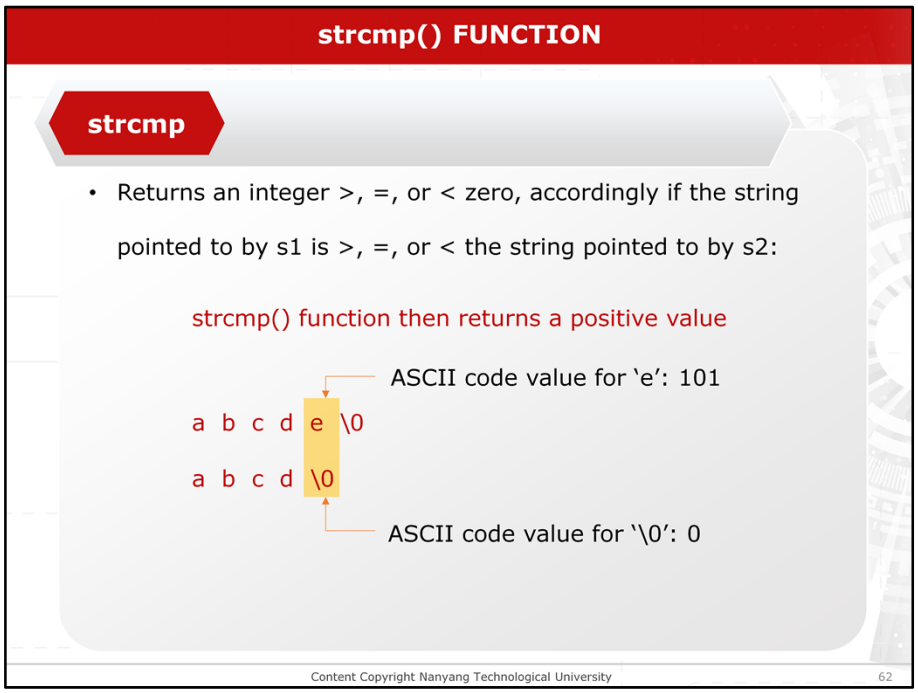

with a **b c d**



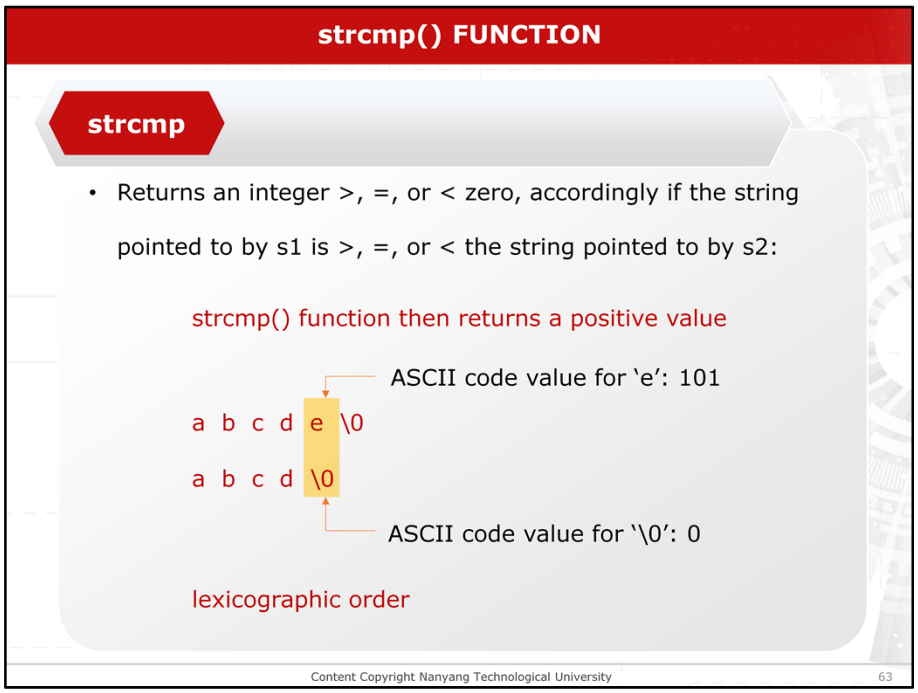
the first four characters are the same in the two strings.



But when it comes to the character '**e**' in the first string, it will be comparing with the terminating null character with ASCII code value of 0 in the second string



The string compare() function then returns a positive value.



This way of ordering strings is called *lexicographic order.*

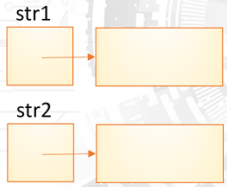
strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```


In the program, it uses the **string compare()** function to compare two strings.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```



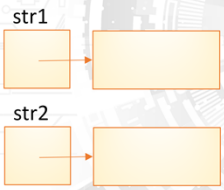
Character array of size 81 are declared for string 1 and string 2

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string:
```



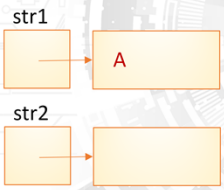
Asking user to input string 1

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string:  A
```



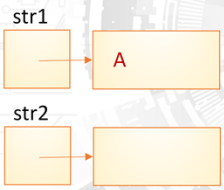
If user enter uppercase A as string 1, get s() function will store string A in string 1

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string:  A
Enter the second string:
```



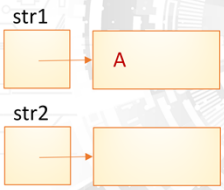
Asking user to input string 2

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string: A
Enter the second string: B
```



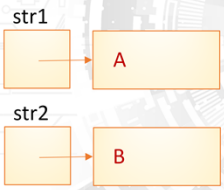
If user enter uppercase B as string 2, get s() function will store string B in string 2

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string:  A
Enter the second string:  B
```



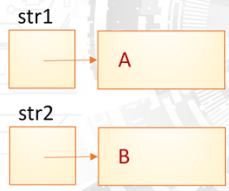
Using the string compare() function to compare string1 and string2

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string: A
Enter the second string: B
```



ASCII code value for 'A': 65

ASCII code value for 'B': 66

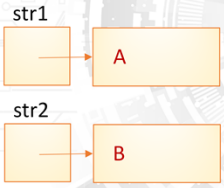
The ASCII code value for uppercase 'A' is 65 and the ASCII code value for uppercase 'B' is 66.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output1

```
String Comparison:
Enter the first string: A
Enter the second string: B
The result of the comparison is -1
```



ASCII code value for 'A': 65

ASCII code value for 'B': 66

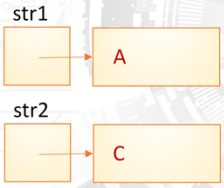
The result will be -1

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output2

```
String Comparison:
Enter the first string: A
Enter the second string: C
The result of the comparison is -1
```



ASCII code value for 'A': 65

ASCII code value for 'C': 67

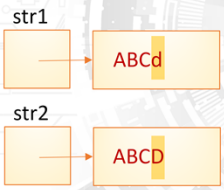
When comparing uppercase "A" to uppercase "C", some systems return –1, others return –2, that is the difference in ASCII code values. However, in many applications, we are only interested to find out whether the two strings are equal or not.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output3

```
String Comparison:
Enter the first string:  ABCd
Enter the second string:  ABCD
```



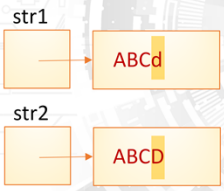
When comparing uppercase "ABC“ with lower case “d” to all uppercase “ABCD", string compare will compare the 4th character since the first 3 characters are the same.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output3

```
String Comparison:
Enter the first string: ABCd
Enter the second string: ABCD
```



ASCII code value for 'd': 100

ASCII code value for 'D': 68

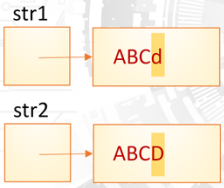
The ASCII code value for lowercase 'd' is 100 and the ASCII code value for uppercase 'D' is 68.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output3

```
String Comparison:
Enter the first string: ABCd
Enter the second string: ABCD
The result of the comparison is 1
```



ASCII code value for 'd': 100

ASCII code value for 'D': 68

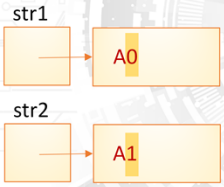
The result will be 1

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output4

```
String Comparison:
Enter the first string: A0
Enter the second string: A1
```



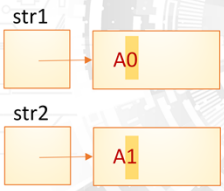
When comparing uppercase "A0“ with uppercase “A1", string compare will compare the 2nd character since the first character is the same.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output4

```
String Comparison:
Enter the first string: A0
Enter the second string: A1
```



ASCII code value for '0': 48

ASCII code value for '1': 49

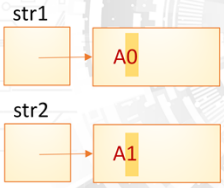
The ASCII code value for 0 is 48 and the ASCII code value for 1 is 49.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Output4

```
String Comparison:
Enter the first string: A0
Enter the second string: A1
The result of the comparison is -1
```



ASCII code value for '0': 48

ASCII code value for '1': 49

The result will be -1

strcmp() FUNCTION – EXAMPLE 1

Output1

String Comparison:
Enter the first string: A
Enter the second string: B
The result of the comparison is -1

Output2

String Comparison:
Enter the first string: A
Enter the second string: C
The result of the comparison is -1

Output3

String Comparison:
Enter the first string: ABCd
Enter the second string: ABCD
The result of the comparison is 1

Output4

String Comparison:
Enter the first string: A0
Enter the second string: A1
The result of the comparison is -1

Here, only 1, 0 or -1 is returned, it could also be the difference in ASCII values depending on the system.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Question: In this program, could we use:

```
if (str1 == str2)
    .....
```

instead?

In this program, if we use the following statement:

if (string1 equivalent string2)

to compare the two strings instead of using the **string compare()** function, will this be ok?

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

Question: In this program, could we use:

```
if (str1 == str2)
    .......
```

instead?

Answer: No

The answer to this question is “not ok”.

strcmp() FUNCTION – EXAMPLE 1

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[81], str2[81];
    int result;

    printf("String Comparison:\n");
    printf("Enter the first string: ");
    gets(str1);
    printf("Enter the second string: ");
    gets(str2);
    result = strcmp(str1, str2);
    printf("The result of the comparison is
                %d\n\n", result);
    return 0;
}
```

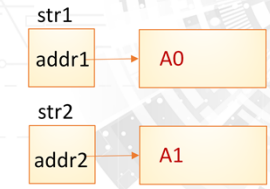
Question: In this program, could we use:

if (str1 == str2)

.....

instead?

Answer: No



Contents of the two string variables str1 and str2 are addresses

It is because the contents of the two strings are addresses. Comparing the two strings in this way only compares the addresses contained in the two string variables. It does not compare the character contents stored in the strings. Therefore, we need to use the **string compare()** function to compare the two strings.

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```


In the program, it uses the **string compare()** function to check whether to exit the loop after reading in an input string. The loop will end if the input string is "**END**".

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
            reverse(line);
            printf("%s\n", line);
            gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```


In the **main()** function of the program, it reads in a string from the standard input, calls the function **reverse()** to reverse the characters in the string

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
          reverse(line);
          printf("%s\n", line);
          gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```


and writes the reversed string to the standard output.

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```


The loop will continue until the user enters the string "**END**".

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```


In the **reverse()** function, it accepts an input string as its parameter.

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
            reverse(line);
            printf("%s\n", line);
            gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

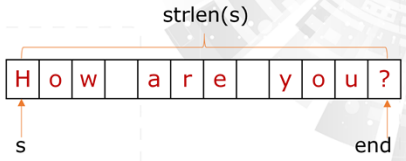
H	o	w		a	r	e		y	o	u	?

Let's see how the reverse function works for the input string "How are you?"

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```



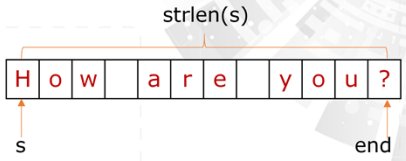
The function then determines the position of the **end** pointer with the statement:

end = s + string length of (s) - 1;

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
          reverse(line);
          printf("%s\n", line);
          gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
        /* 2 ends approaching centre */
        /* swapping operation */
           c = *s;
           *s++ = *end;   /*postfix op*/
                 // i.e. *s = *end; s++;
           *end-- = c;
                 // i.e. *end = c; end--;    }
}
```

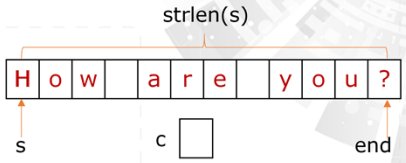


The function then uses a **while** loop to process each character in the string.

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

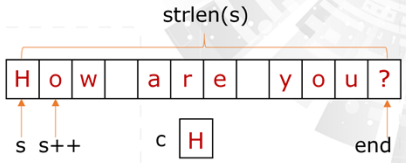


The character at s is stored at variable c

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
          reverse(line);
          printf("%s\n", line);
          gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
          c = *s;
          *s++ = *end;   /*postfix op*/
                // i.e. *s = *end; s++;
          *end-- = c;
                // i.e. *end = c; end--;    }
}
```

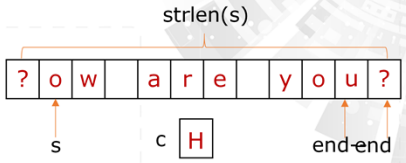


The character at end replace the character at s

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
          reverse(line);
          printf("%s\n", line);
          gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;   /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

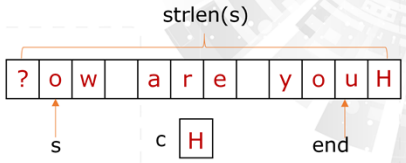


The character at end is replaced by the character stored in variable c. Thus, the characters are swapped

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
          reverse(line);
          printf("%s\n", line);
          gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
          c = *s;
          *s++ = *end;   /*postfix op*/
               // i.e. *s = *end; s++;
          *end-- = c;
               // i.e. *end = c; end--;    }
}
```

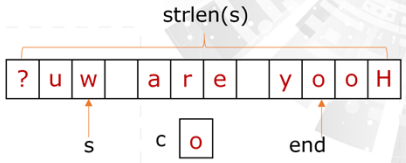


In the loop, it swaps the characters from both ends of the string, and then moves the string pointer **s** and the end string pointer **end** towards the center of the string with **s++** and **end--**.

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
“END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, “END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

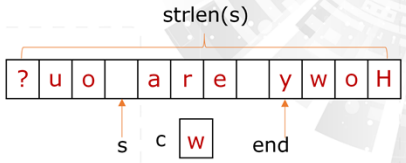


[audio pause]

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;   /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

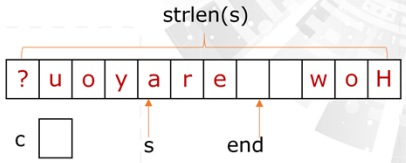


[audio pause]

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

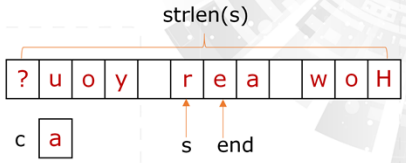


[audio pause]

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

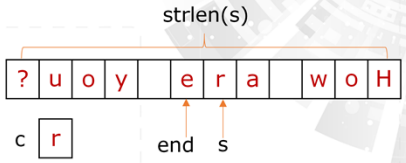


[audio pause]

strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
               reverse(line);
               printf("%s\n", line);
               gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;    /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```

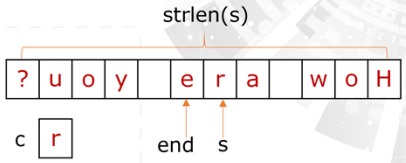


After the operation, the input string will be reversed.

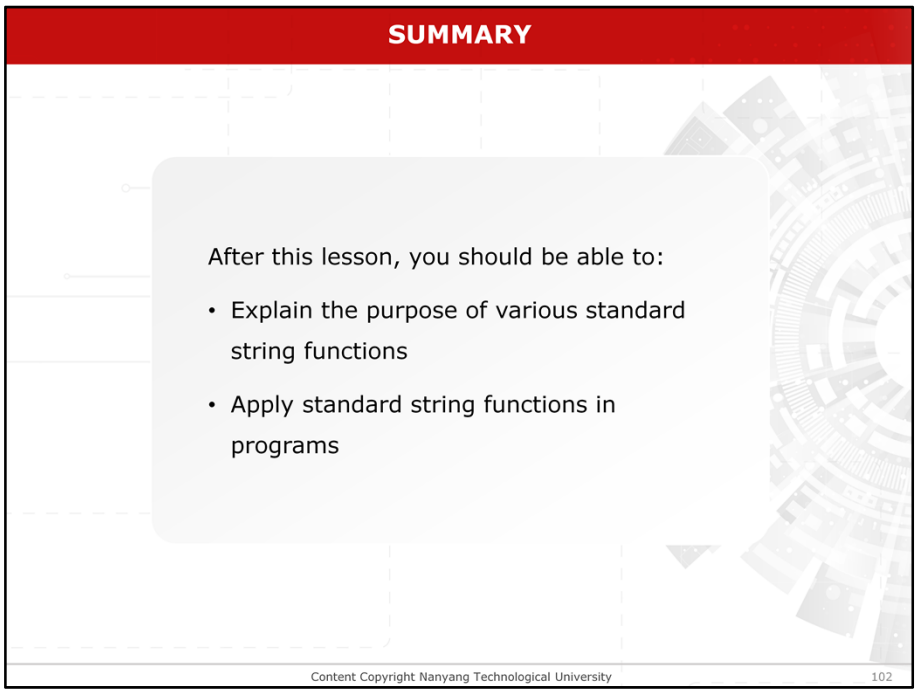
strcmp() FUNCTION – EXAMPLE 2

```
/* Read a few lines from standard
input & write each line to standard
output with the characters reversed.
The input terminates with the line
"END"*/

#include  <stdio.h>
#include  <string.h>
void reverse(char *s);
int main()
{
     char line[132];
     gets(line);
     while (strcmp(line, "END") !=0) {
          reverse(line);
          printf("%s\n", line);
          gets(line);
     }
}
```

```
void reverse(char *s)
{
     char c, *end;
     end = s + strlen(s) - 1;
     while (s < end) {
       /* 2 ends approaching centre */
       /* swapping operation */
            c = *s;
            *s++ = *end;   /*postfix op*/
                  // i.e. *s = *end; s++;
            *end-- = c;
                  // i.e. *end = c; end--;    }
}
```



Note that the **reverse()** function illustrates a typical example on string processing.



After watching this video lesson, you should be able to do the points listed.