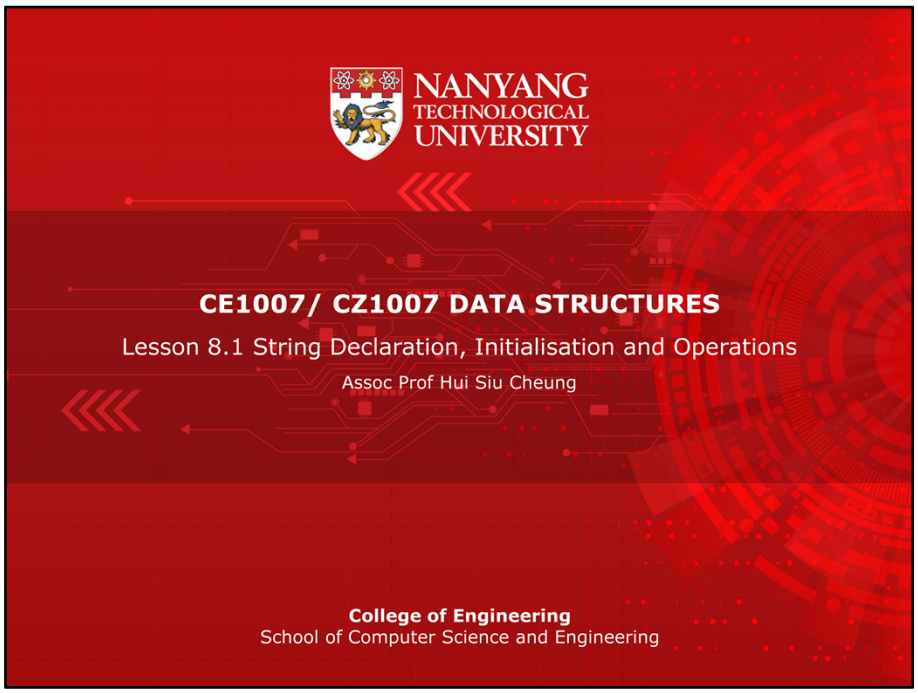
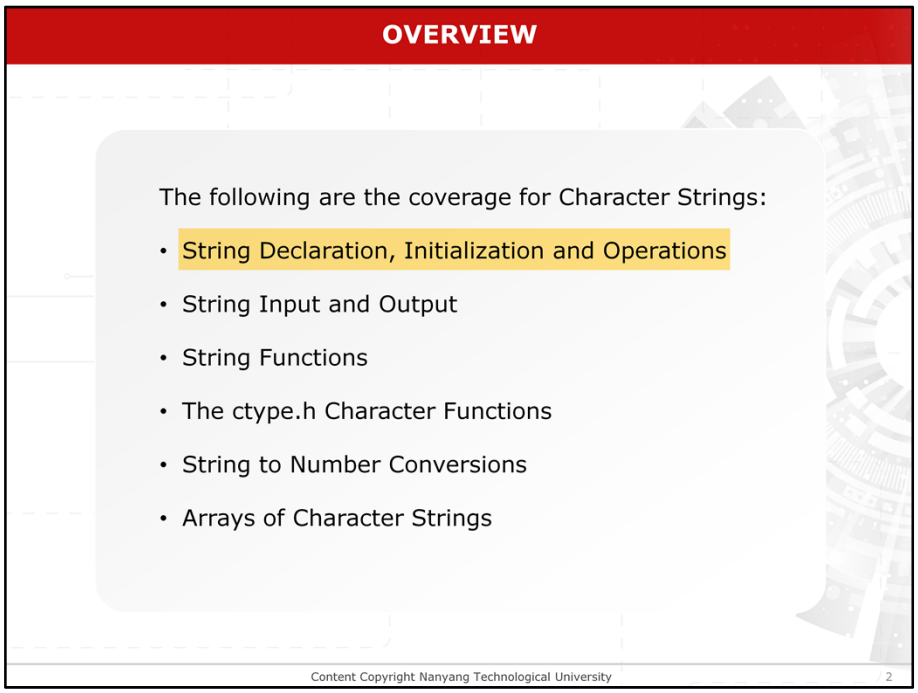
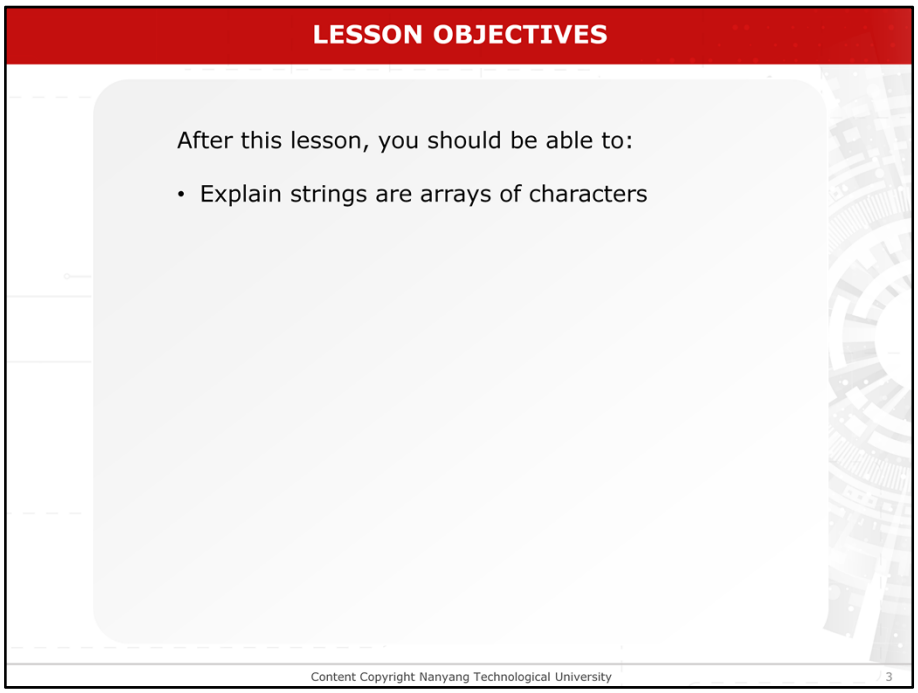


Lesson 8.1:

In addition to handling numerical data, programs are also required to deal with alphabetic data. Strings are arrays of characters. C libraries provide a number of functions for performing operations on strings. In this chapter, string constants and string variables are first introduced. The different commonly used string functions from C libraries are discussed.

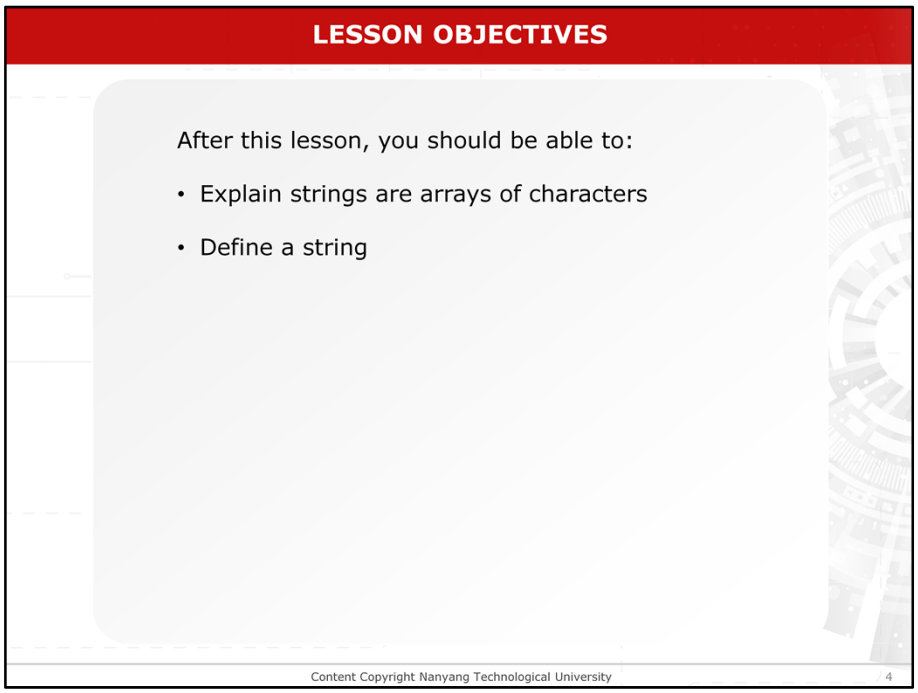


There are 6 main sections to cover for Character Strings as shown. This video lesson focuses on the first part of String Declaration, Initialization and Operations.

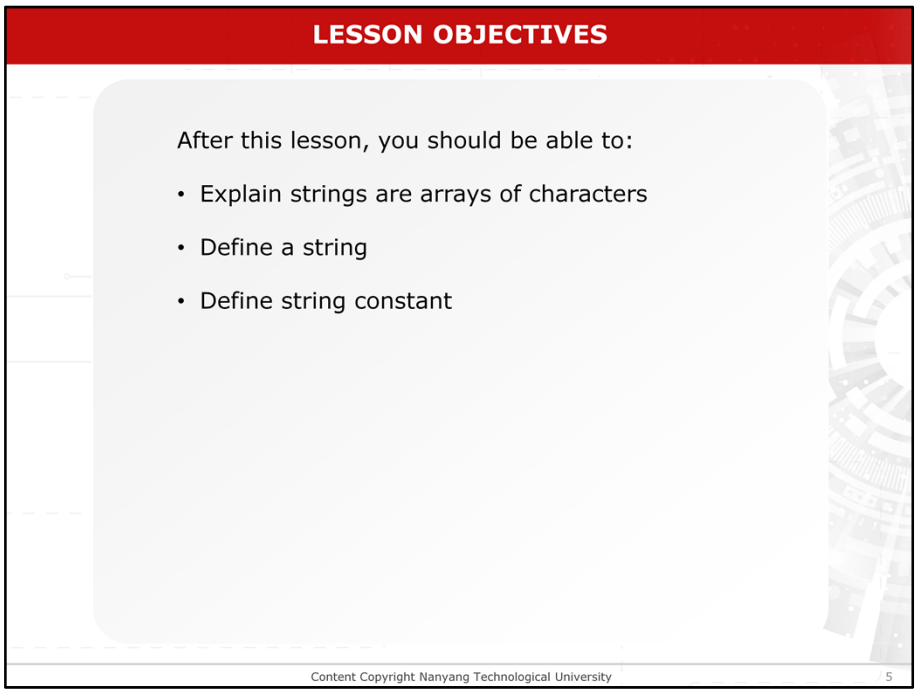


After this lesson, you should be able to:

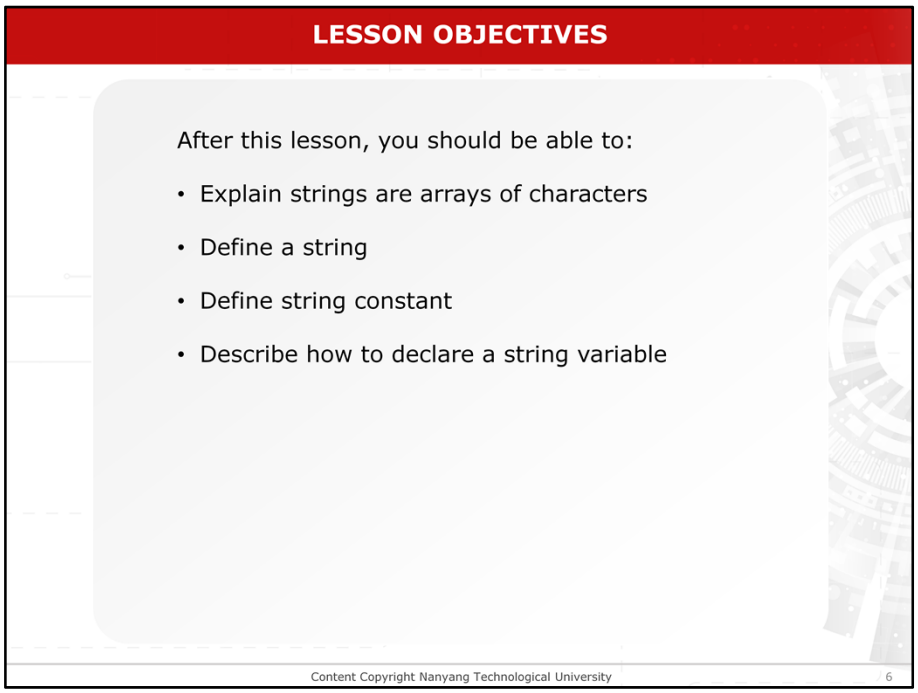
Explain strings are arrays of characters



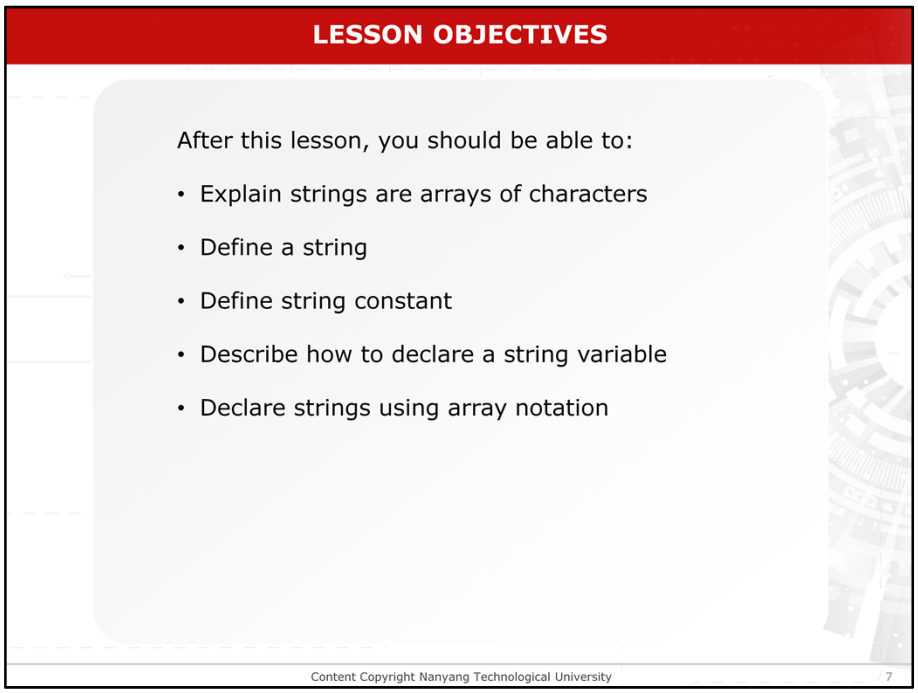
Define a string



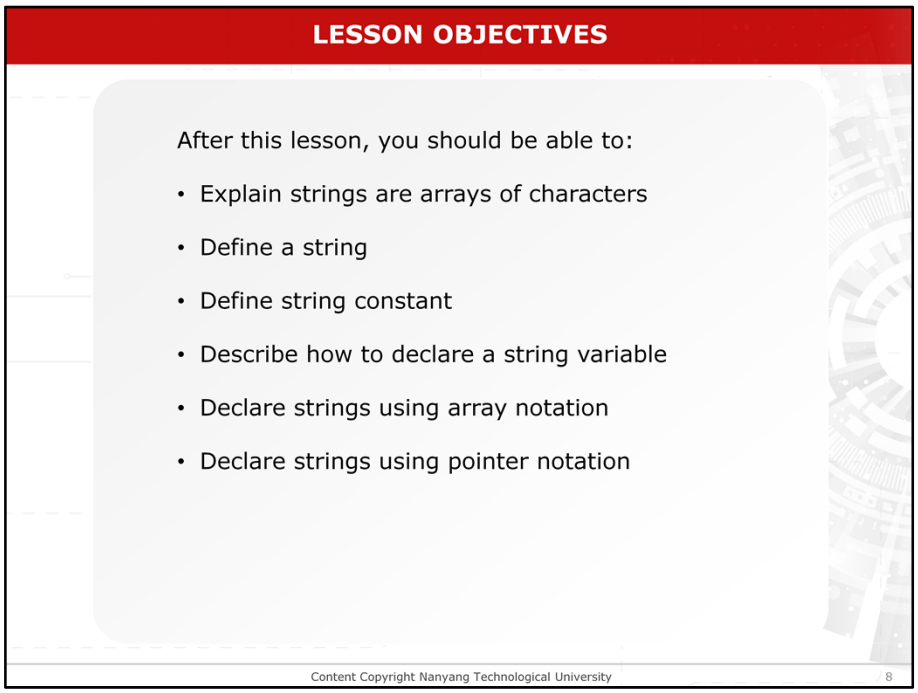
Define string constant



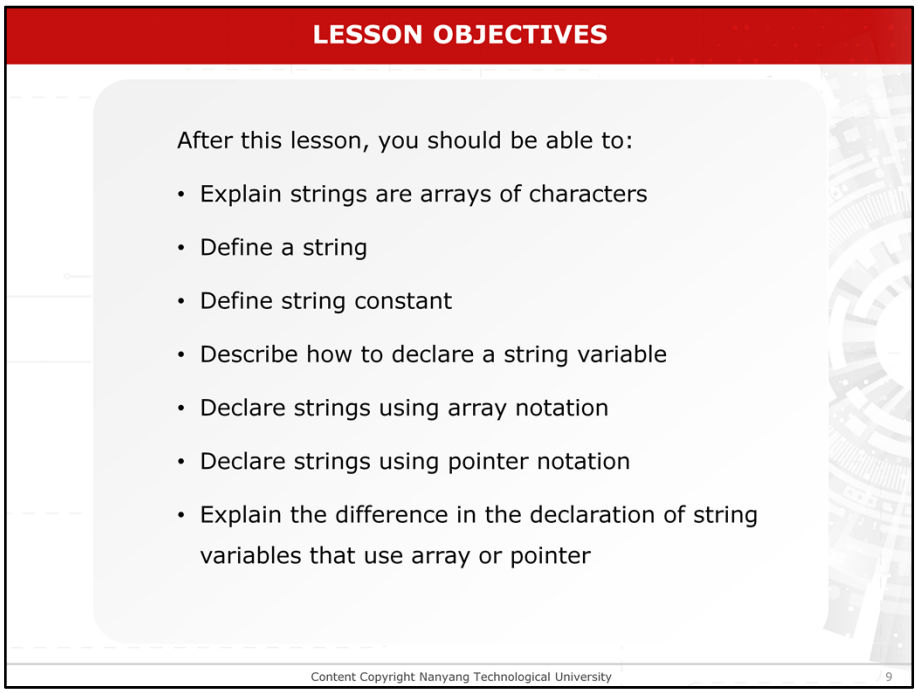
Describe how to declare a string variable



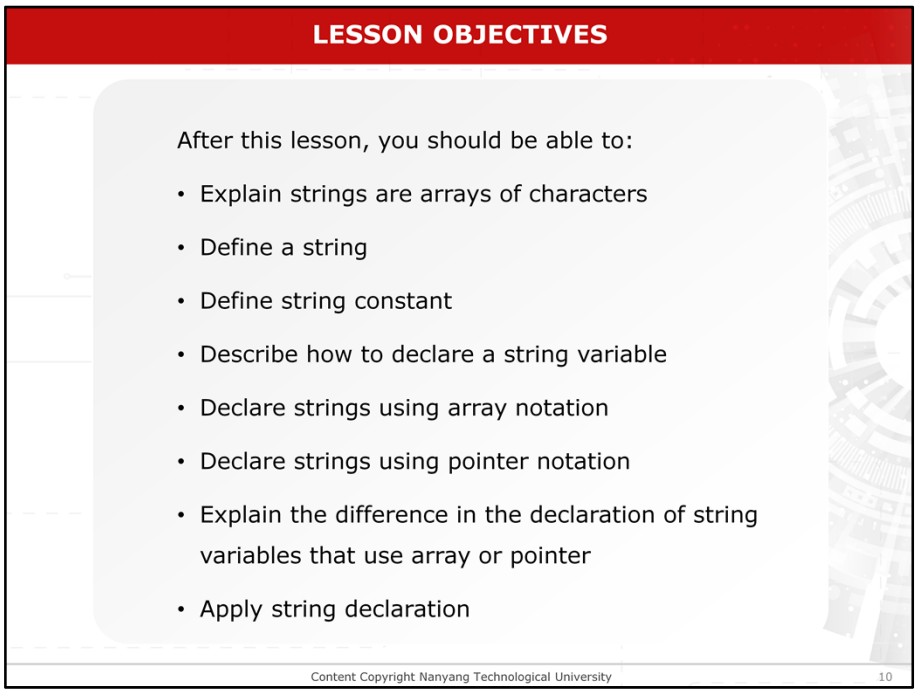
Declare strings using the array notation



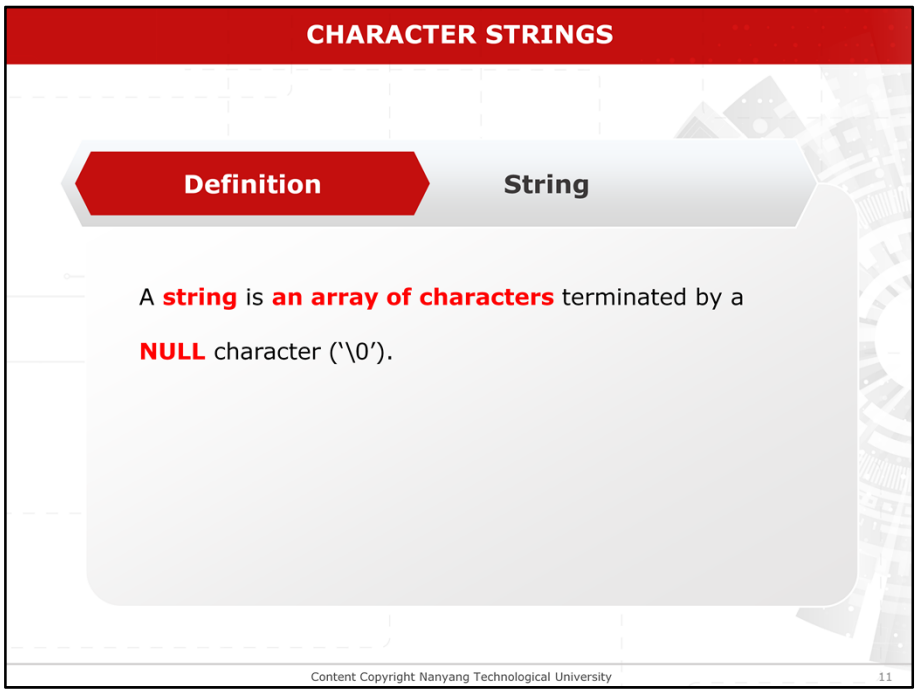
Declare strings using pointer notation



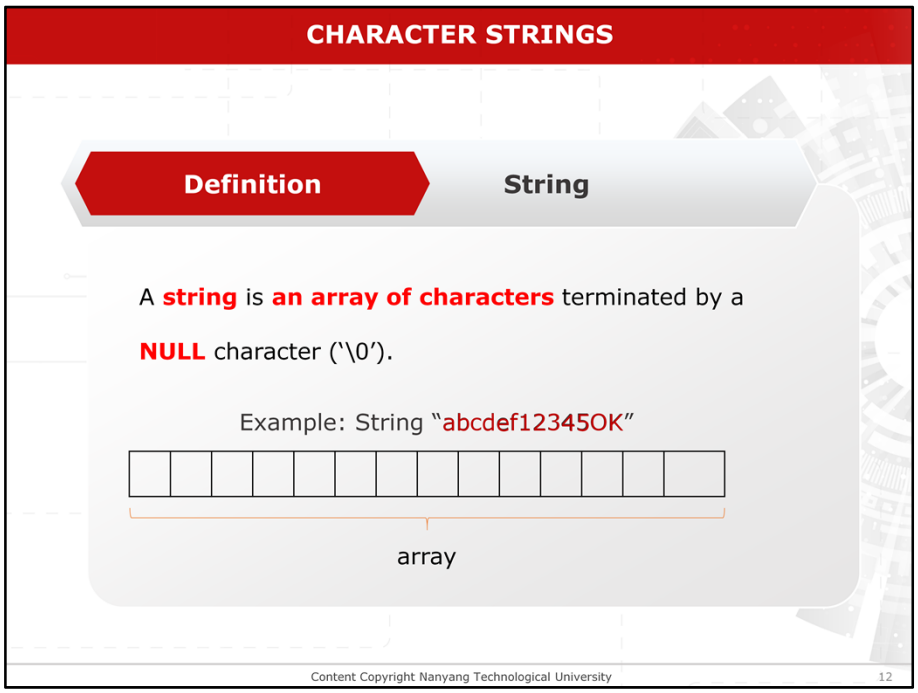
Explain the difference in the declaration of string variables that use array or pointer



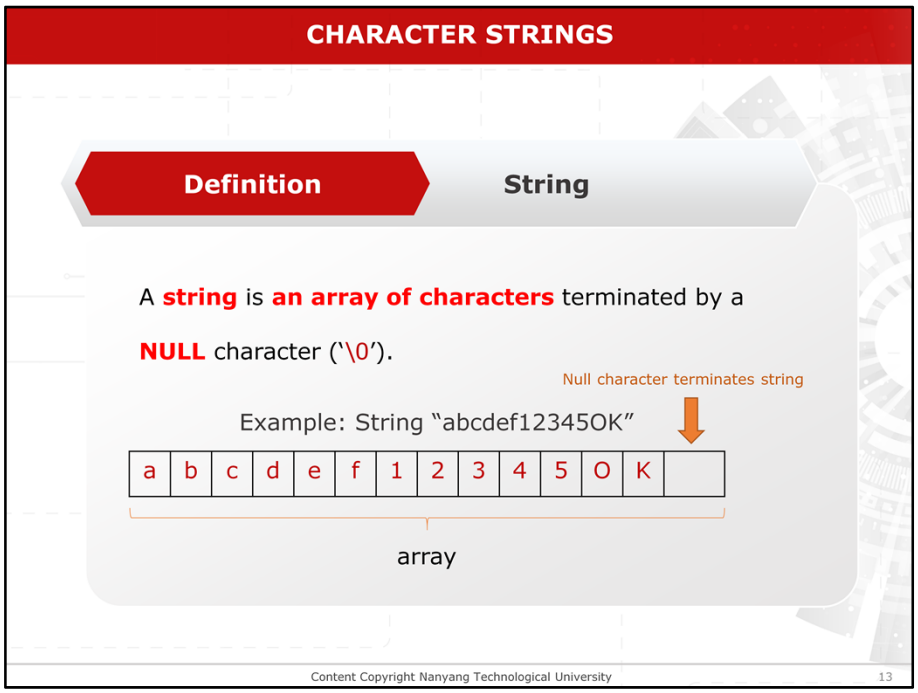
Apply string declaration



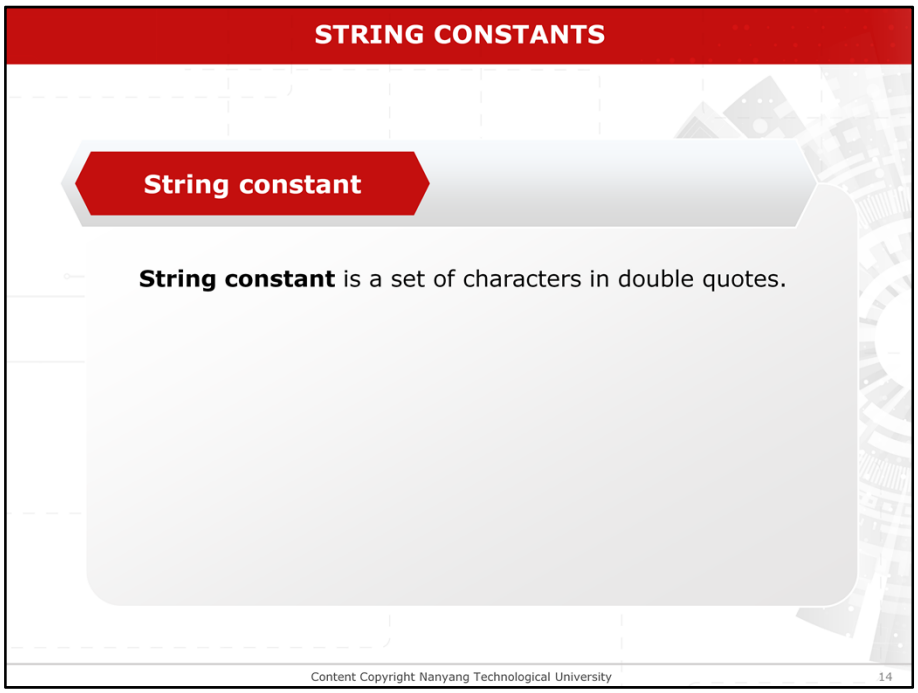
A string is an array of characters terminated by a null character.



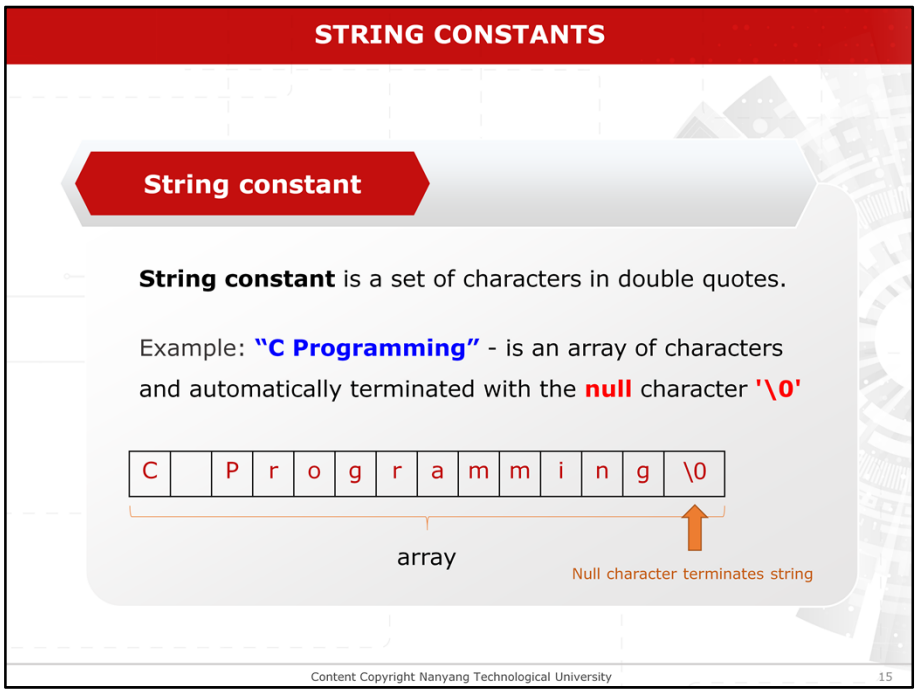
For example, the string double quote a b c d e f 1 2 3 4 5 O K double quote is actually stores as characters in array



terminated by a **NULL** character.

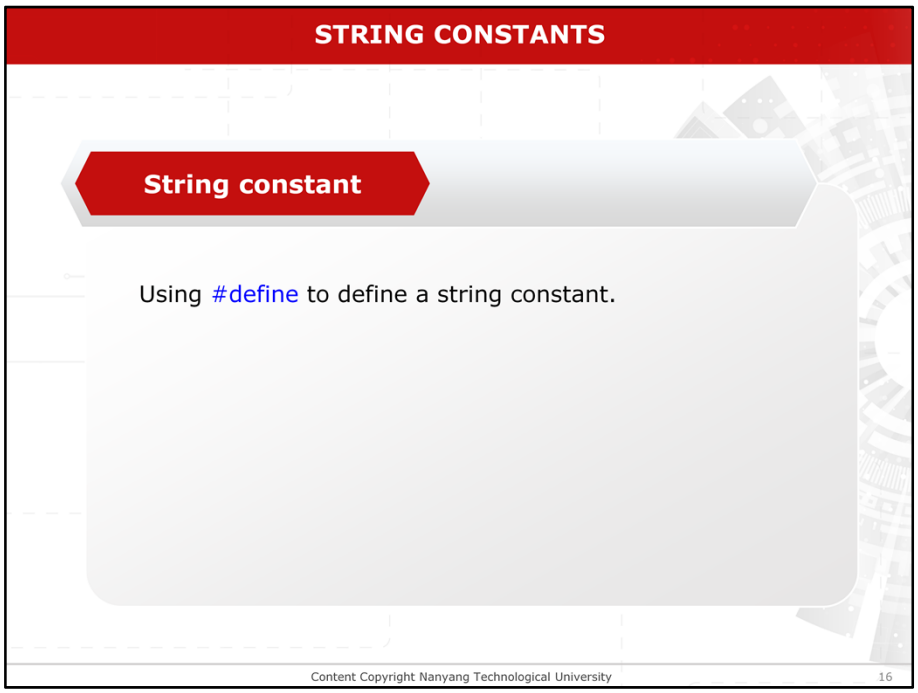


String constant is a set of characters in double quotes.

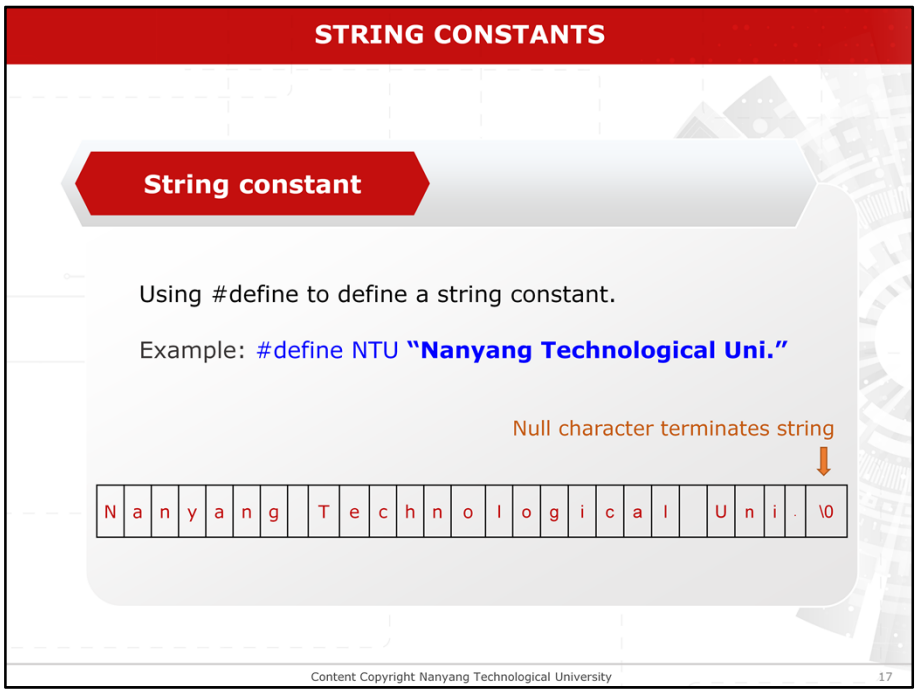


For example, the string **double quote C Programming double quote** is an array of characters and automatically terminated with the **null** character.

When the compiler encounters a string constant, it allocates space for each individual character of the string and adds a terminating null character at the end of the string.

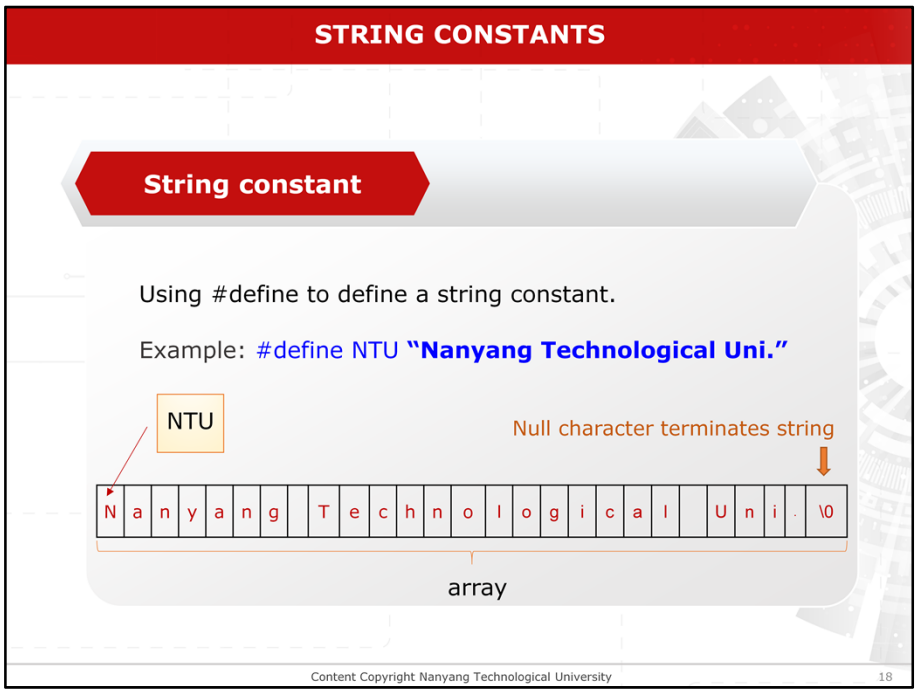


We can use **pound define** to define string constants.

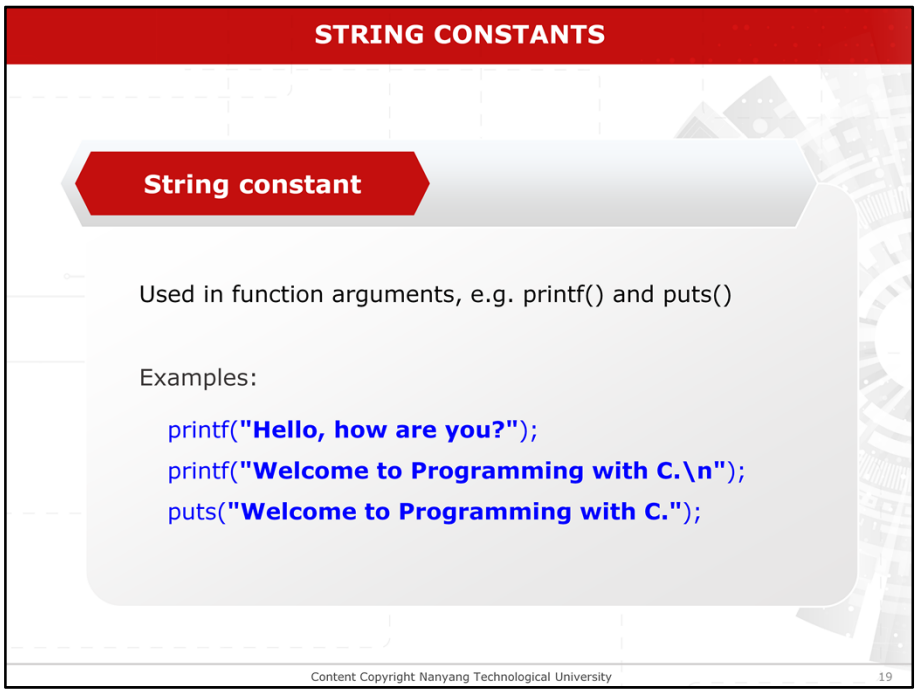


For example,

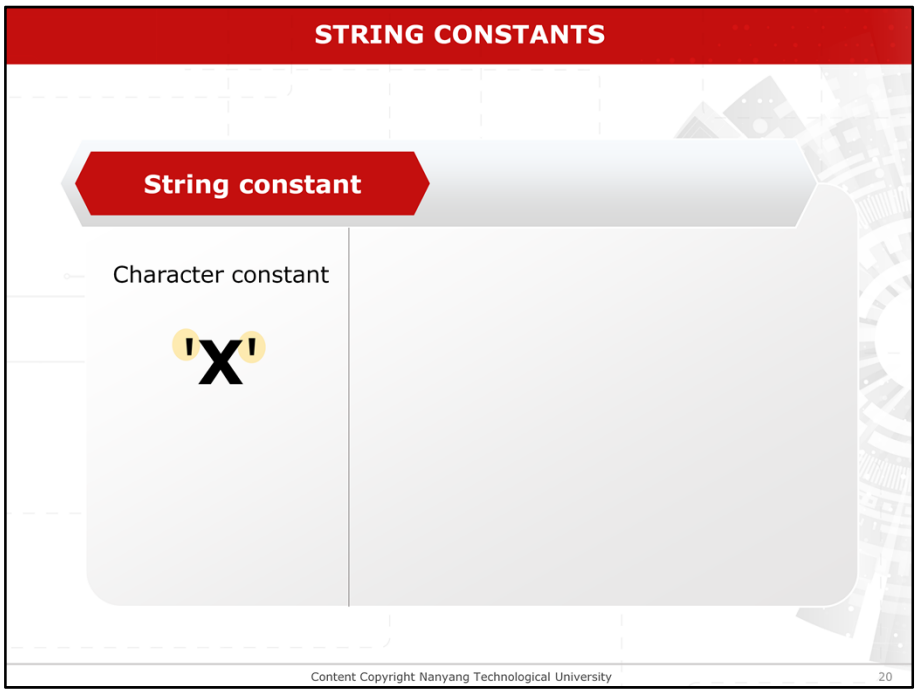
pound define NTU double quote Nanyang Technological University double quote



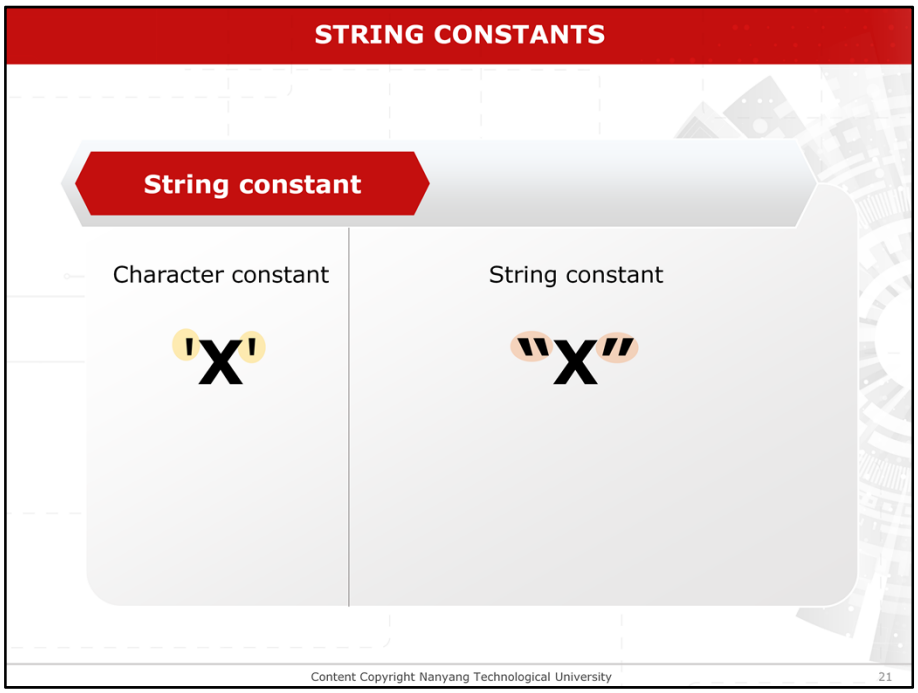
A pointer pointing to its first character is returned.
A string constant is essentially a pointer to the first character of an array of characters.



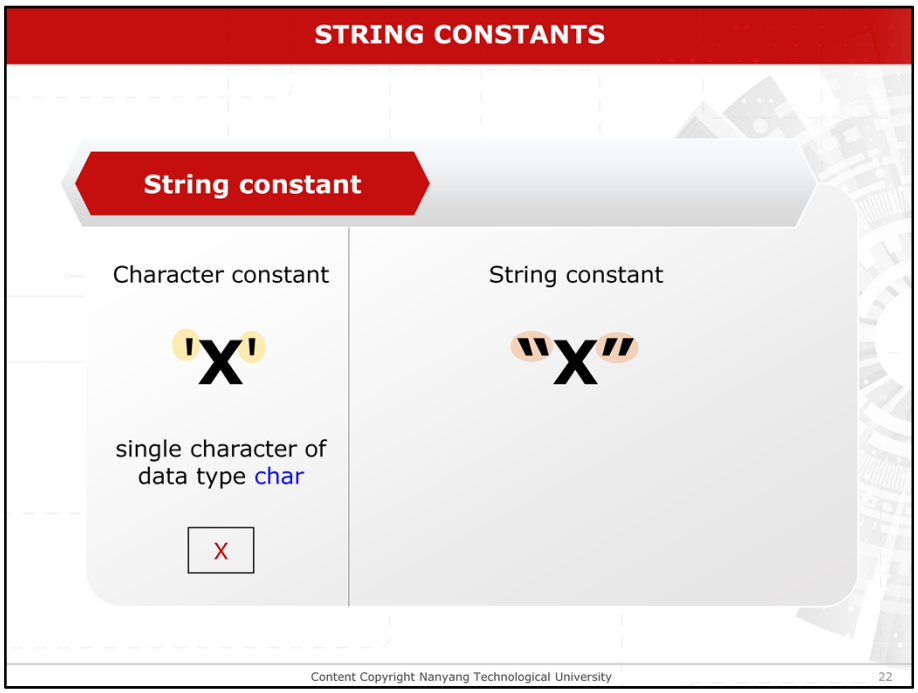
String constants can also be used as function arguments for **print f** and **put s** functions:



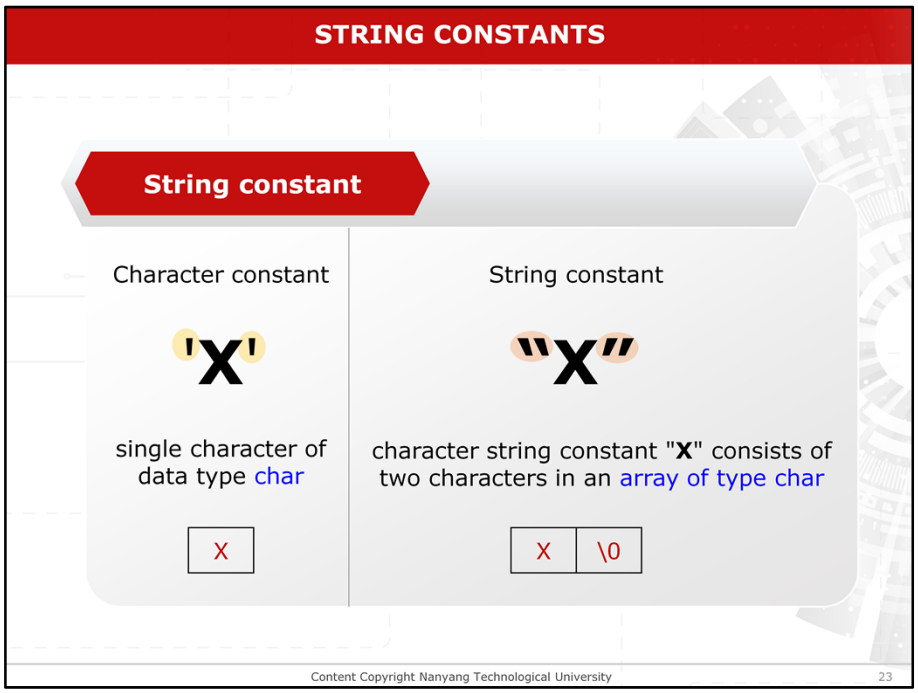
It is important to distinguish the difference between the character constant single quote X single quote



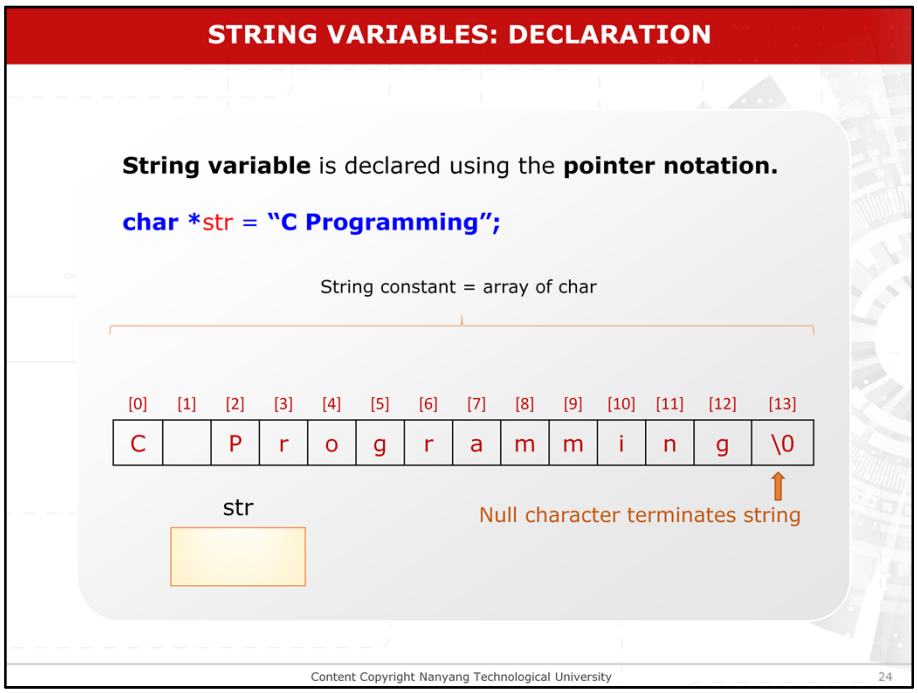
and a string constant double quote X double quote



The character constant single quote X single quote consists of a single character of data type **char**acter

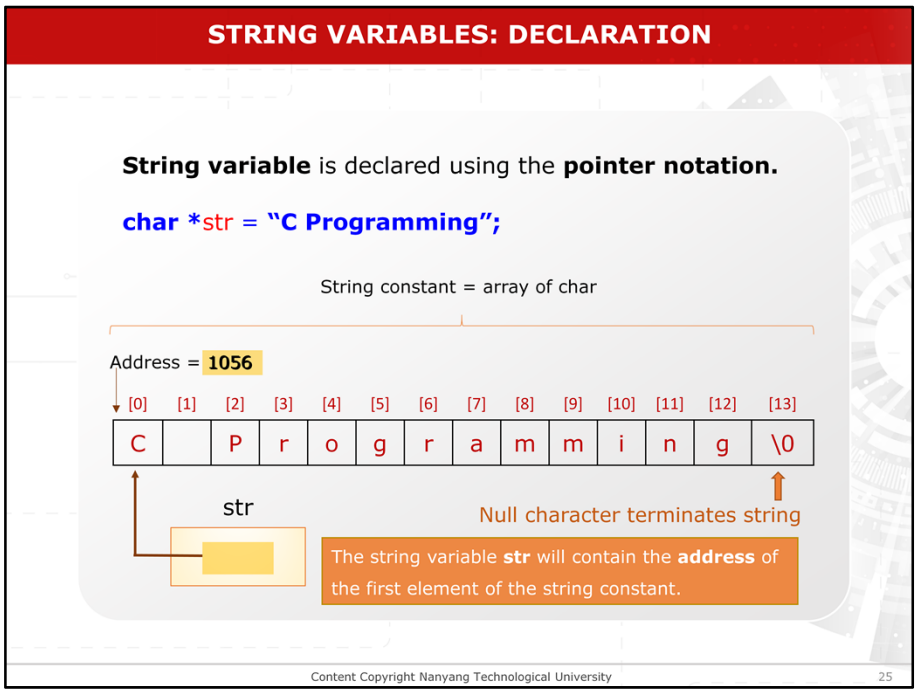


while the character string constant double quote X double quote consists of two characters, that is, the character X and the null character and is an array of type **character**.

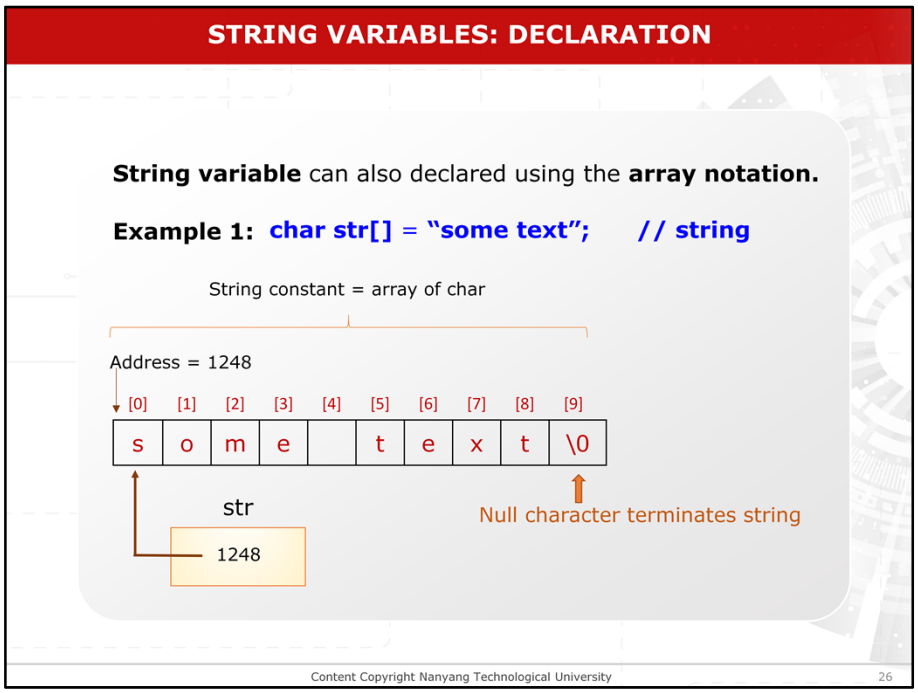


Typically, we can declare a string variable by assigning the pointer of the string constant to a **pointer variable**:

character asterisk string = double quote C Programming double quote;

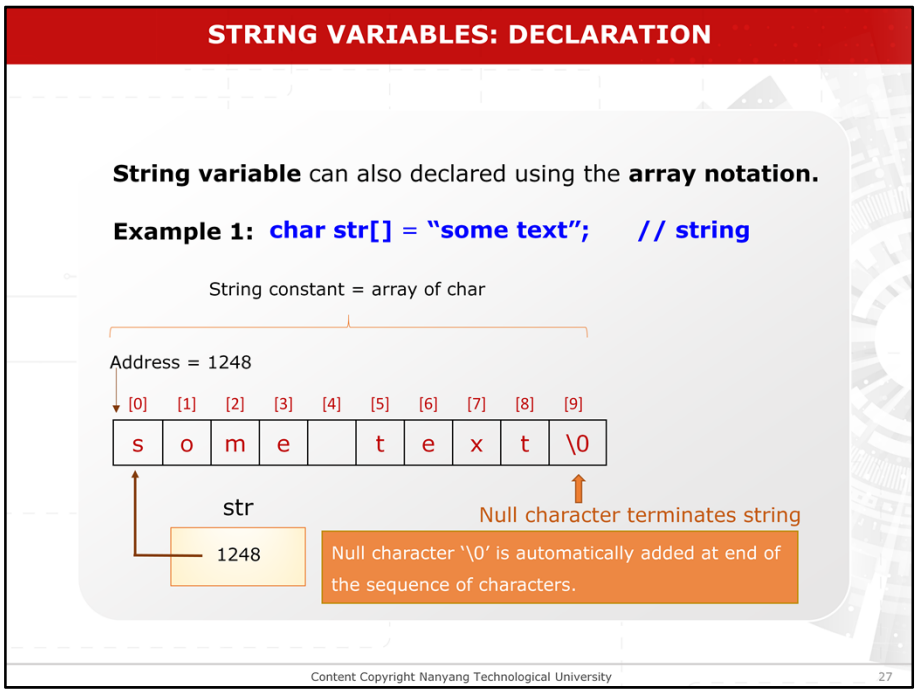


The string variable **string** will then contain the **address** of the first element of the string constant.

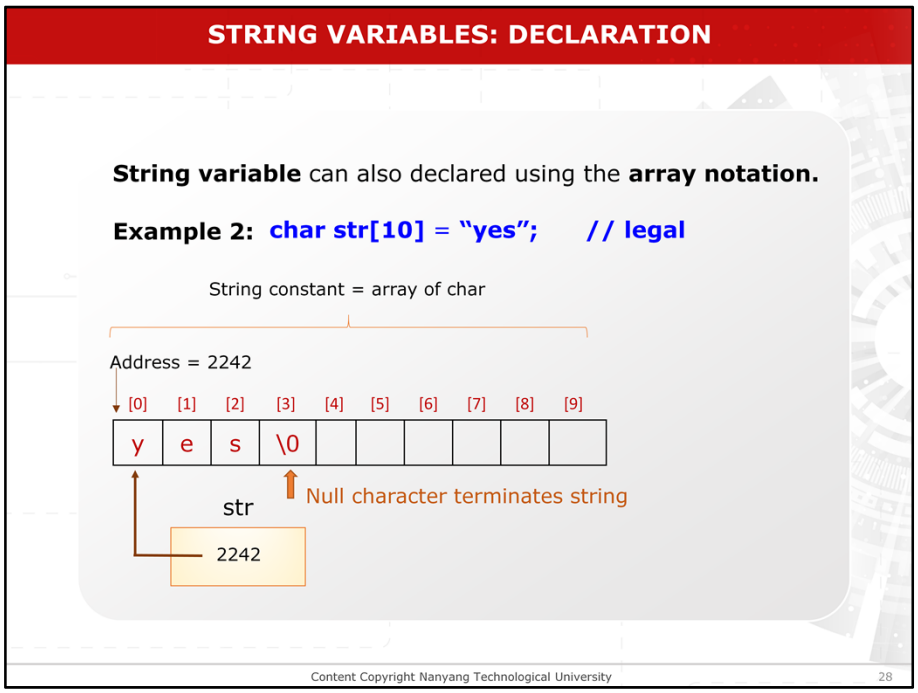


A string is an array of characters. Therefore, we can also define strings using the **array notation**, in addition to the pointer notation. To declare strings using array notation, we can use the following examples:

character array string = double quote some text double quote;



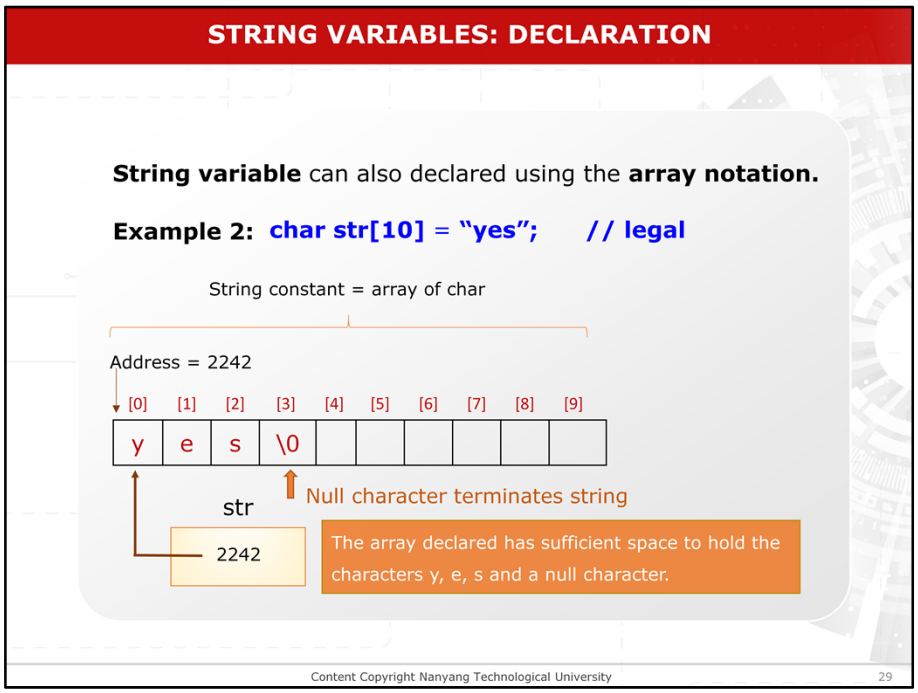
A null character is automatically added at end of the sequence of characters to make variable string a string.



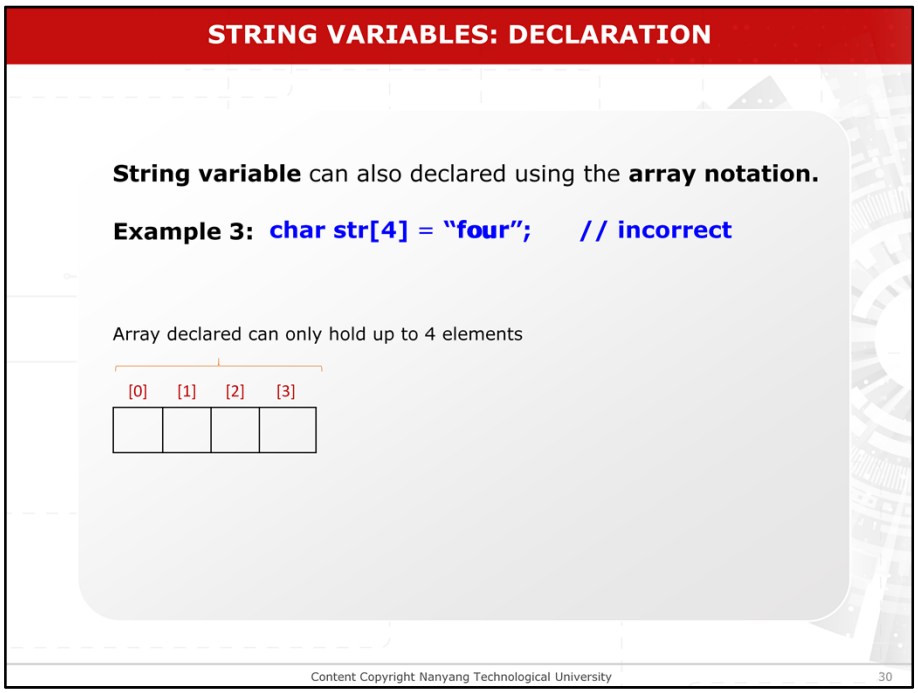
Example 2

character array of size 10, string = double quote yes double quote;

The array **string** is allocated space to hold up to 10 elements.



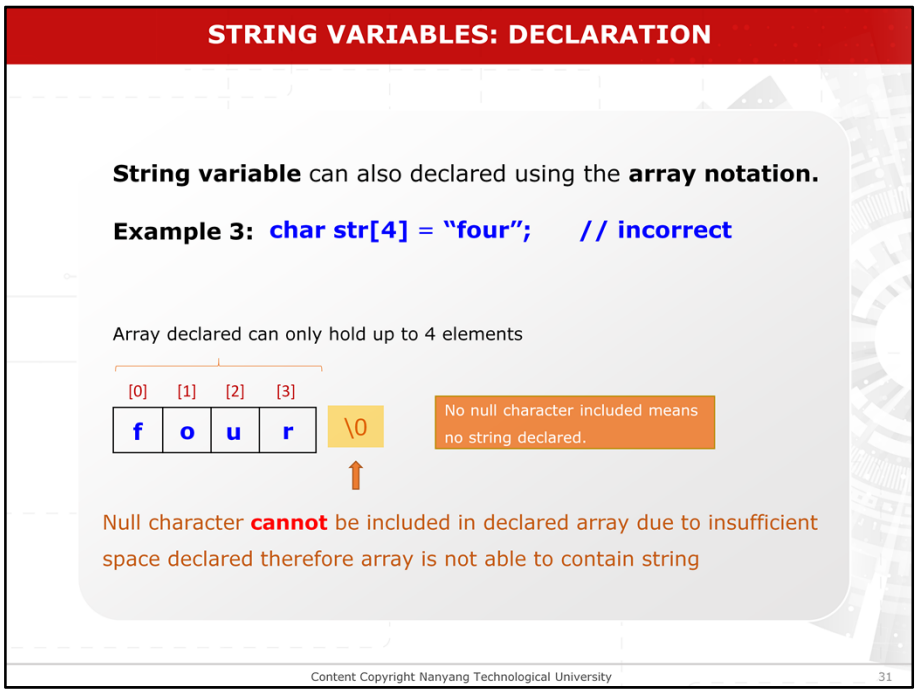
The array has sufficient space to hold the characters y, e, s and a null character. So this is a legal declaration.



Example 3

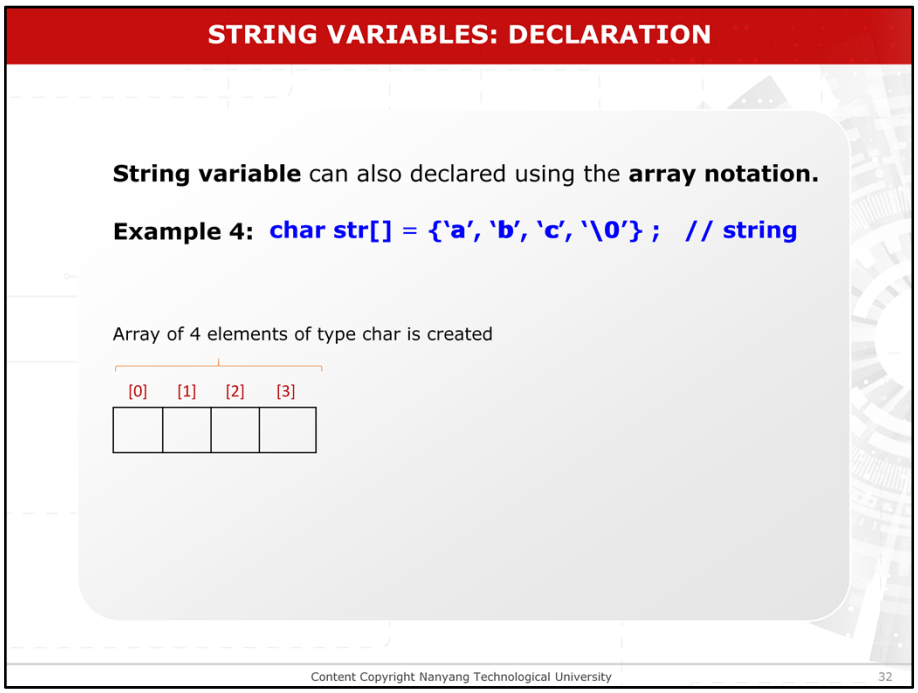
character array of size 4, string = double quote four double quote;

The array **string** is allocated space to hold up to only 4 elements.



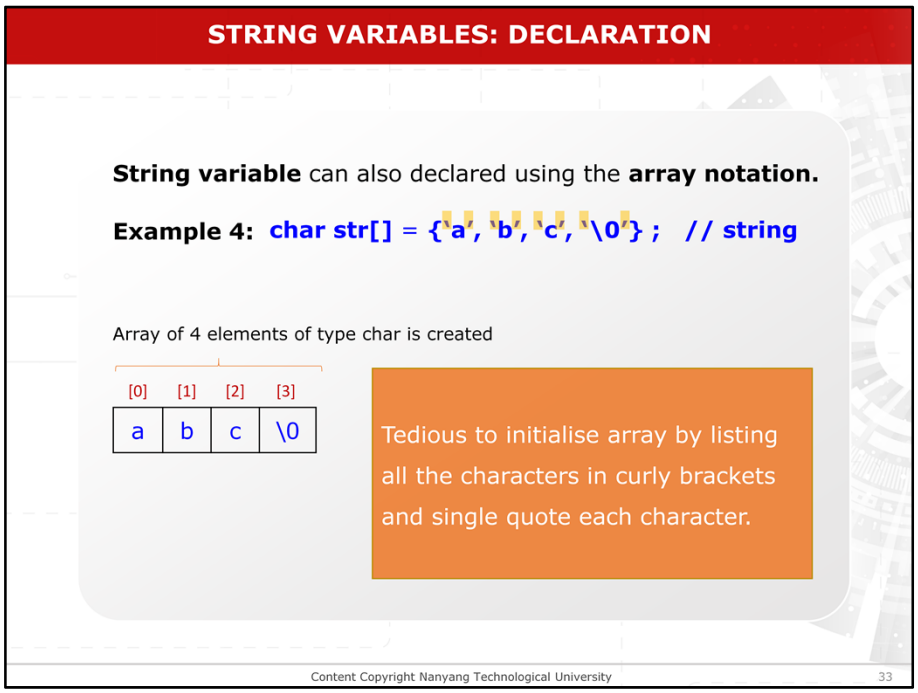
However, the null character can**not** be included in declared array due to insufficient space declared. Therefore, array is not able to contain string.

The **string** array only allocates 4 elements to hold its data, which is not enough as it needs an additional element to hold the null character that is automatically added at the end of the sequence of characters.

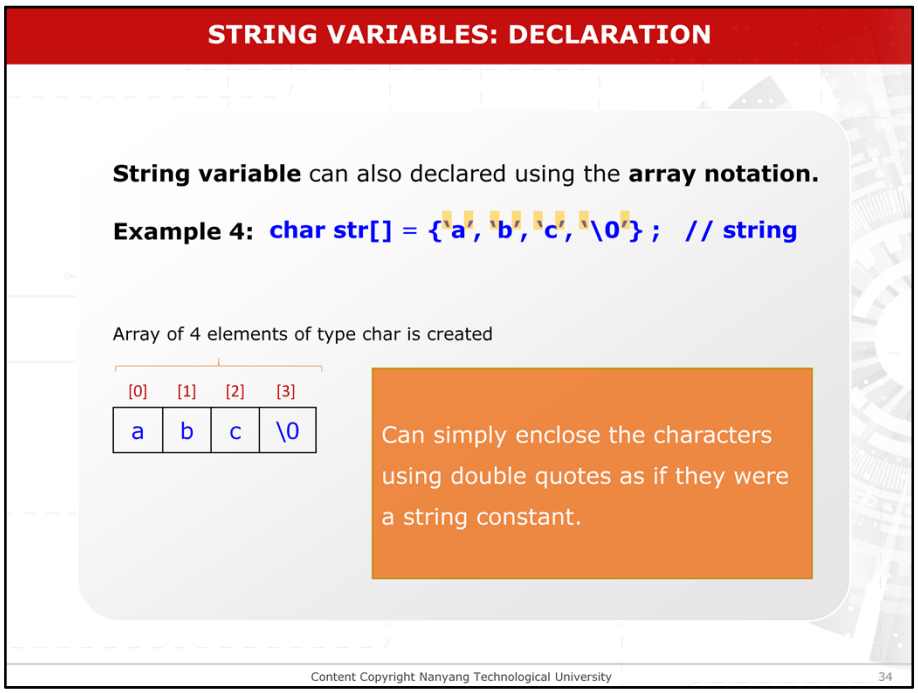


We can also create strings as follows:

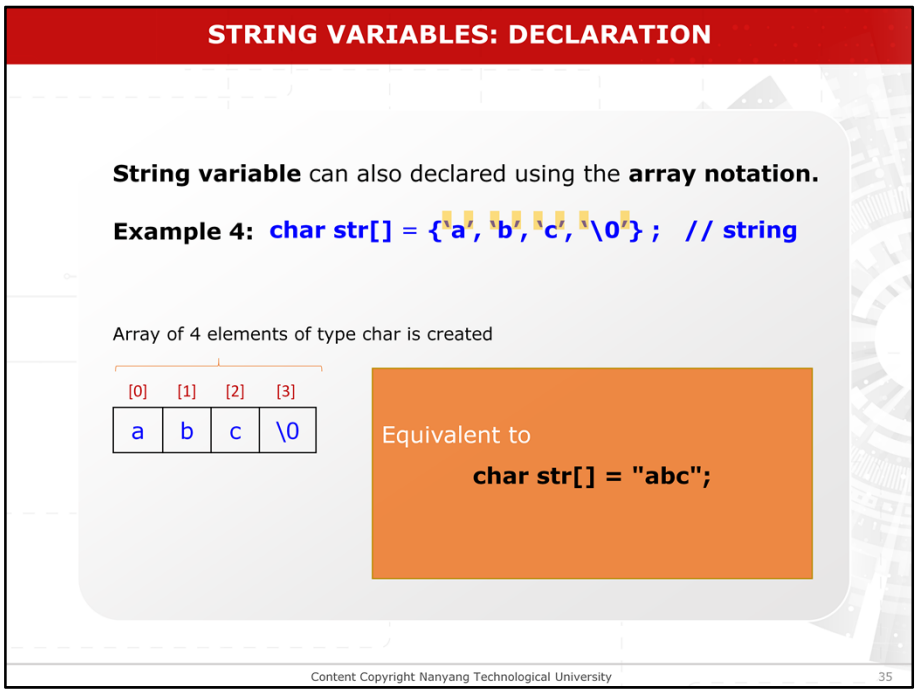
character array string = open curly bracket, single quote a single quote, single quote b single quote, single quote c single quote and single quote null character single quote, close curly bracket;



However, it is tedious to initialize array by listing all the characters in curly brackets and single quote each character



We can simply enclose the characters using double quotes as if they were a string constant.



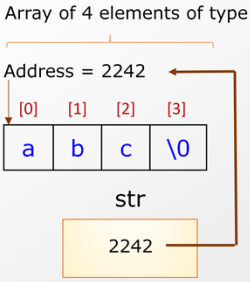
This is equivalent to

character array string = double quote a b c double quote;

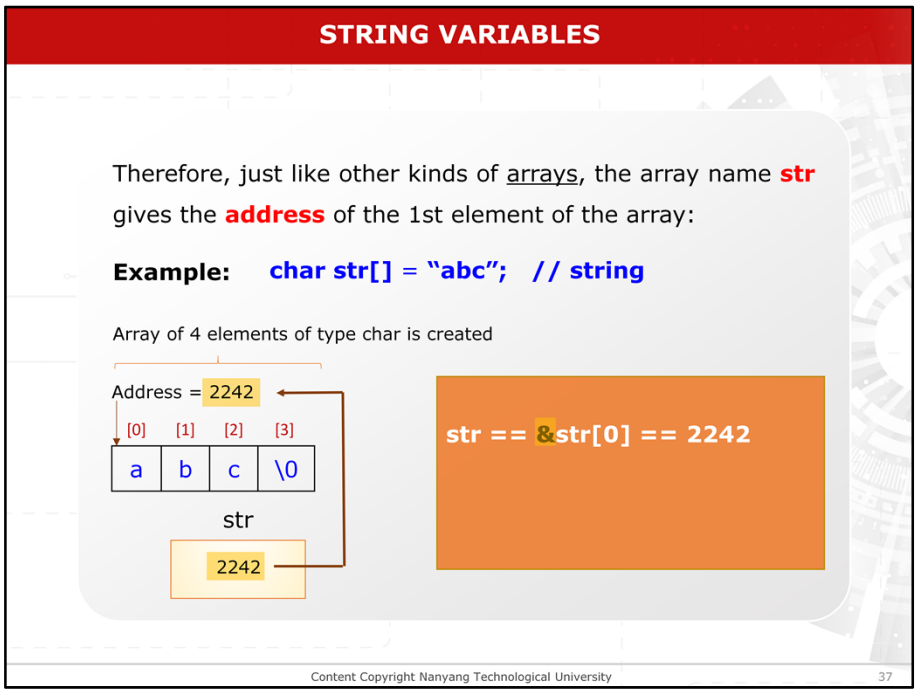
STRING VARIABLES

Therefore, just like other kinds of arrays, the array name **str** gives the **address** of the 1st element of the array.

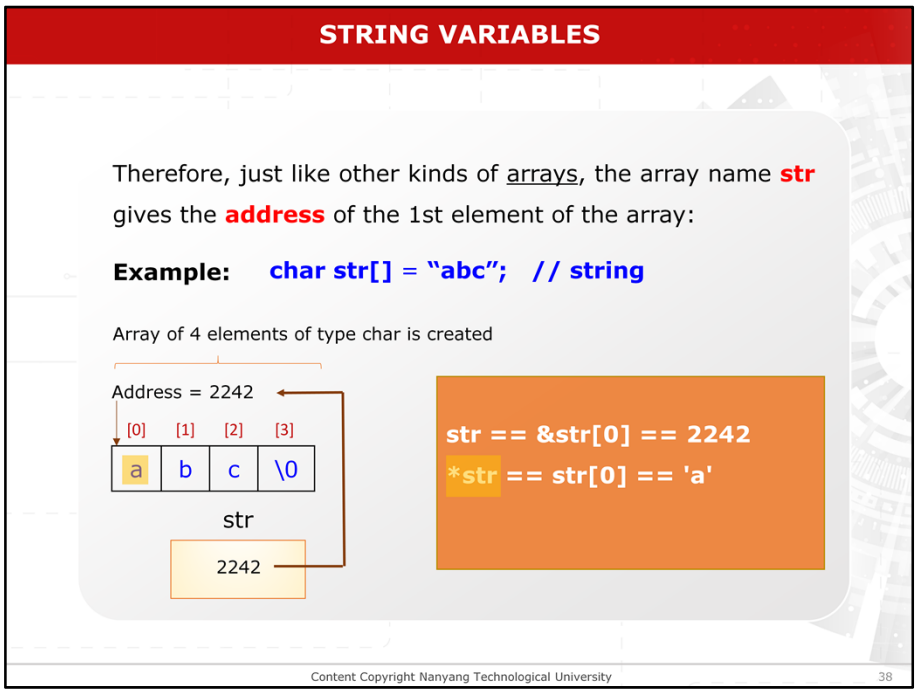
Example: **char str[] = "abc"; // string**



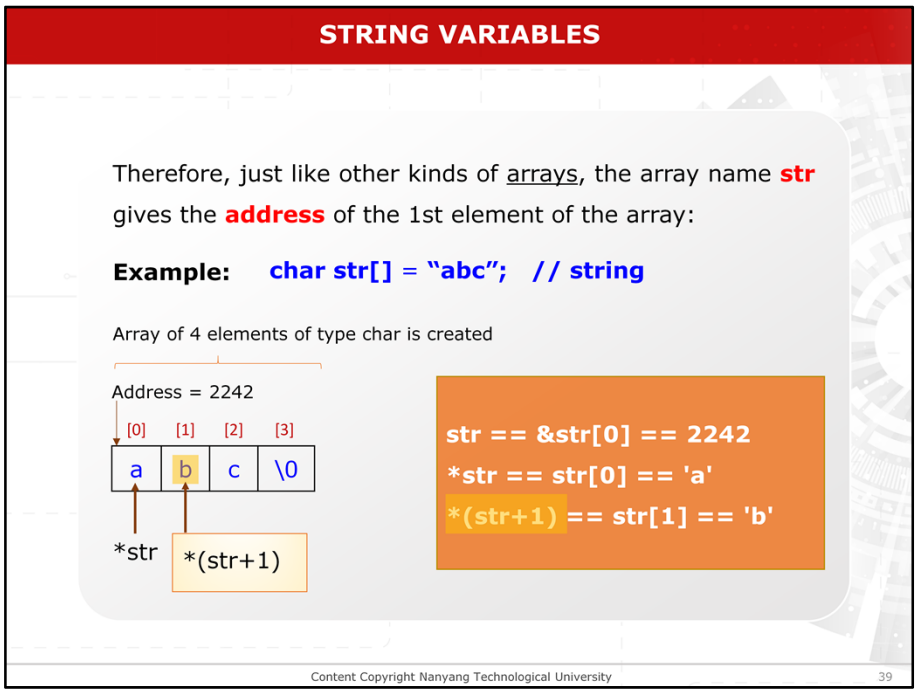
As a string constant returns a pointer to its first character, the array name **string** contains the address of the first element of the array as other kinds of arrays



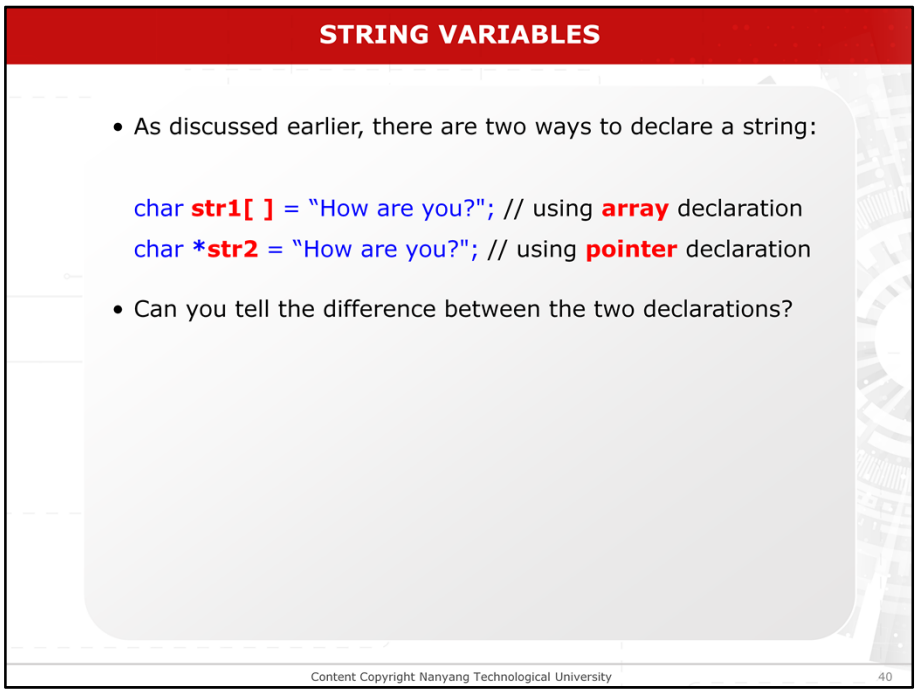
the **string** contains the address of the first element of the array of characters.



we can use **asterisk string** to retrieve the first element of the string.



similarly, we can use **asterisk string +1** to retrieve the second element of the character string.

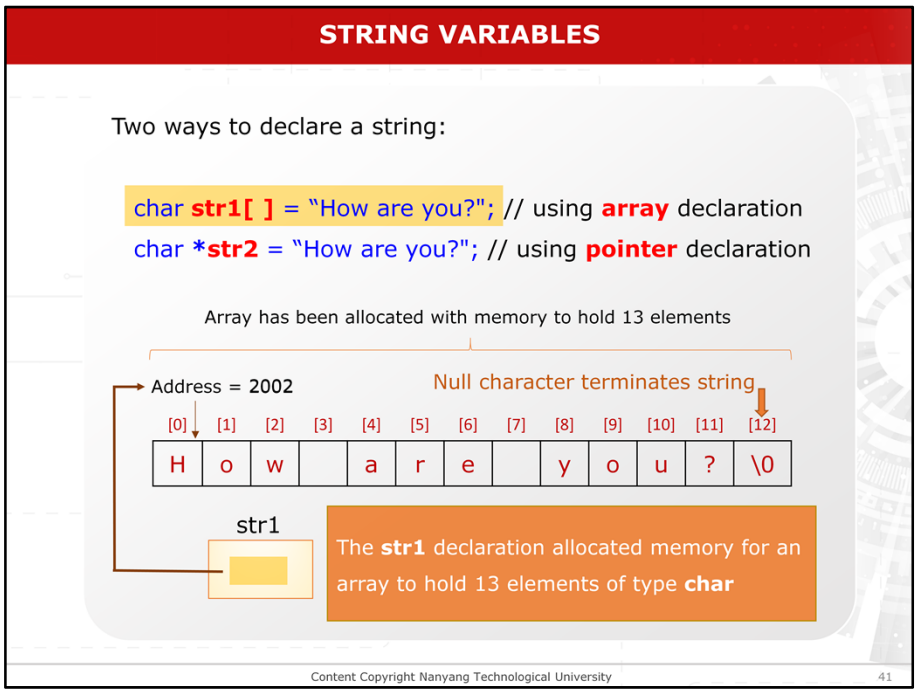


It is important to distinguish the declaration of string variables that use array or pointer. For example, the following strings are defined as:

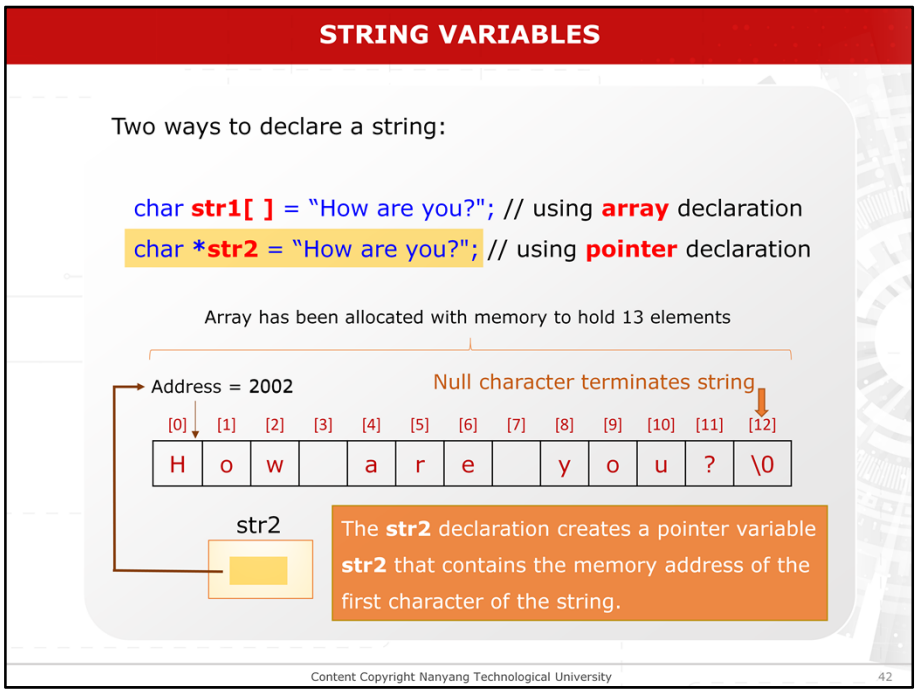
character array string 1 = double quote How are you? double quote;

character asterisk string 2 = double quote How are you? double quote;

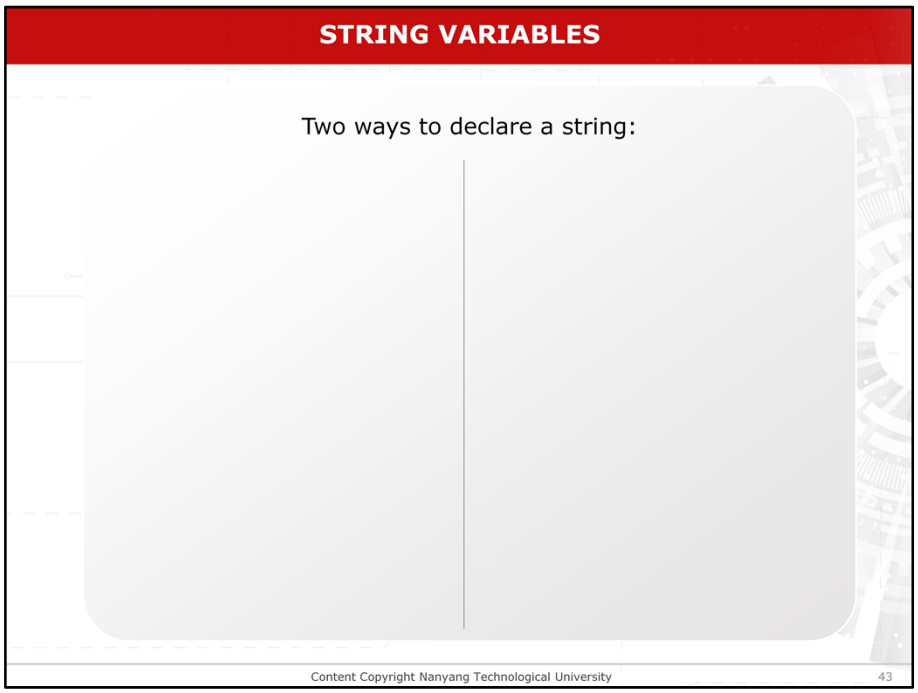
Question: Can you tell the difference between the two declarations?



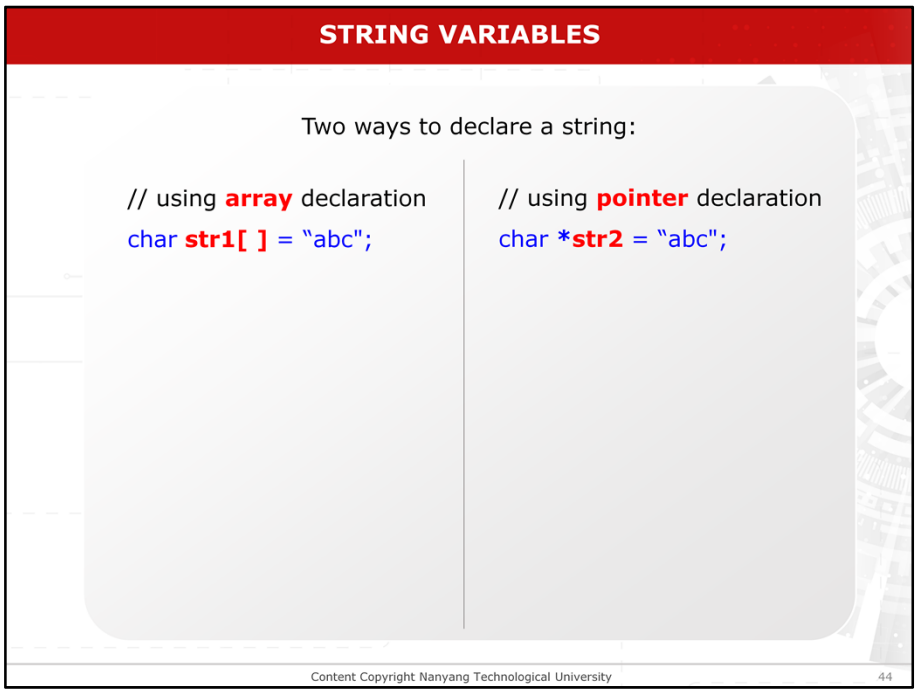
The **string 1** declaration creates an array of type **character**. The array has been allocated with memory to hold 13 elements including the null character. The C compiler also creates a pointer constant that is initialized to point to the first element of the array.



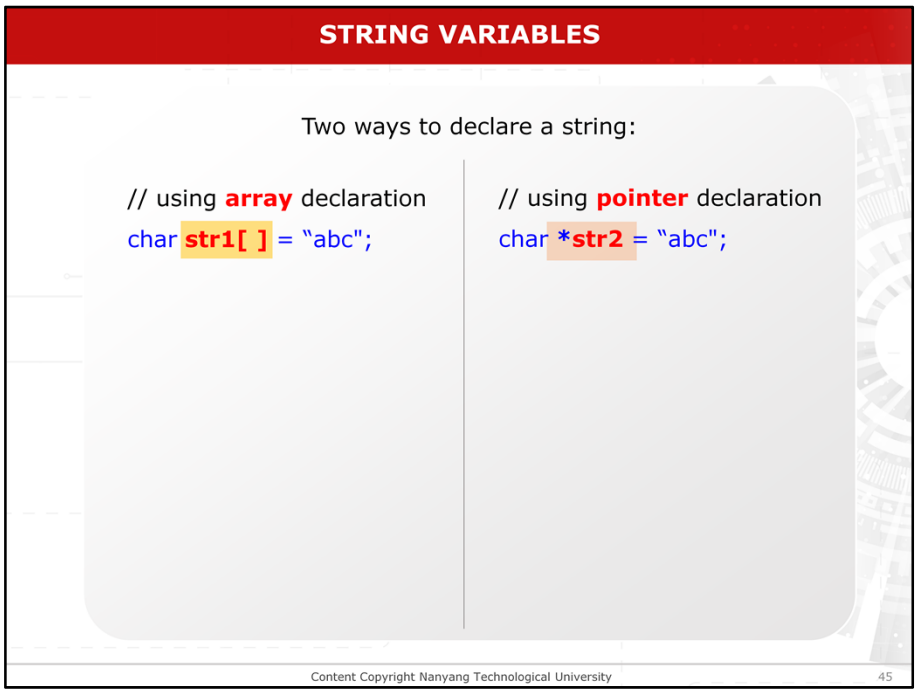
In the **string 2** declaration, the C compiler creates a pointer variable **string 2** that points to the first character of the string. It contains the memory address of the first character of the string '**H**'.



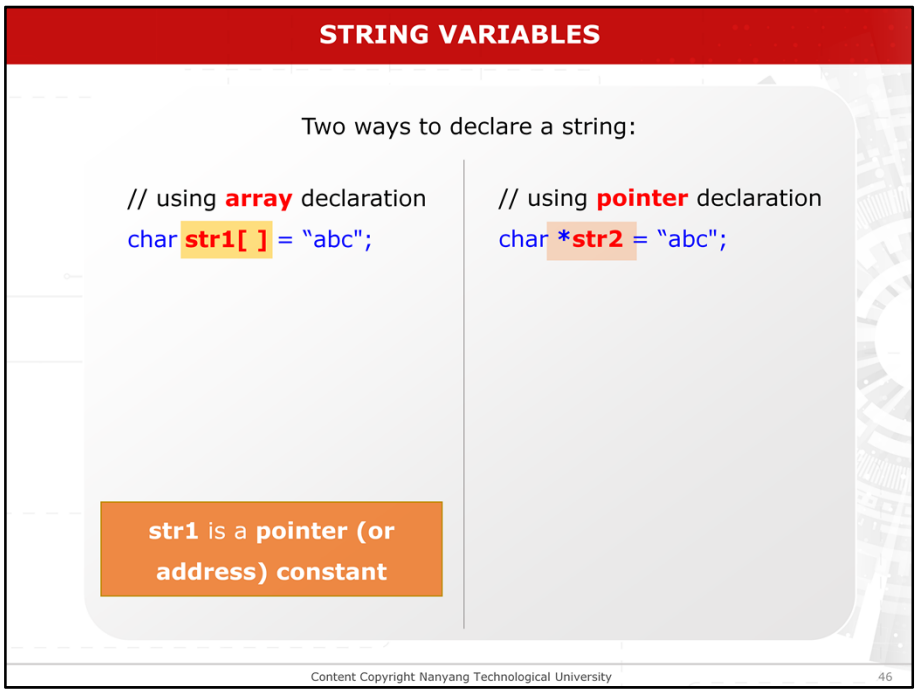
As discussed earlier, there are two ways to declare a string



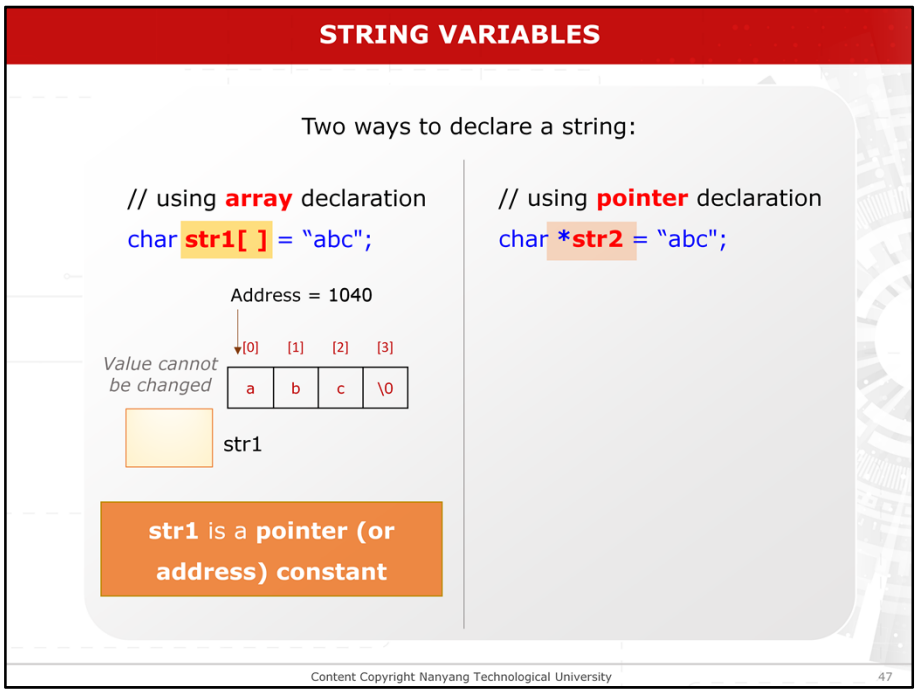
We can declare a string by using a array declaration or pointer declaration



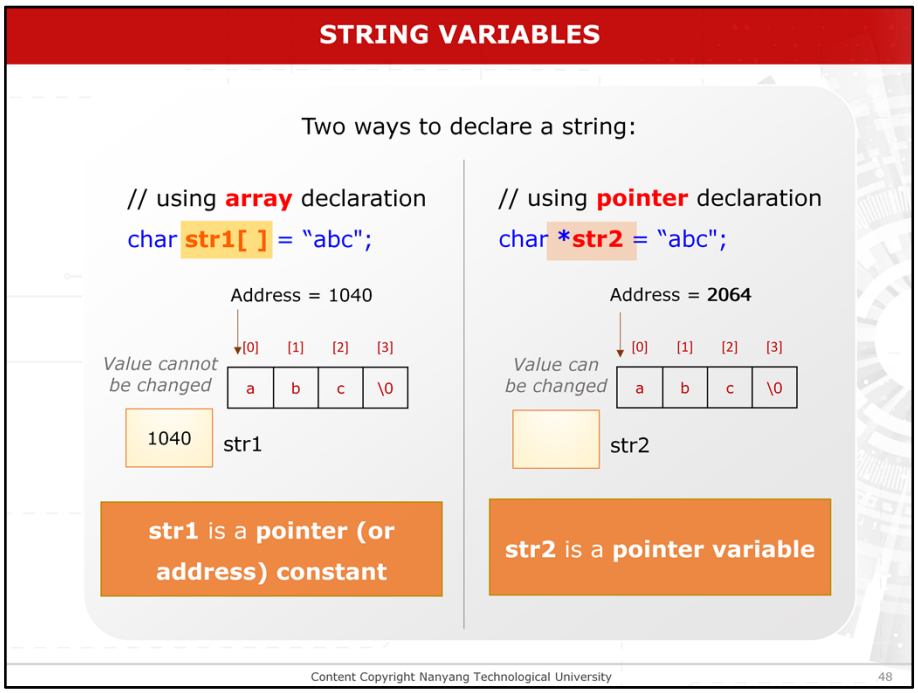
Both **string 1** and **string 2** are pointers.



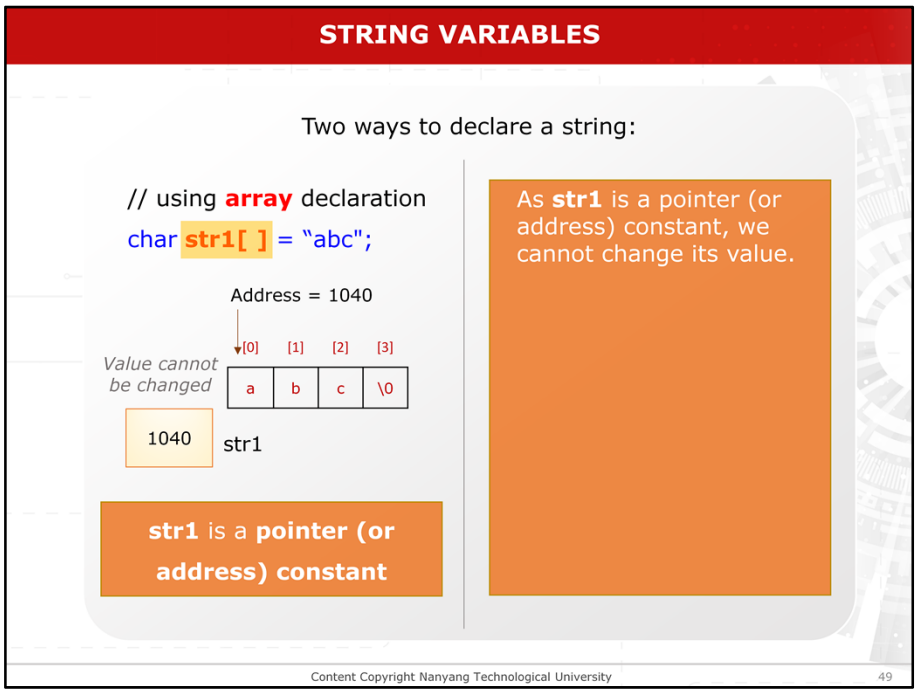
However, there is a difference between the two declarations.
String 1 is a **pointer (or address) constant**,



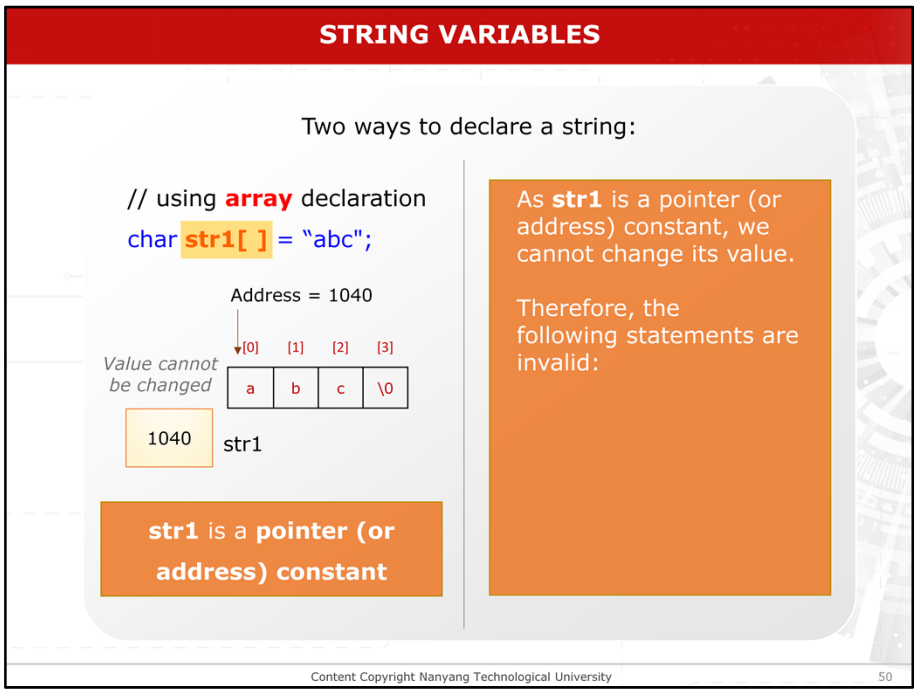
Pointer constant means that the value cannot be changed



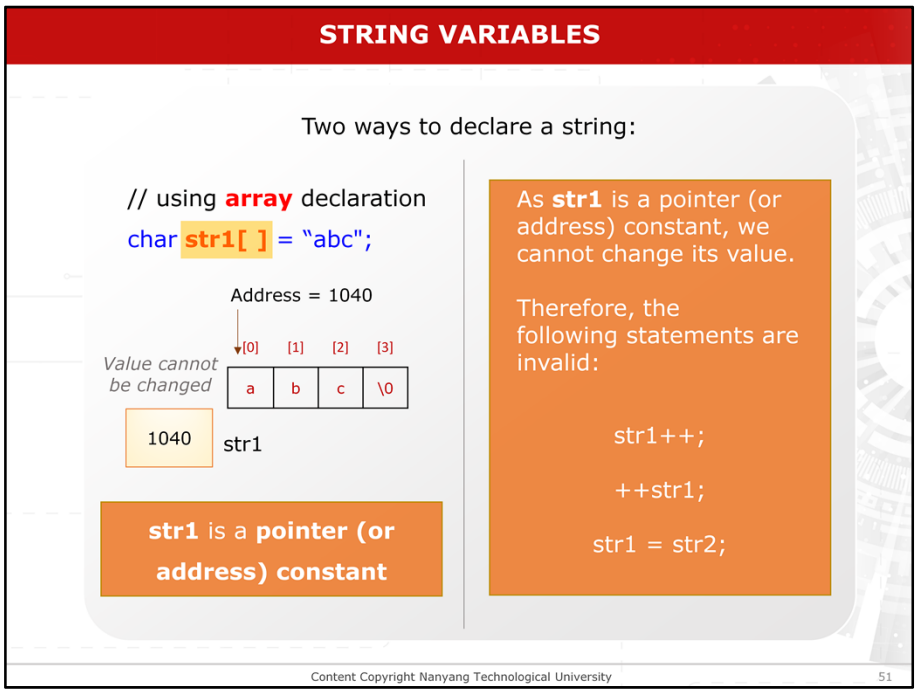
String 2 is a **pointer variable**.



As **string1** is a pointer (or address) constant, we cannot change its value.



Therefore, the following statements are invalid:



String 1 plus plus.

Or plus plus string 1.

Or string 1 equal string 2.

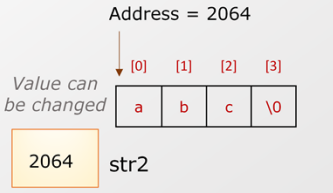
These are all invalid statements.

STRING VARIABLES

Two ways to declare a string:

Pointer variable allows its value to be changed.

```
// using pointer declaration
char *str2 = "abc";
```



str2 is a **pointer variable**

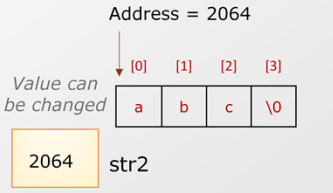
STRING VARIABLES

Two ways to declare a string:

Pointer variable allows its value to be changed.

Therefore the following statements are valid:

```
// using pointer declaration
char *str2 = "abc";
```



str2 is a **pointer variable**

STRING VARIABLES

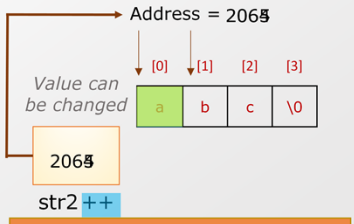
Two ways to declare a string:

Pointer variable allows its value to be changed.

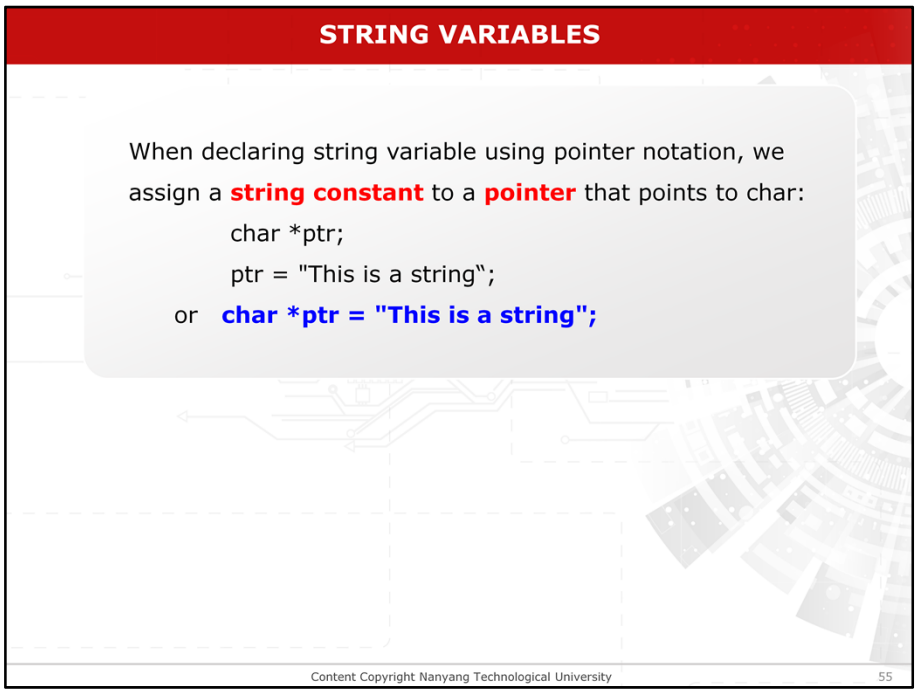
Therefore the following statements are valid:

```
str2++;
++str2;
str2 = str1;
```

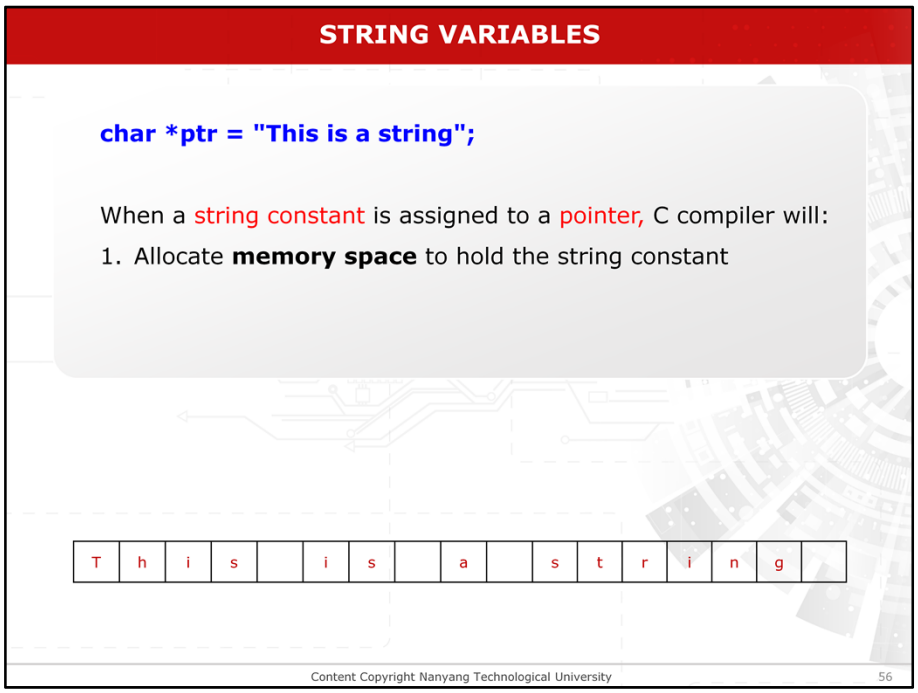
```
// using pointer declaration
char *str2 = "abc";
```



str2 is a **pointer variable**



We can assign a string constant to a pointer that points to type **character**:
character asterisk pointer = double quote This is a string double quote;



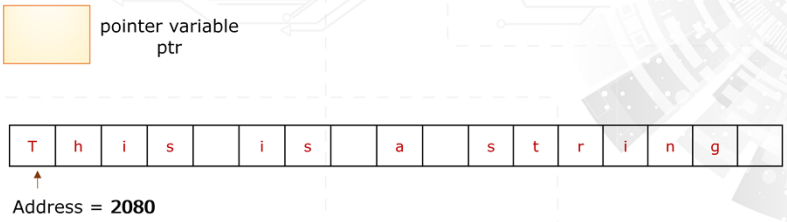
When a string constant is assigned to a pointer, the C compiler will allocate memory space to hold the string constant

STRING VARIABLES

char *ptr = "This is a string";

When a string constant is assigned to a pointer, C compiler will:

1. Allocate **memory space** to hold the string constant
2. Store the starting address of the string in the pointer variable



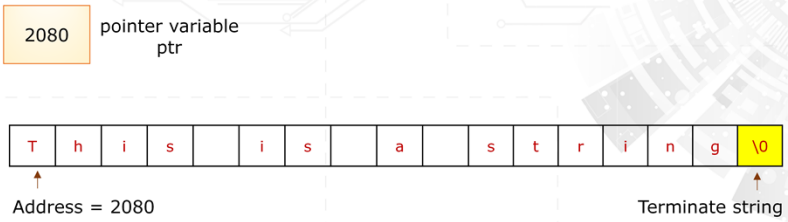
To store the starting address of the string (that is the address of the character **T**) in the pointer

STRING VARIABLES

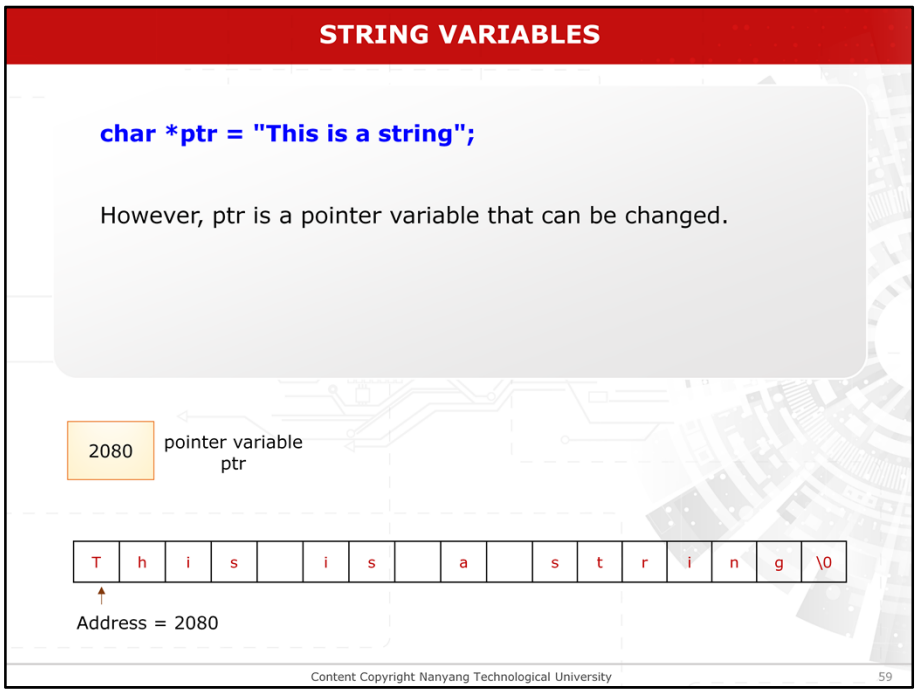
char *ptr = "This is a string";

When a string constant is assigned to a pointer, C compiler will:

1. Allocate **memory space** to hold the string constant
2. Store the starting address of the string in the pointer variable
3. Terminate the string with **NULL** ('\0') character



And terminate the string with a null character.



However, pointer is a pointer variable that can be changed.

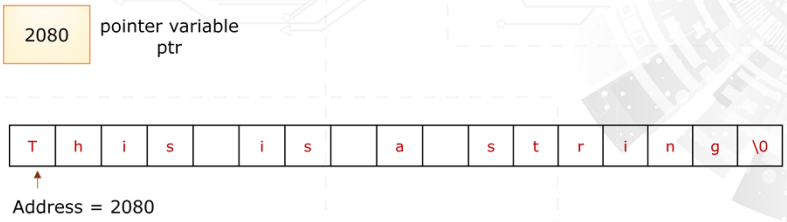
STRING VARIABLES

```
char *ptr = "This is a string";
```

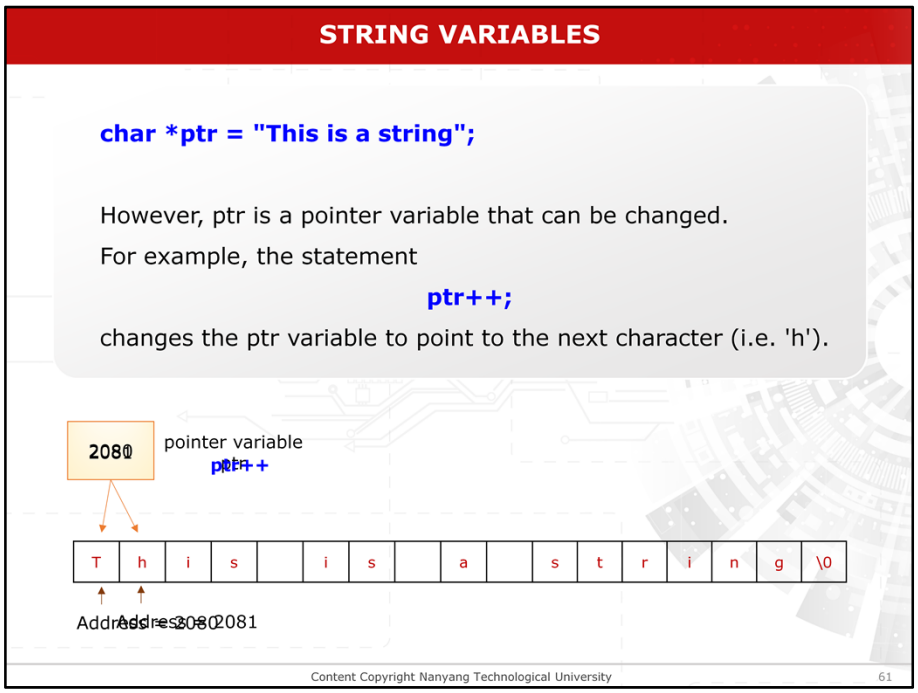
However, ptr is a pointer variable that can be changed.

For example, the statement

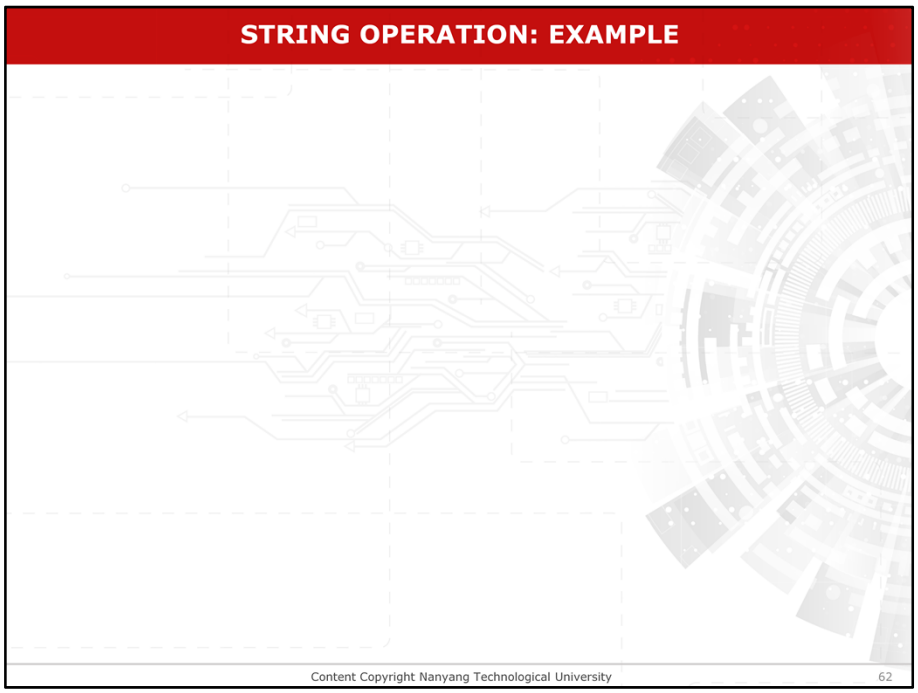
```
ptr++;
```



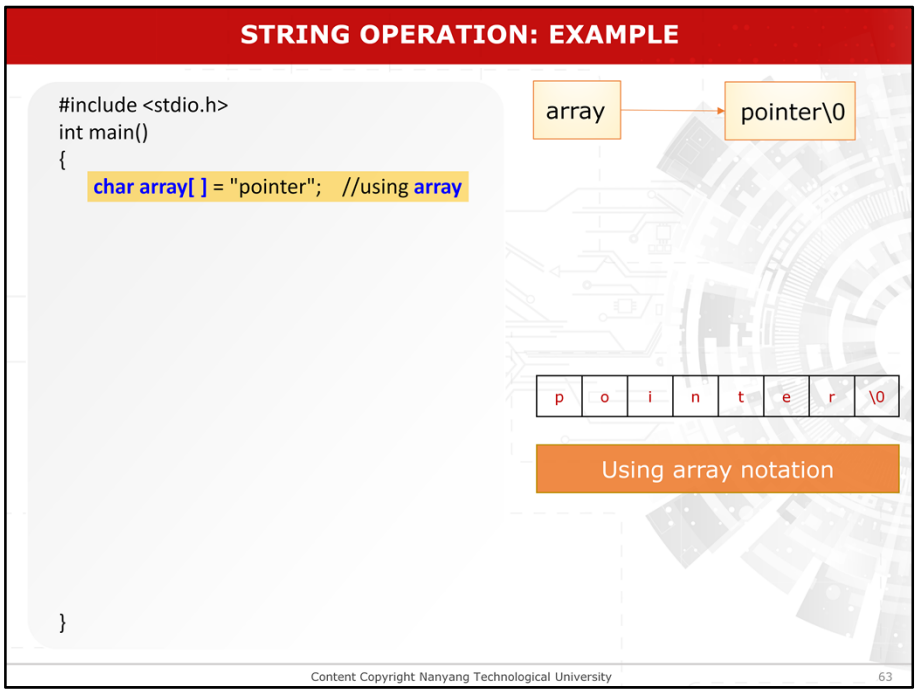
For example, the statement pointer++



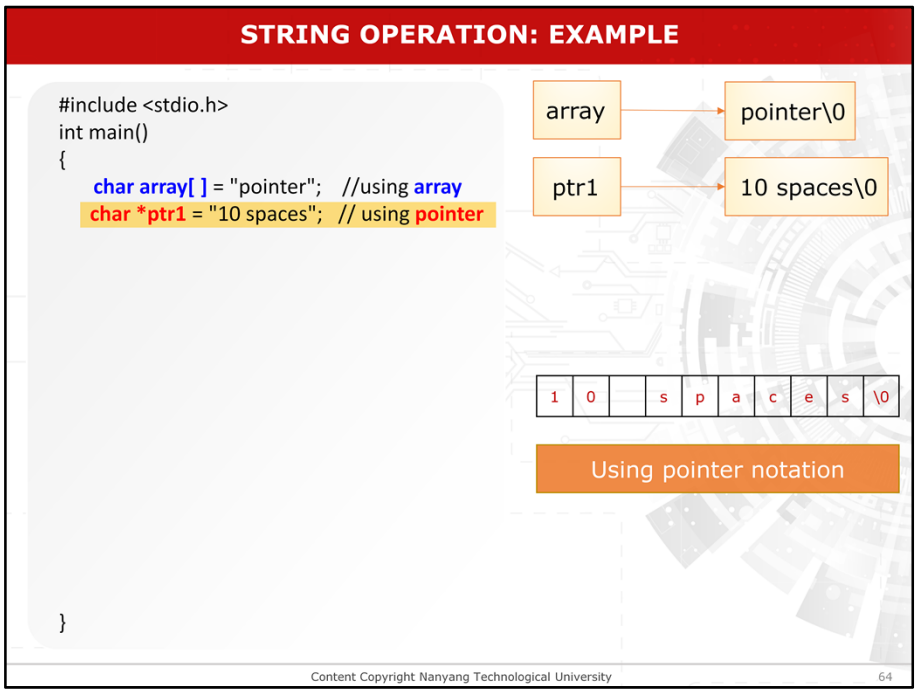
changes the **pointer** variable to point to the next character (that is '**h**') in the string "**This is a string**".



String operation: Example



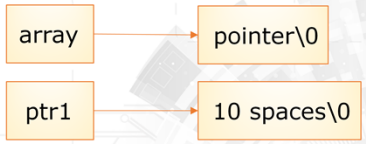
The declaration
Character array = double quote pointer double quote;
declares a string variable **array** using array notation which is initialized with 7 characters plus the null character.



The declaration
Character asterisk pointer 1 = double quote 10 spaces double quote;
declares a string variable **pointer 1** using pointer notation which is initialized and pointed to the string "**10 spaces**".

STRING OPERATION: EXAMPLE

```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";    //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);




}
```



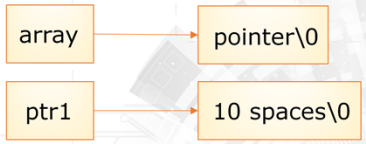
Output

ptr1 = 10 spaces

When **pointer1** is printed, the string **"10 spaces"** will be displayed.

STRING OPERATION: EXAMPLE

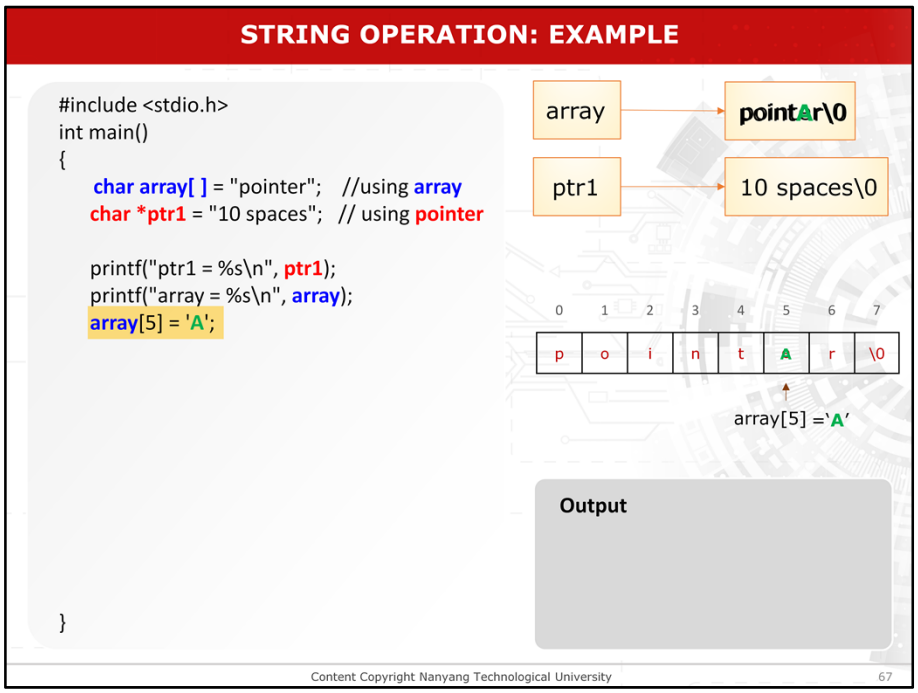
```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";    //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);




}
```



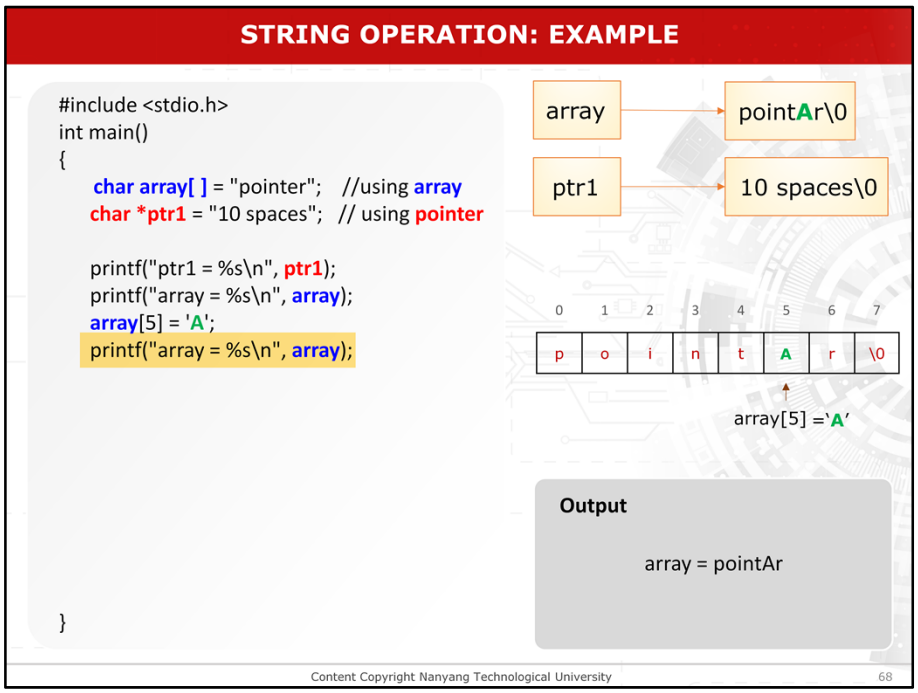
Output

array = pointer

When **array** is printed, the string **"pointer"** will be displayed.



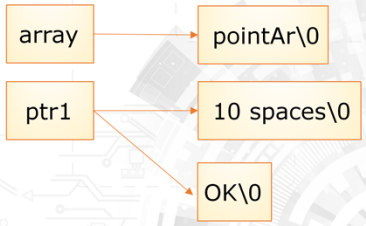
The statement
Array 5 = single quote A single quote;
is valid and the string is updated accordingly.



When **array** is printed, the string **p o I n t A r** will be displayed.

STRING OPERATION: EXAMPLE

```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";




}
```

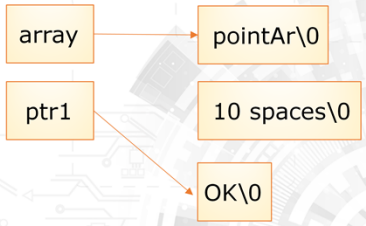


Output

The statement
Pointer 1 = double quote OK double quote;
will update **pointer 1** to point to a new string **"OK"**.

STRING OPERATION: EXAMPLE

```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";    //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);



}
```



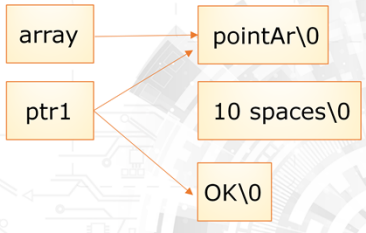
Output

ptr1 = OK

When **pointer1** is printed, the string **"OK"** will be displayed.

STRING OPERATION: EXAMPLE

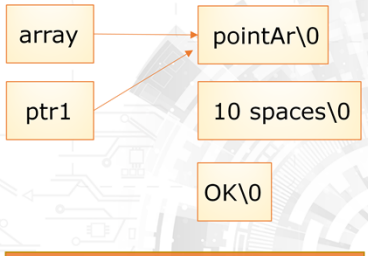
```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;



}
```



The statement
Pointer 1 = array;
changes the pointer variable **pointer 1** to point to the same address contained in **array**.

STRING OPERATION: EXAMPLE

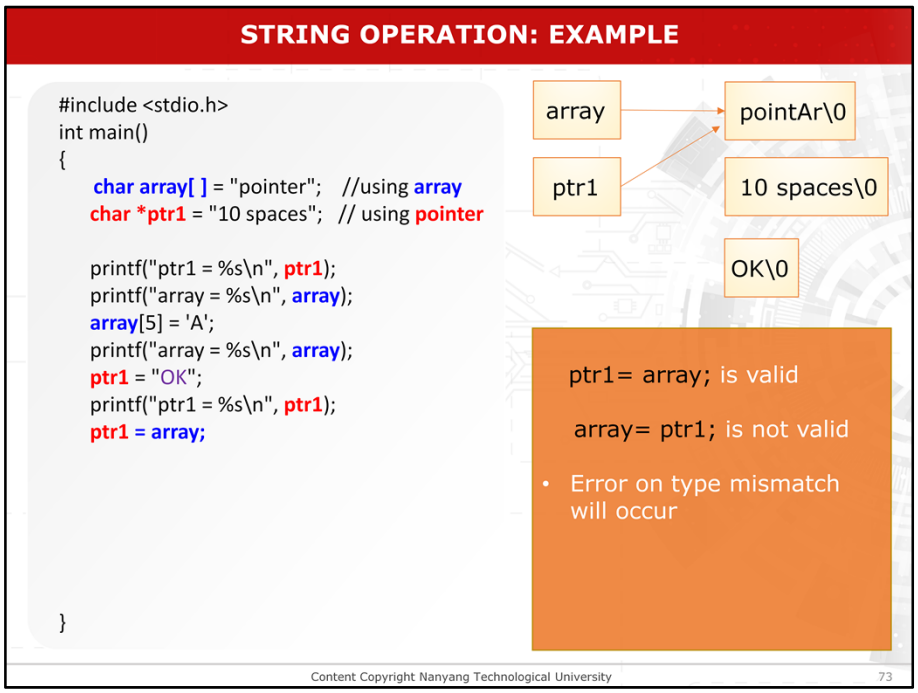
```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;




}
```



ptr1= array; is valid

array= ptr1; is not valid

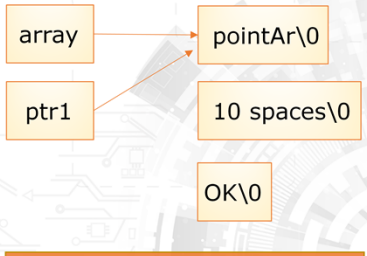
However, if we have
array = pointer 1;
which will be invalid



because an error on type mismatch will occur.

STRING OPERATION: EXAMPLE

```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;




}
```



ptr1= array; is valid

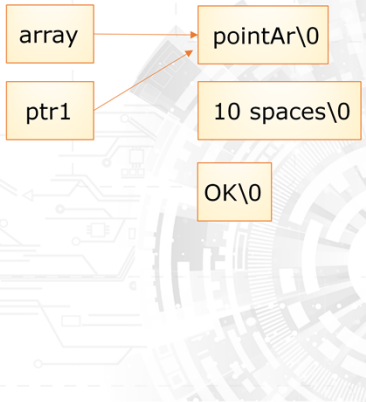
array= ptr1; is not valid

- Error on type mismatch will occur
- Not allowed to change the base address of array

Also, we are not allowed to change the base address of **array**.

STRING OPERATION: EXAMPLE

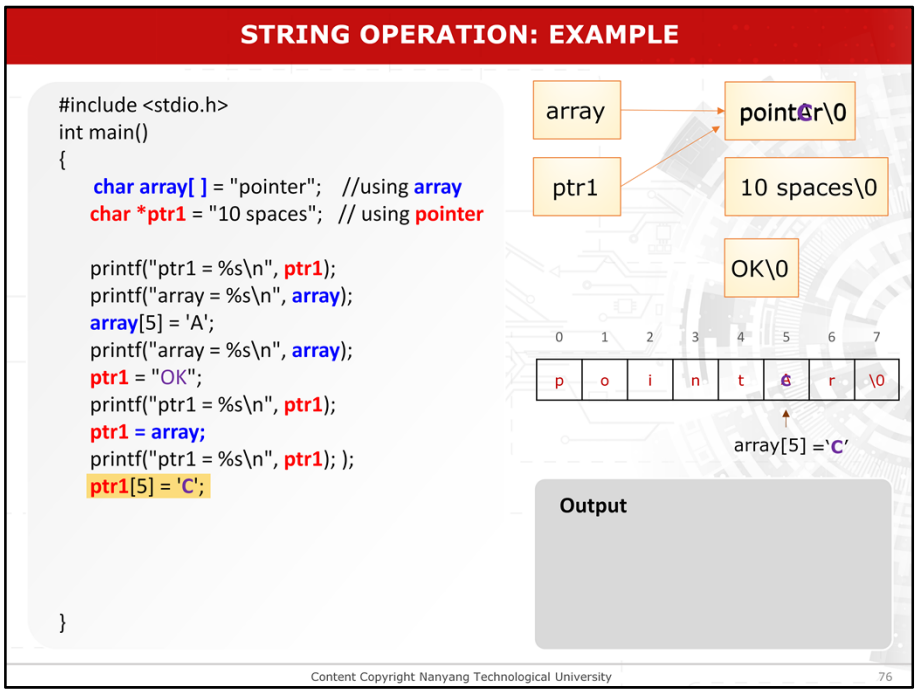
```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;
    printf("ptr1 = %s\n", ptr1);



}
```



Output

ptr1 = pointAr

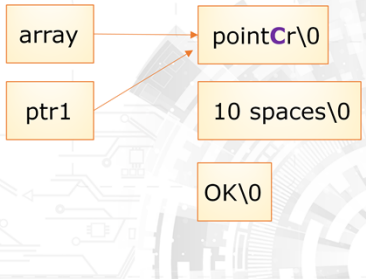
When **pointer 1** is printed, the string **p o I n t A r** will be displayed.



The statement
Pointer 1 5 = single quote C single quote;
is valid and the string is updated accordingly.

STRING OPERATION: EXAMPLE

```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;
    printf("ptr1 = %s\n", ptr1); );
    ptr1[5] = 'C';
    printf("ptr1 = %s\n", ptr1);


}
```



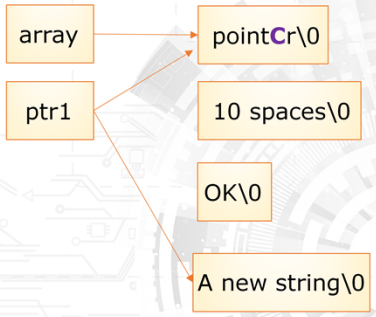
Output

ptr1 = pointCr

When **pointer 1** is printed, the string with **p o i n t c r** will be displayed.

STRING OPERATION: EXAMPLE

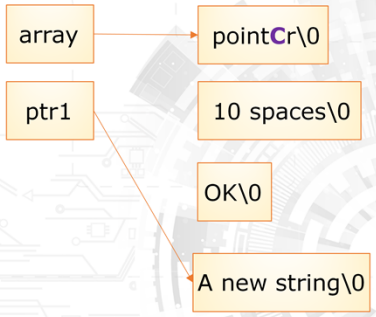
```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;
    printf("ptr1 = %s\n", ptr1); );
    ptr1[5] = 'C';
    printf("ptr1 = %s\n", ptr1);
    ptr1 = "A new string";


}
```



The statement
Pointer 1 = double quote A new string double quote;
is valid which changes the pointer variable **pointer 1** to point to the new string "**A new string**".

STRING OPERATION: EXAMPLE

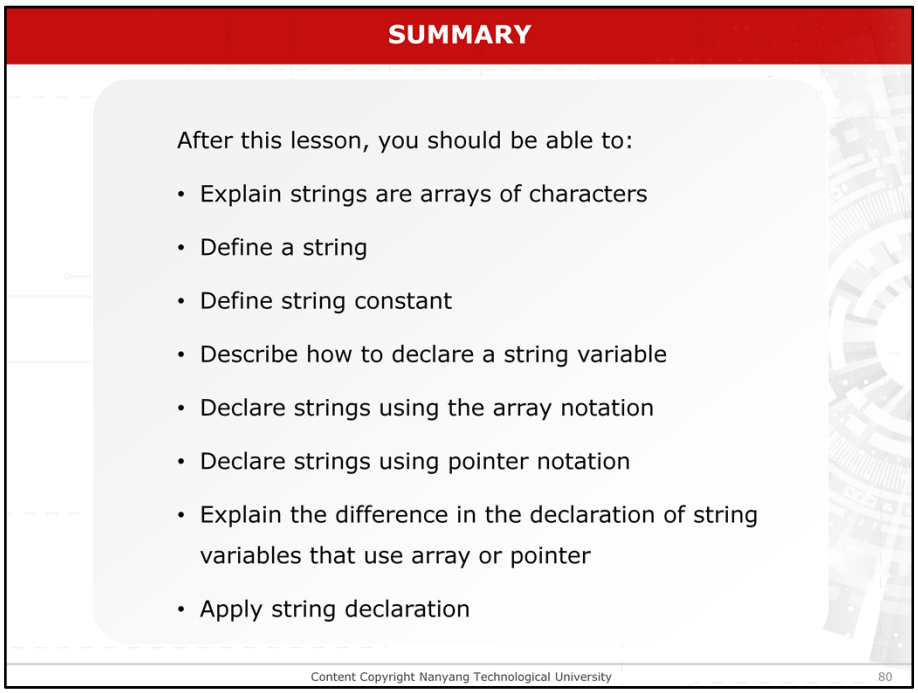
```
#include <stdio.h>
int main()
{
    char array[ ] = "pointer";   //using array
    char *ptr1 = "10 spaces";   // using pointer

    printf("ptr1 = %s\n", ptr1);
    printf("array = %s\n", array);
    array[5] = 'A';
    printf("array = %s\n", array);
    ptr1 = "OK";
    printf("ptr1 = %s\n", ptr1);
    ptr1 = array;
    printf("ptr1 = %s\n", ptr1);
    ptr1[5] = 'C';
    printf("ptr1 = %s\n", ptr1);
    ptr1 = "A new string";
    printf("ptr1 = %s\n", ptr1);
    return 0;
}
```



Output

ptr1 = A new string

When **pointer 1** is printed, the string **"A new string"** will be displayed.



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