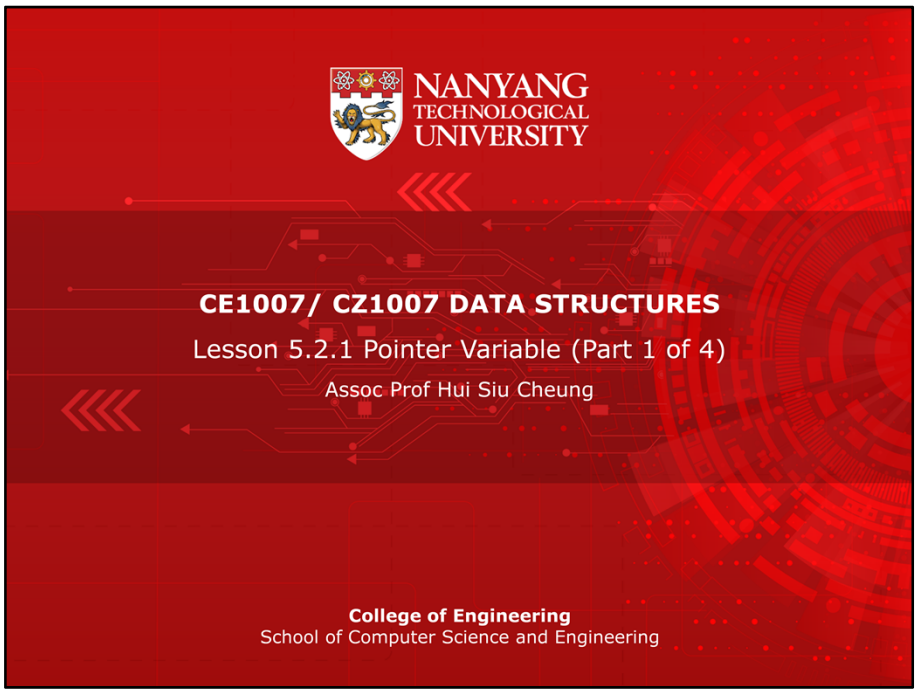
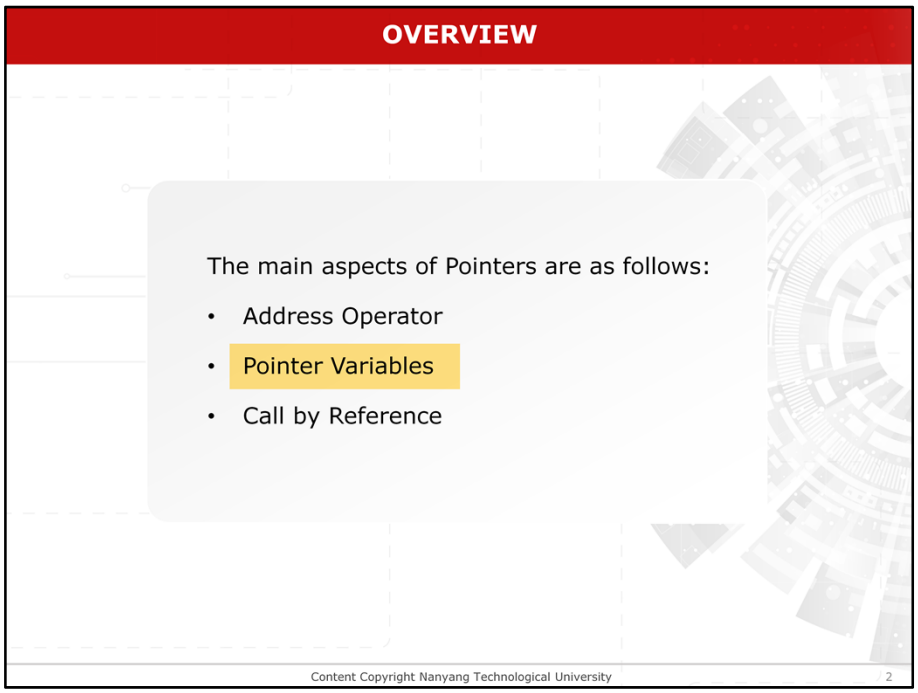
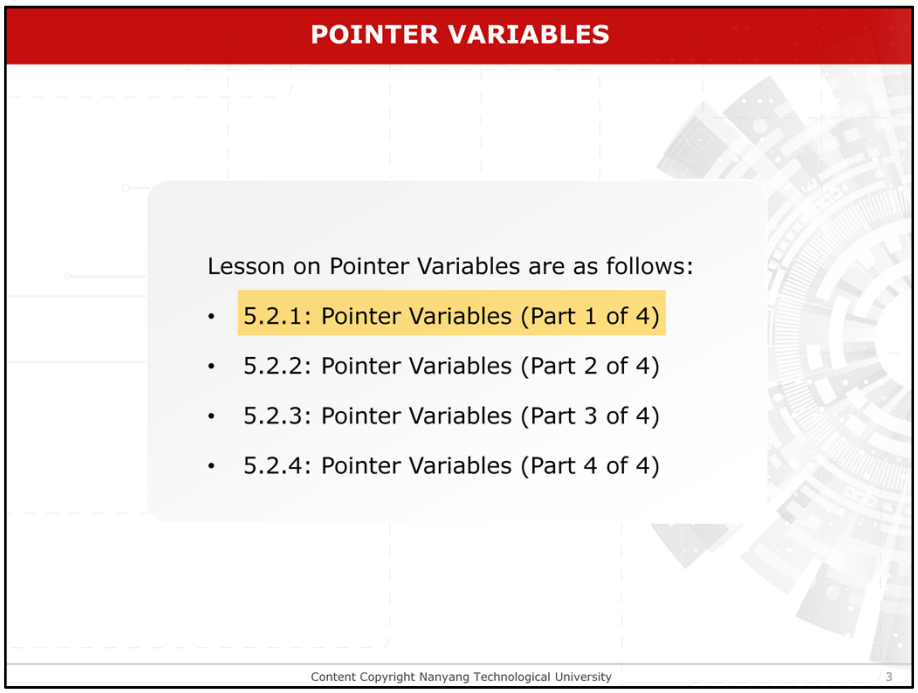


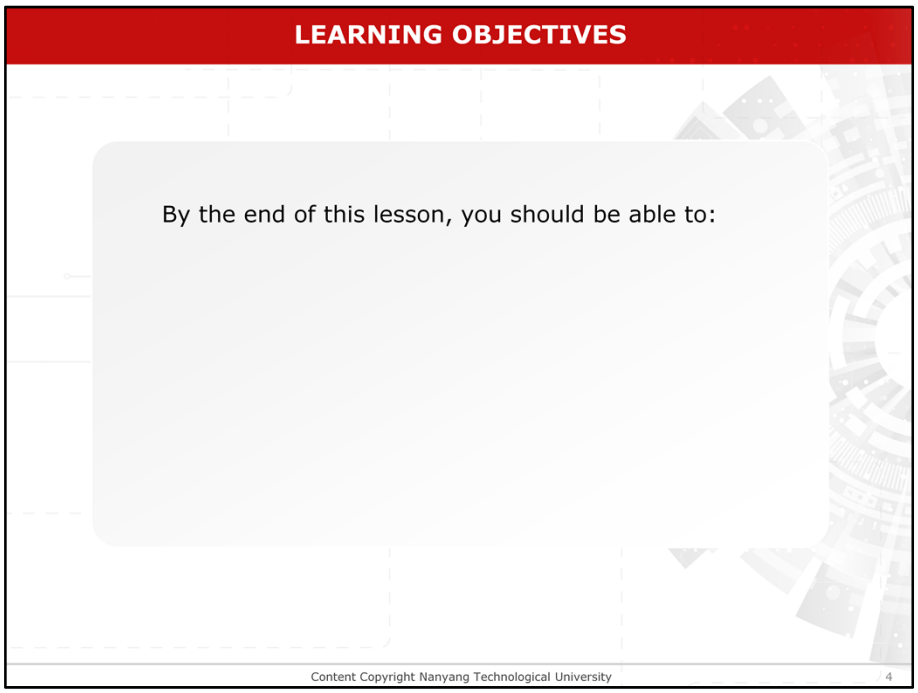
This lesson will discuss further on the concept of Pointer Variable and its application.



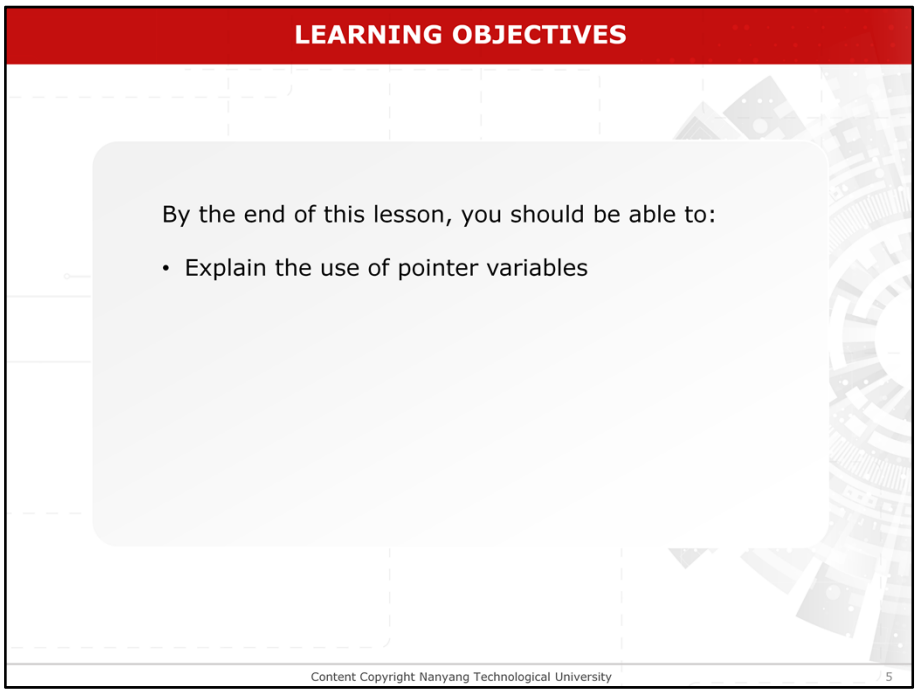
We recalled that there are 3 main aspects of Pointers to be learnt.
In this lesson, we will focus on Pointer Variables.



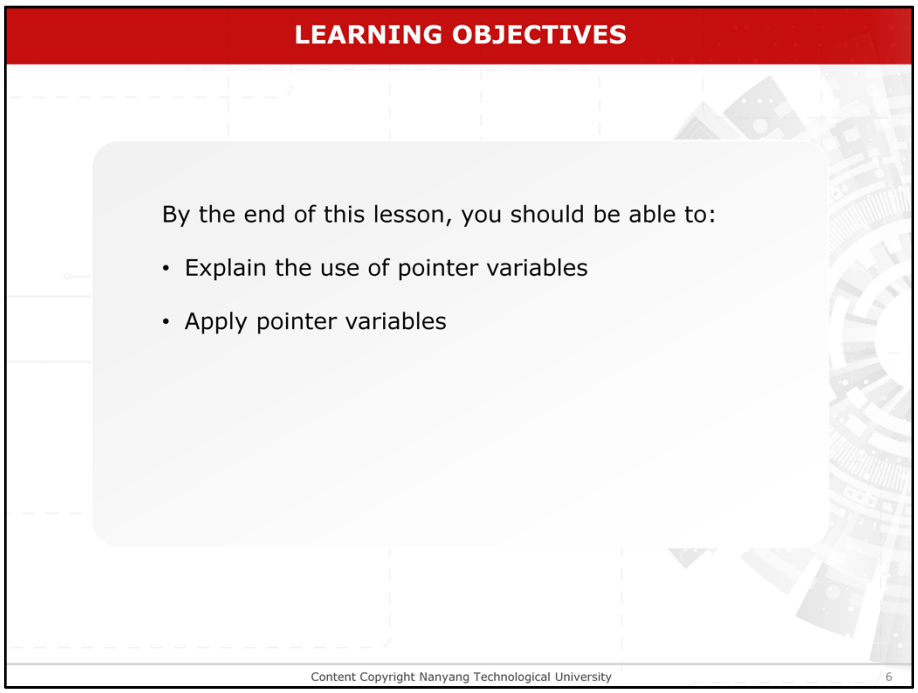
This lesson on Pointer Variables will be covered in 4 parts: 5.2.1, 5.2.2, 5.2.3 and 5.2.4.
This lesson is 5.2.1 and it covers all the main concepts of pointer variables.
Lesson 5.2.2 and 5.2.3 are detailed examples of how to apply pointer variables in programming codes.
Lesson 5.2.4 will be a summary to the lesson on Pointer Variables.



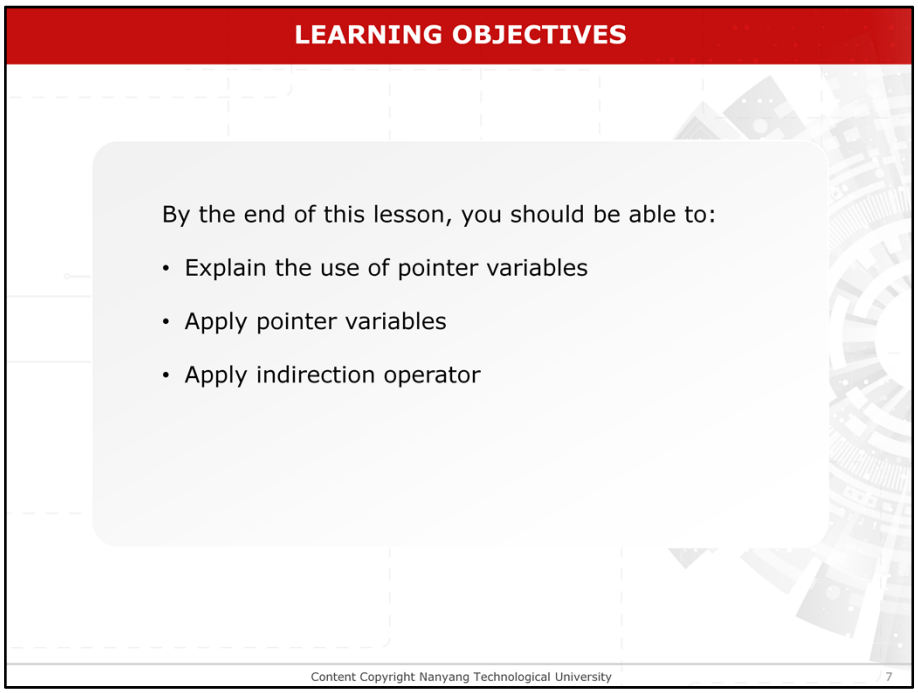
By the end of this lesson, you should be able to:



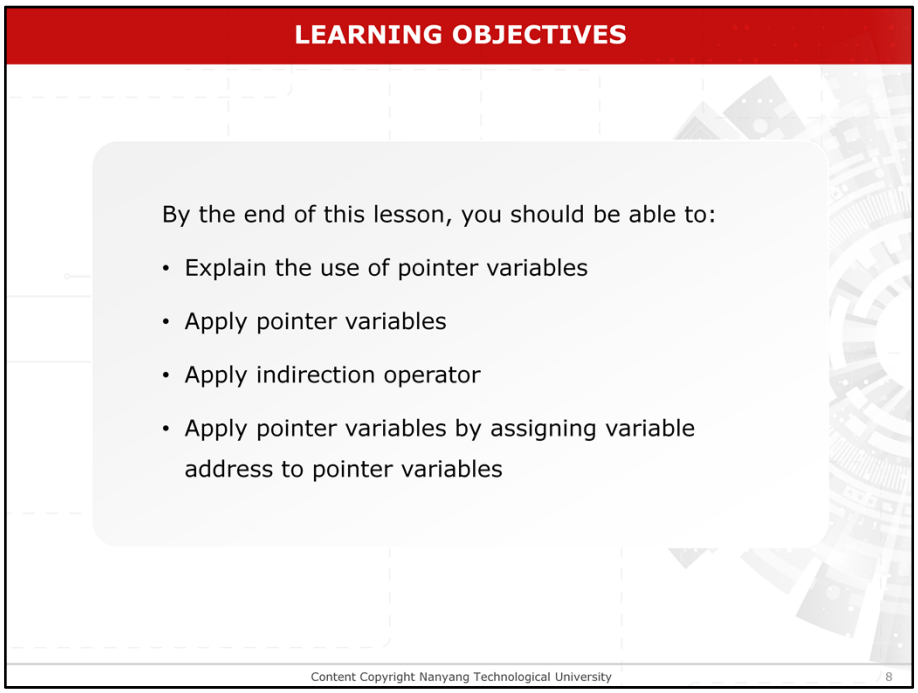
Explain the use of pointer variables



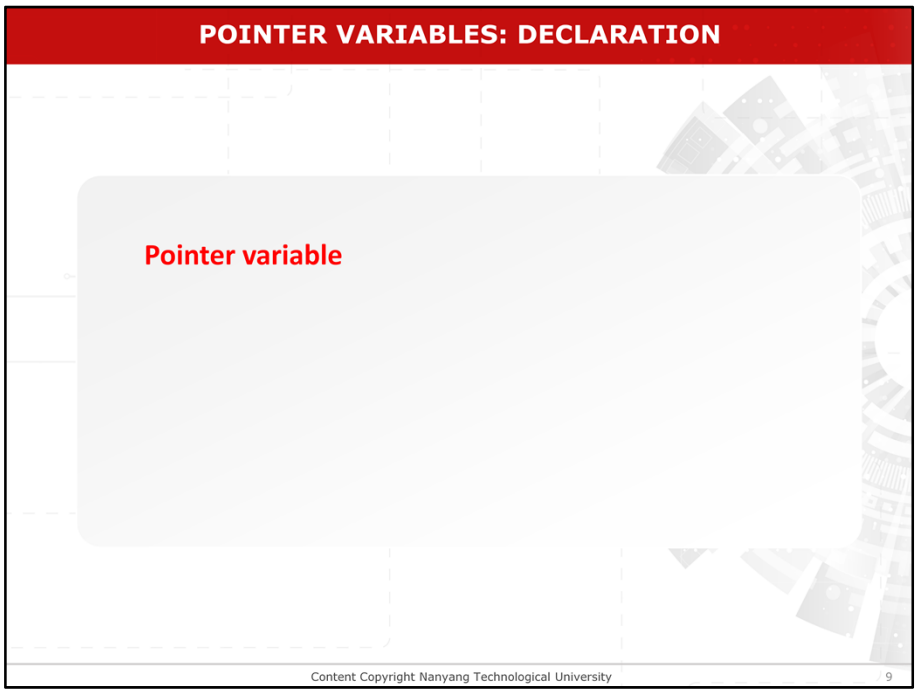
Apply pointer variable



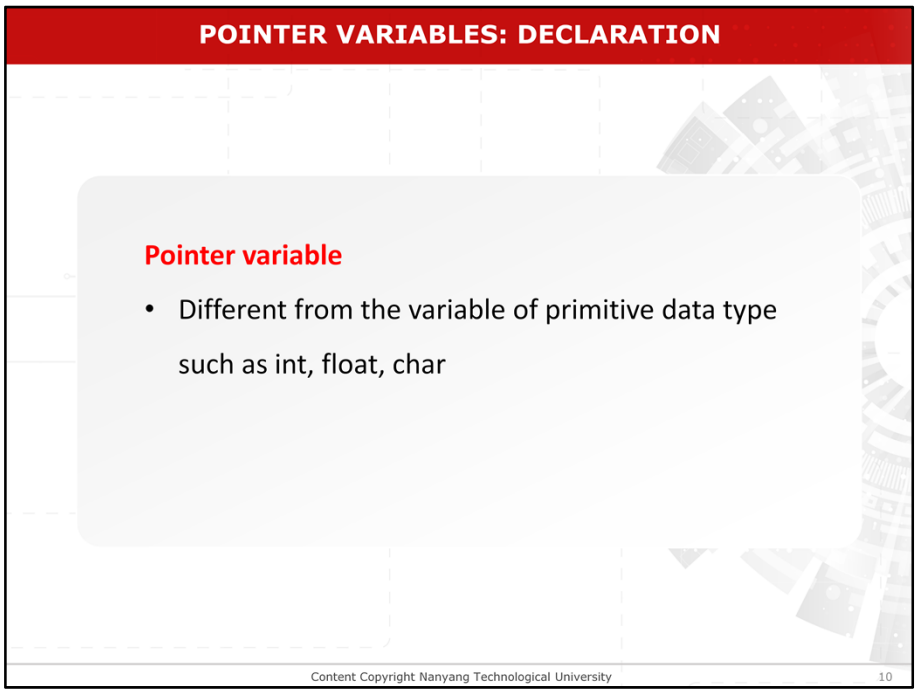
Apply indirection operator



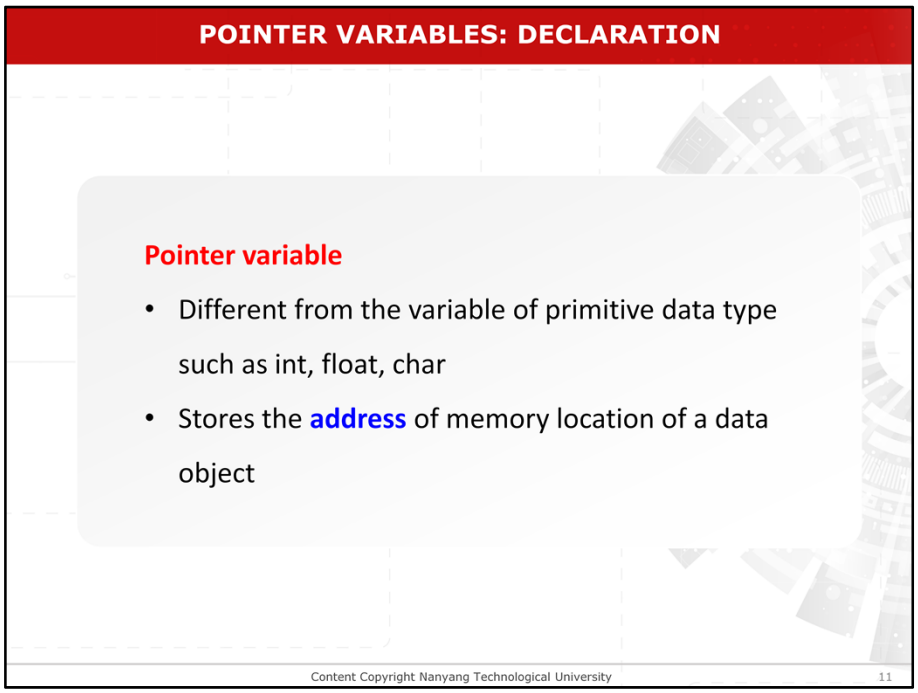
Apply pointer variables by assigning variable address to pointer variables



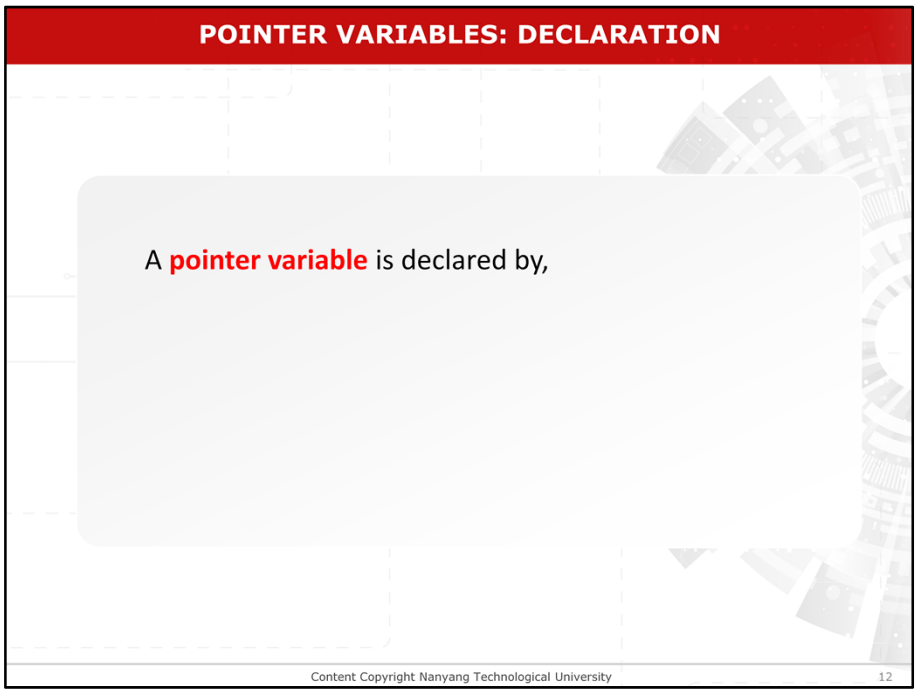
Pointer variable



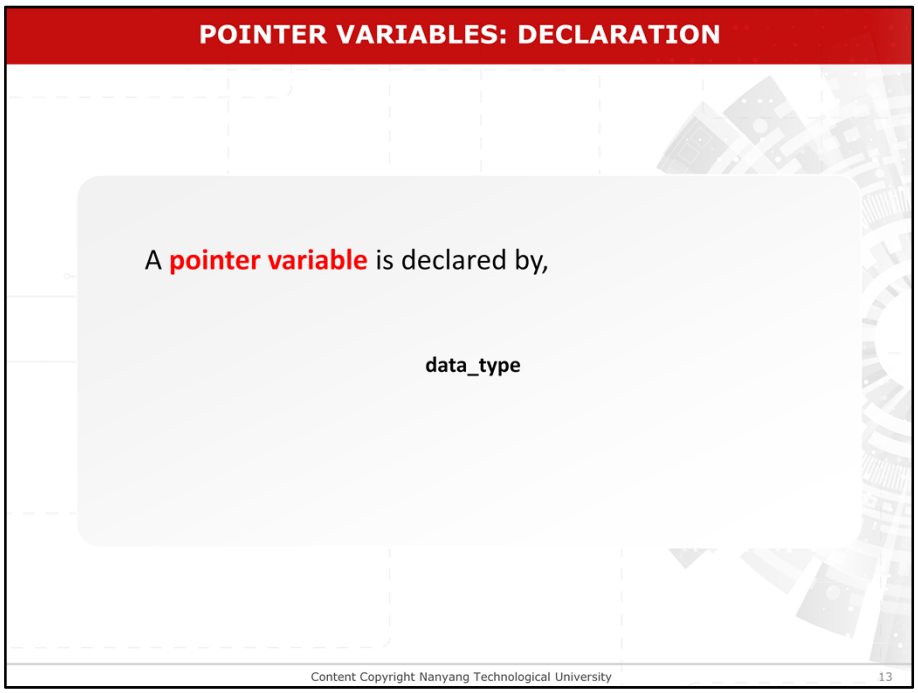
Pointer variable is different from the variables of primitive data types which stores the values directly



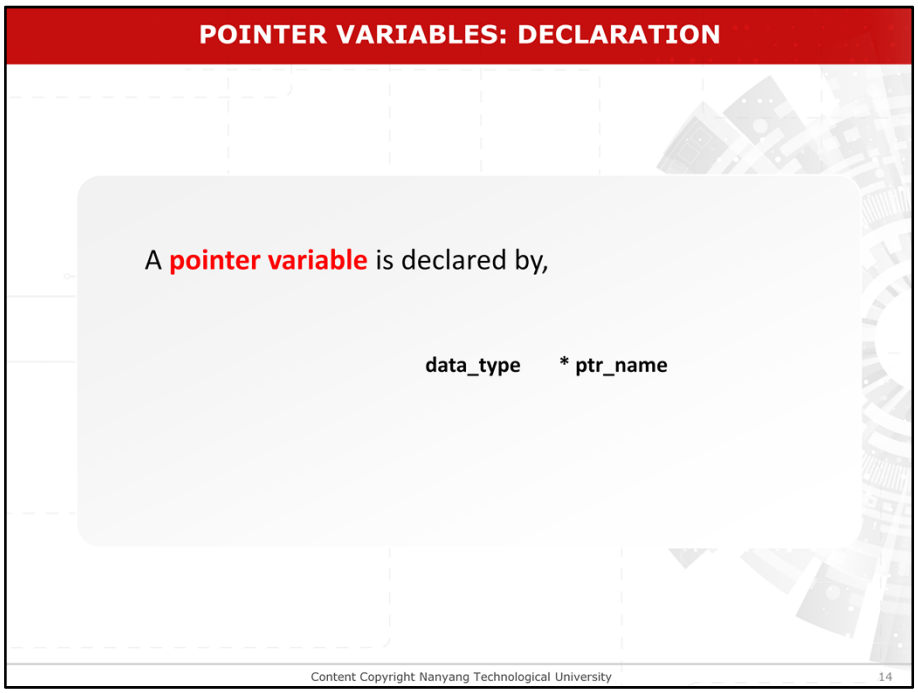
A pointer variable stores the addresses of memory locations of a data object



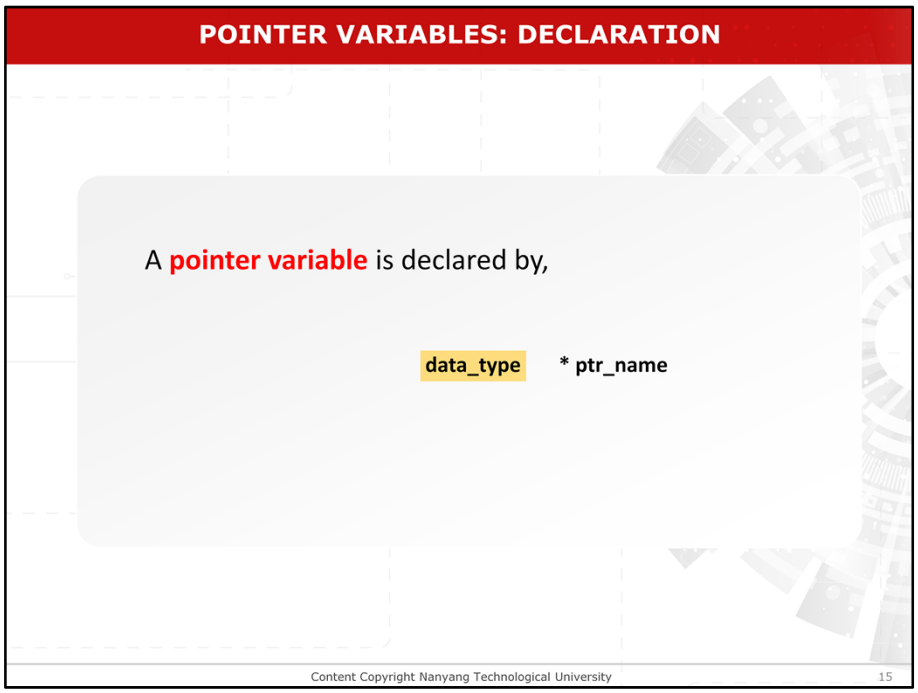
A *pointer variable* is declared by:


POINTER VARIABLES: DECLARATION
A pointer variable is declared by,
data_type
Content Copyright Nanyang Technological University
13

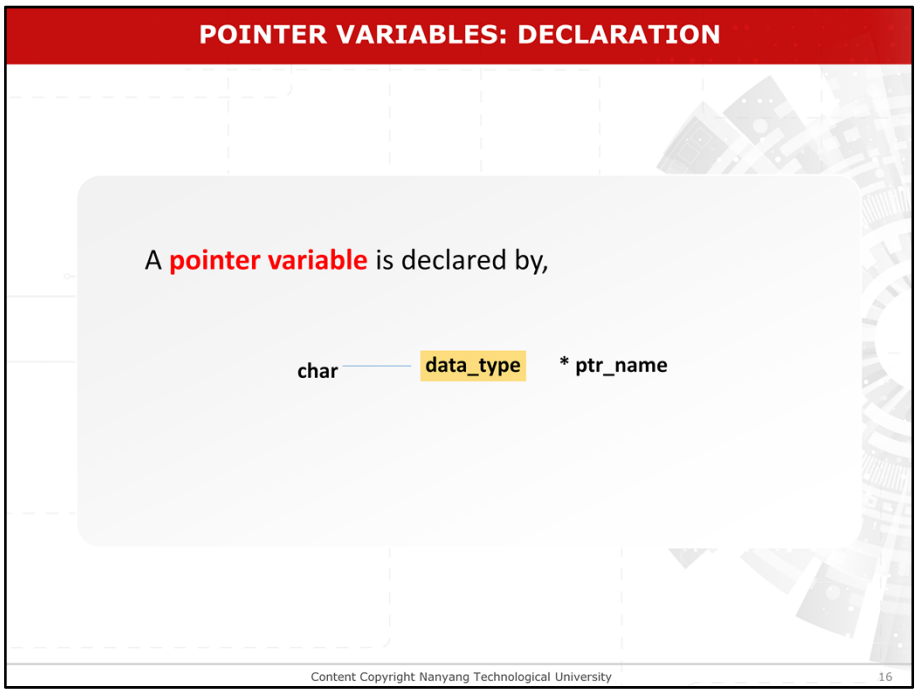
data_type



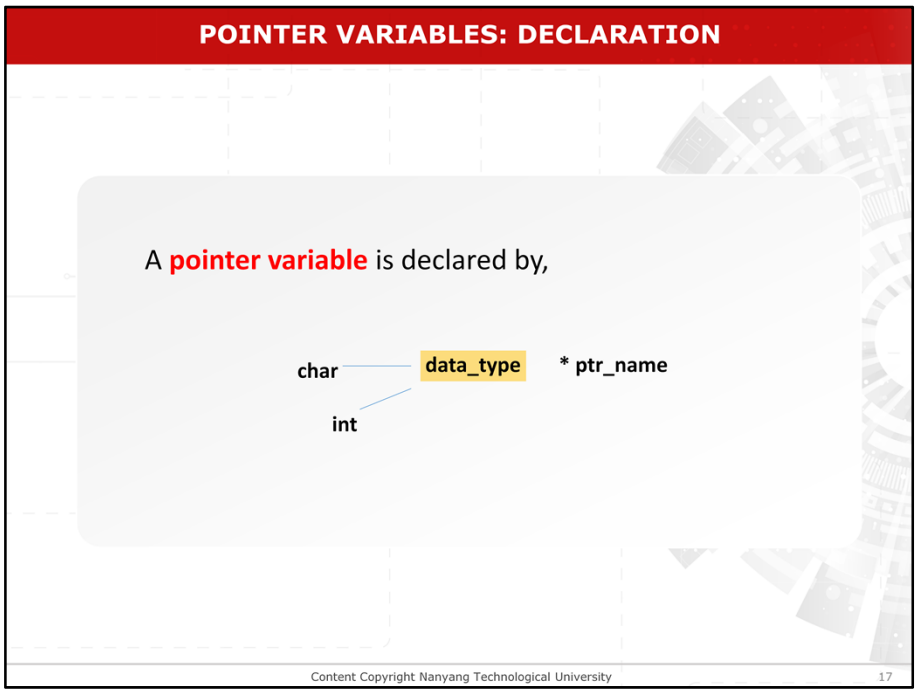
asterisk **pointer_name;**



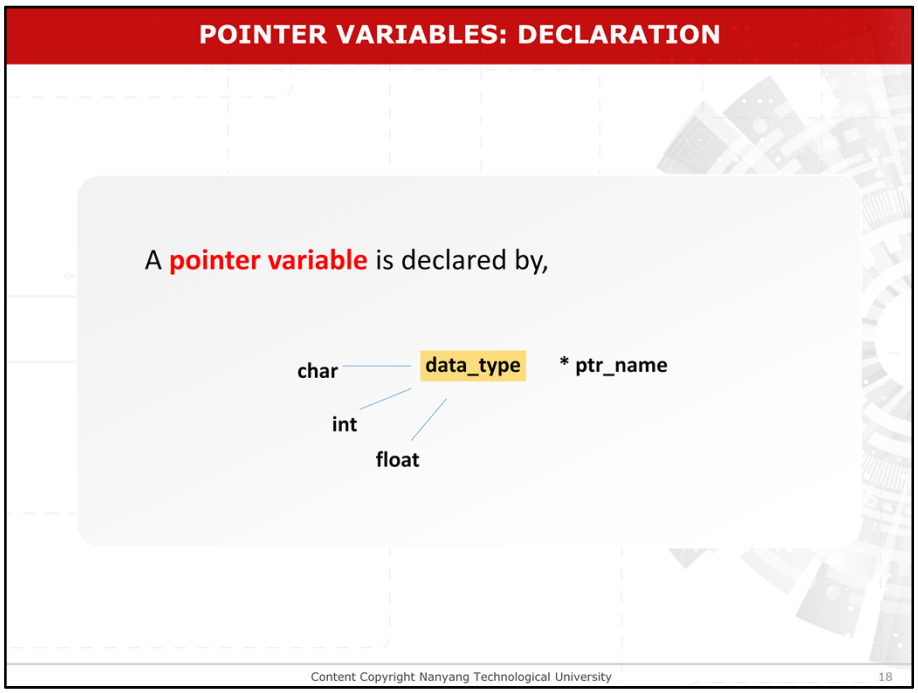
where **data_type** can be any C data type such as


POINTER VARIABLES: DECLARATION
A pointer variable is declared by,
char
data_type
* ptr_name
Content Copyright Nanyang Technological University
16

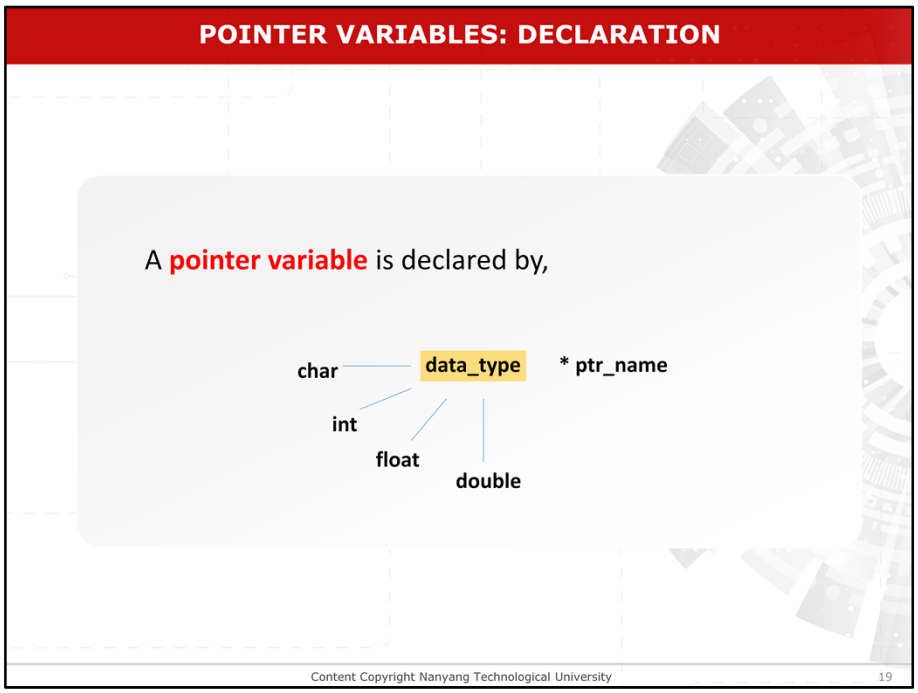
Character


POINTER VARIABLES: DECLARATION
A pointer variable is declared by,
char
int
data_type
* ptr_name
Content Copyright Nanyang Technological University
17

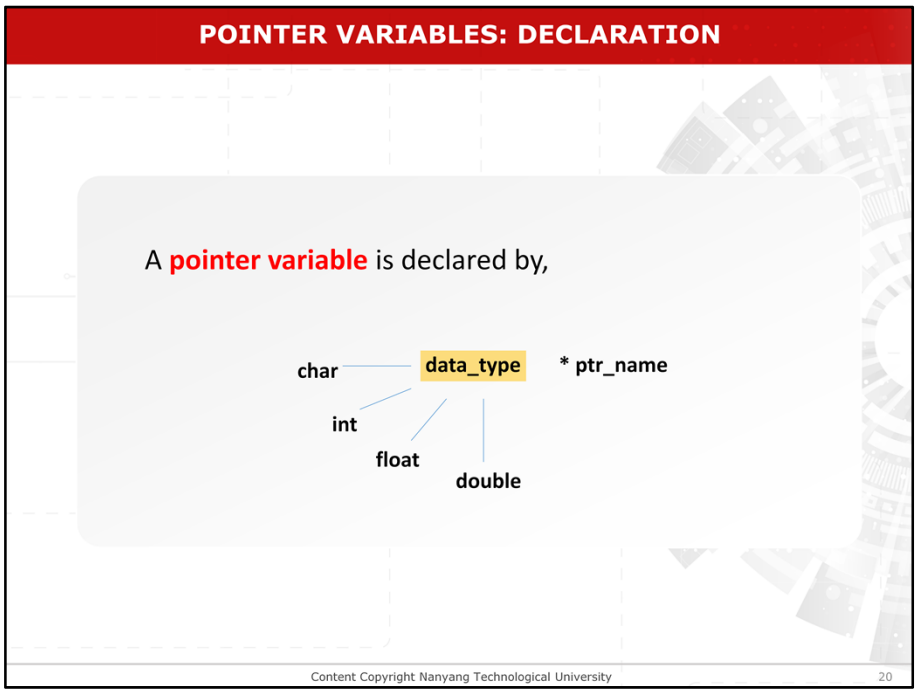
Integer


POINTER VARIABLES: DECLARATION
A pointer variable is declared by,
char
int
float
data_type
* ptr_name
Content Copyright Nanyang Technological University
18

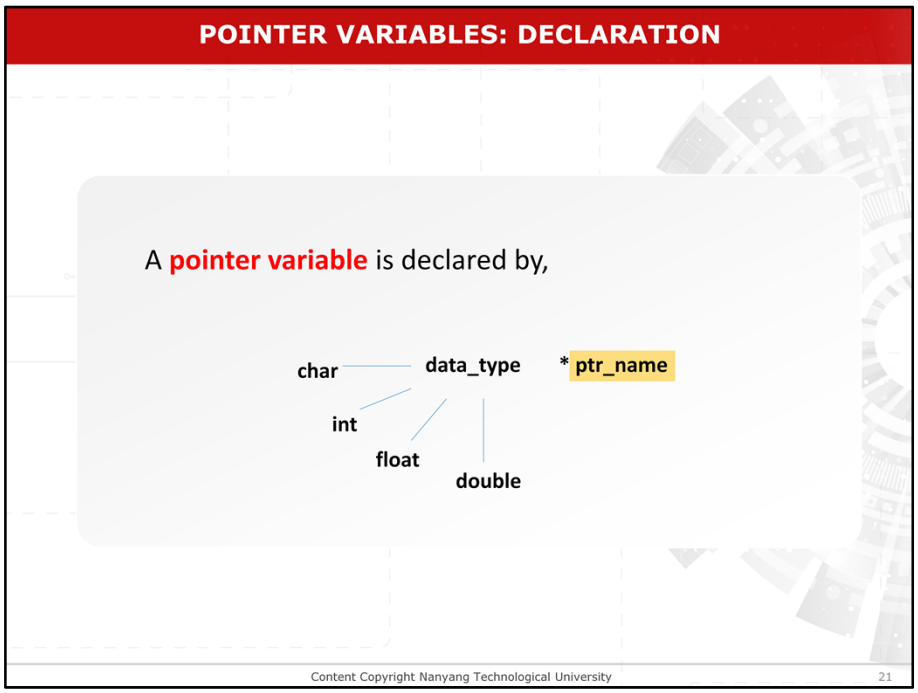
Float


POINTER VARIABLES: DECLARATION
A pointer variable is declared by,
char
data_type
* ptr_name
int
float
double
Content Copyright Nanyang Technological University
19

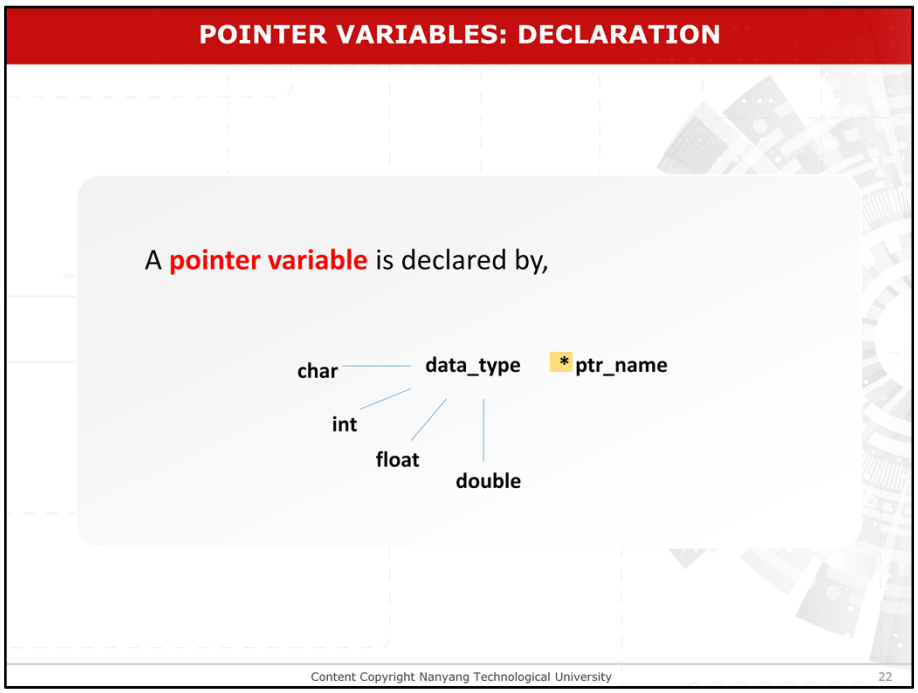
or **double**



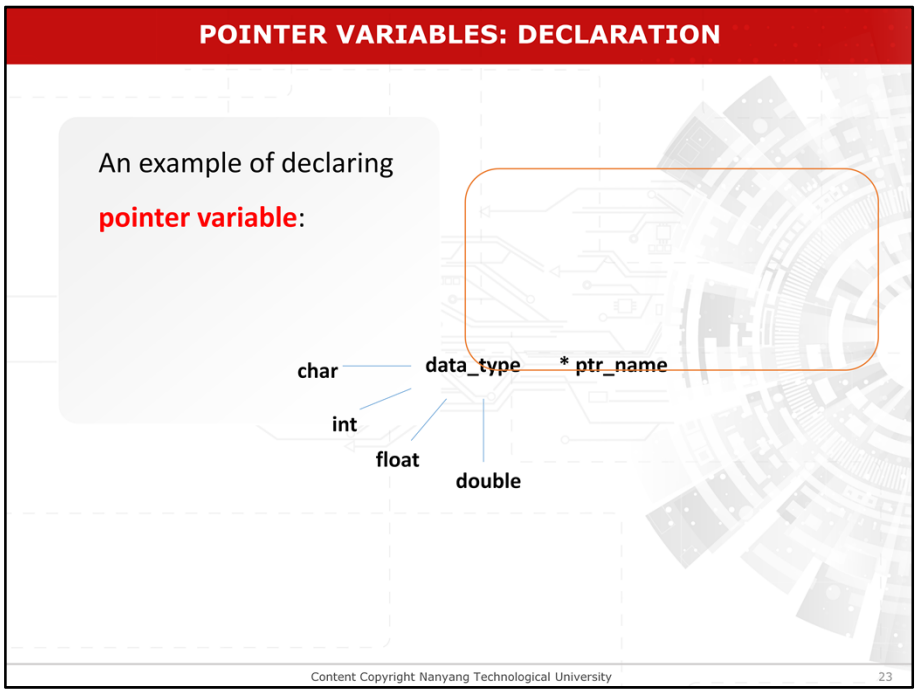
The **data_type** is used to indicate the type of data that the variable is pointed to.



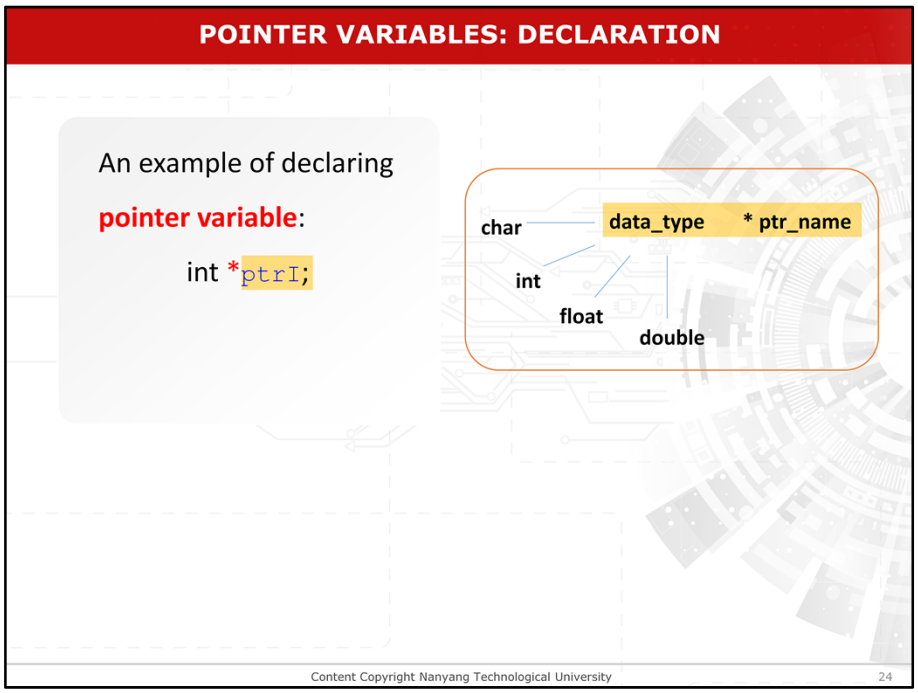
Pointer_name is the name of the pointer variable.



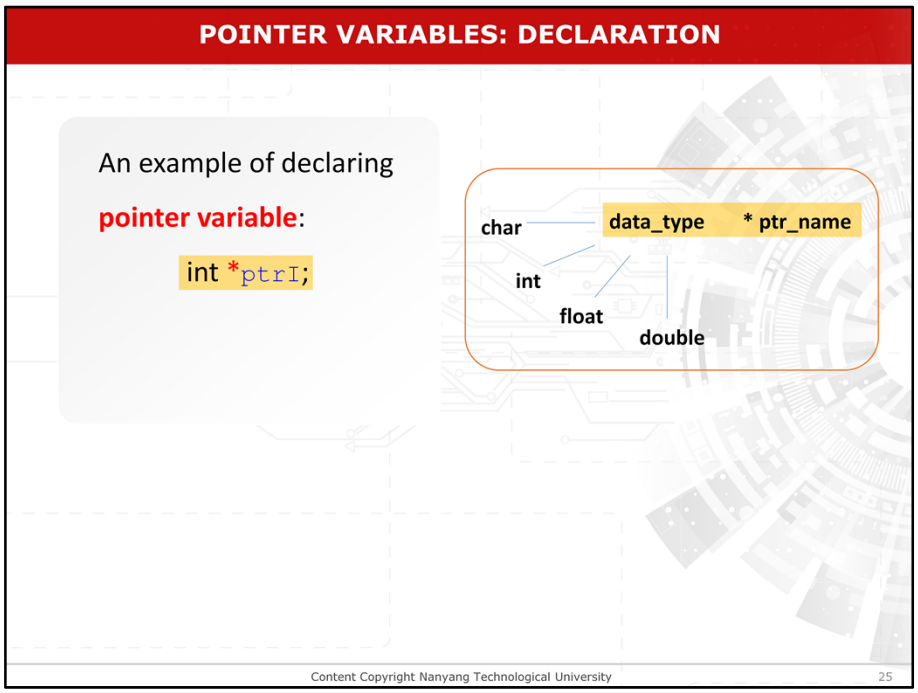
An asterisk (*) is used to indicate that the variable is a pointer variable.



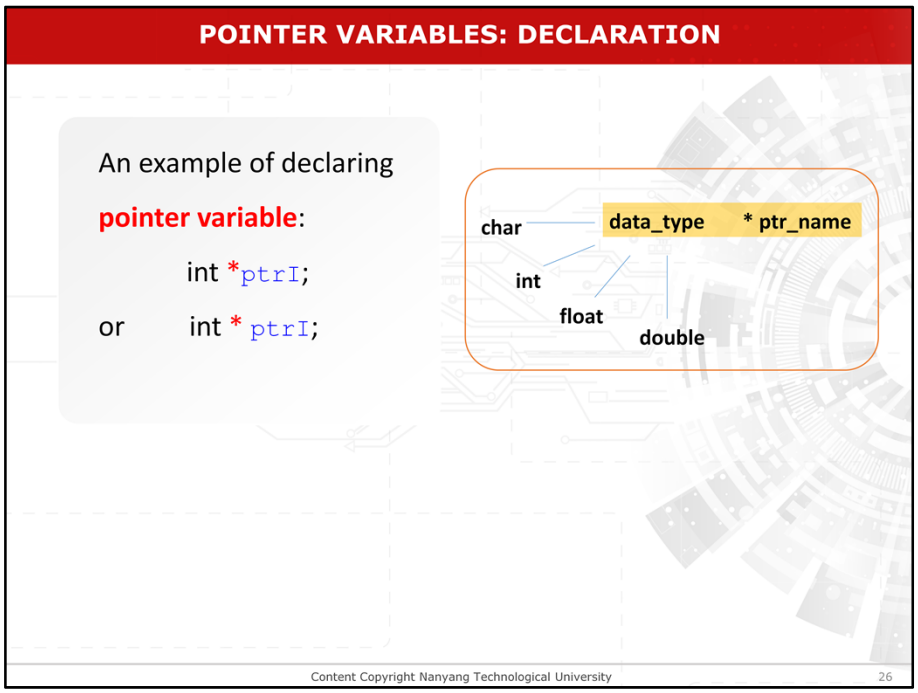
Let's take a look at an example of declaring pointer variable.



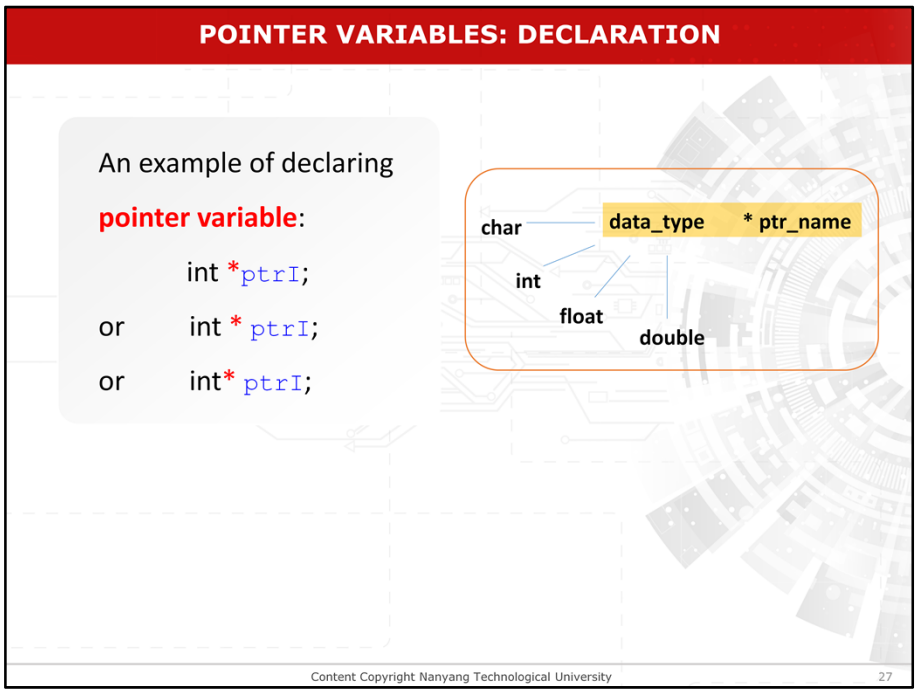
If we want to name the pointer variable as pointer I, we can use the short form p t r I to represent.



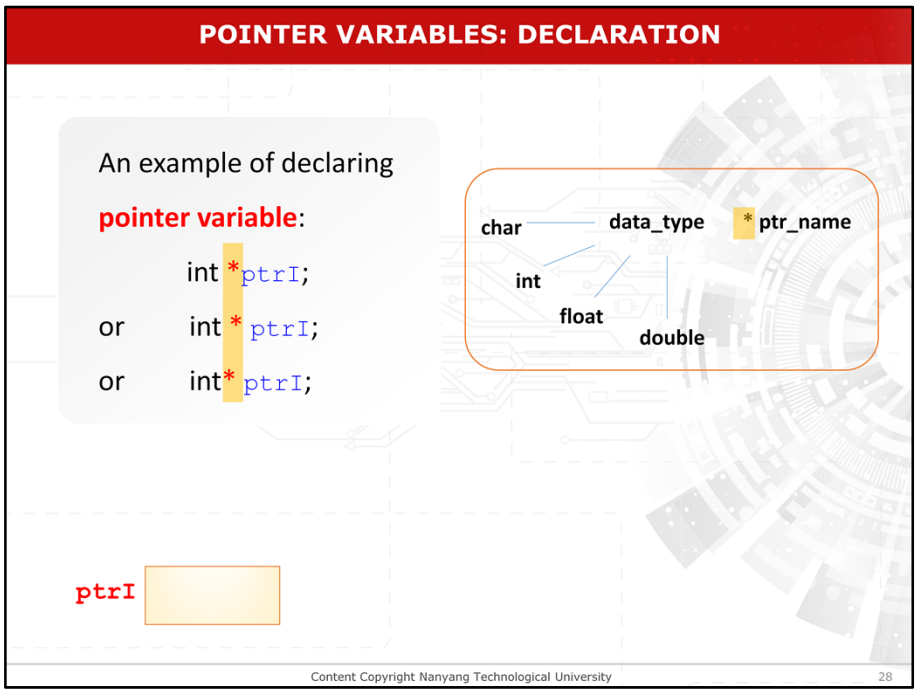
So the declaration statement would be
integer asterisk **pointer i**
Which can be also be written in the following ways as well.



[pause for 1 sec]

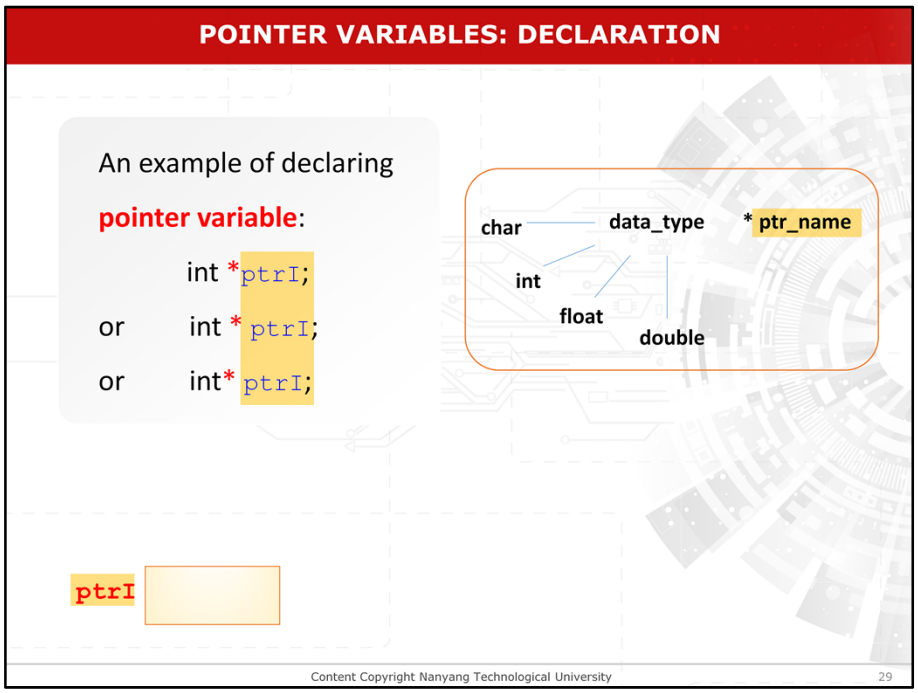


[pause for 1 sec]

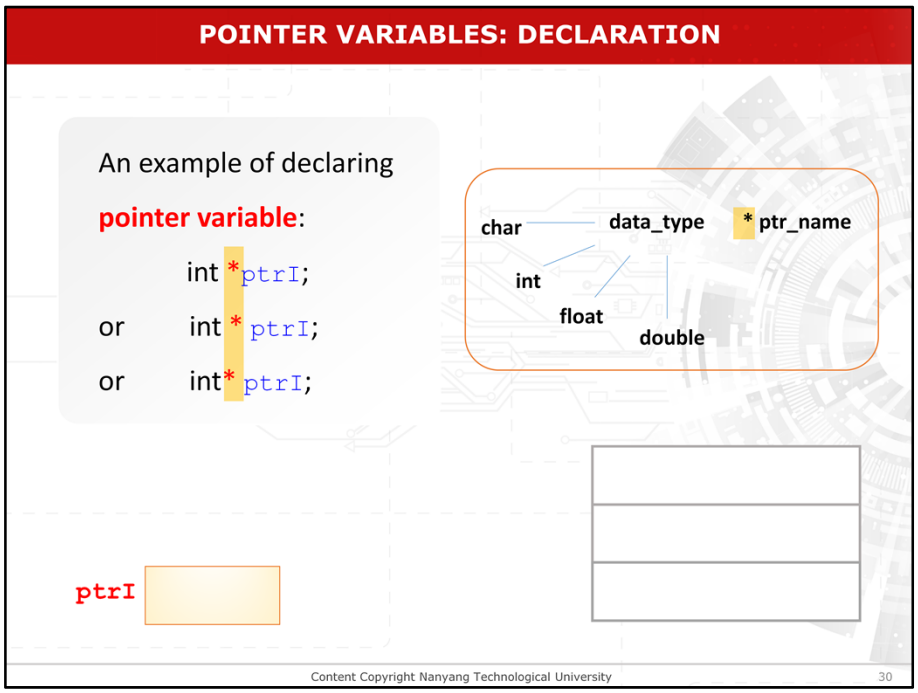


All these 3 statements are the same. The statement declare a pointer variable with the use of the asterisk sign.

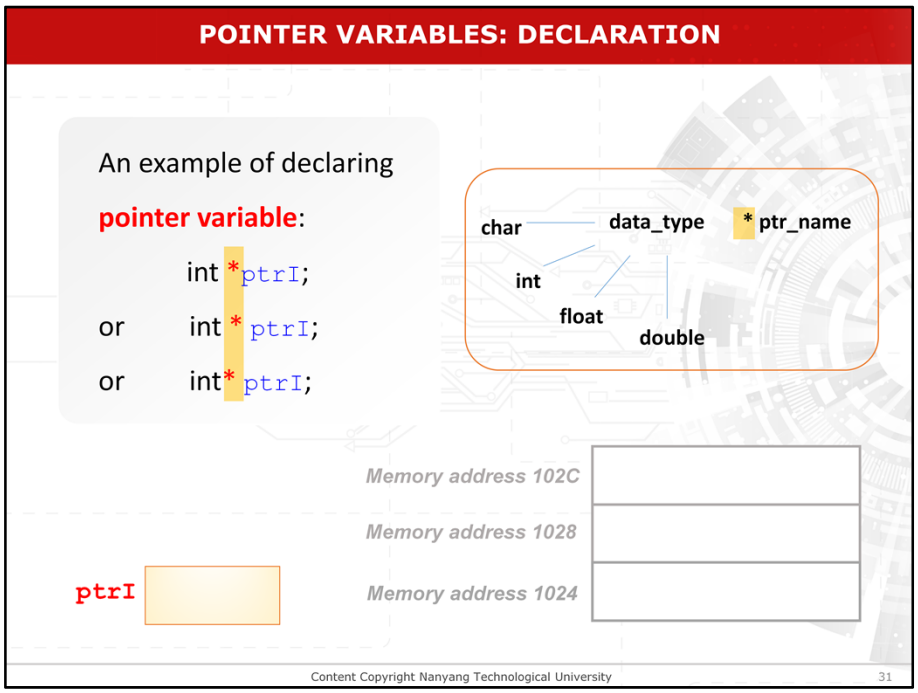
Notice that the whitespace in between the asterisk does not matter.



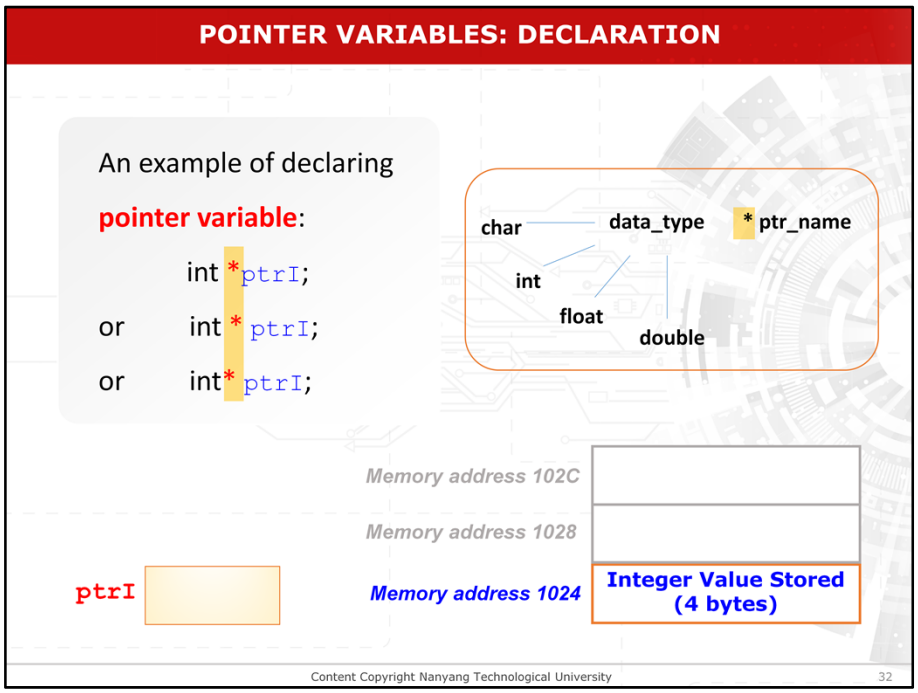
The pointer variable declared is named as pointer I in this example.



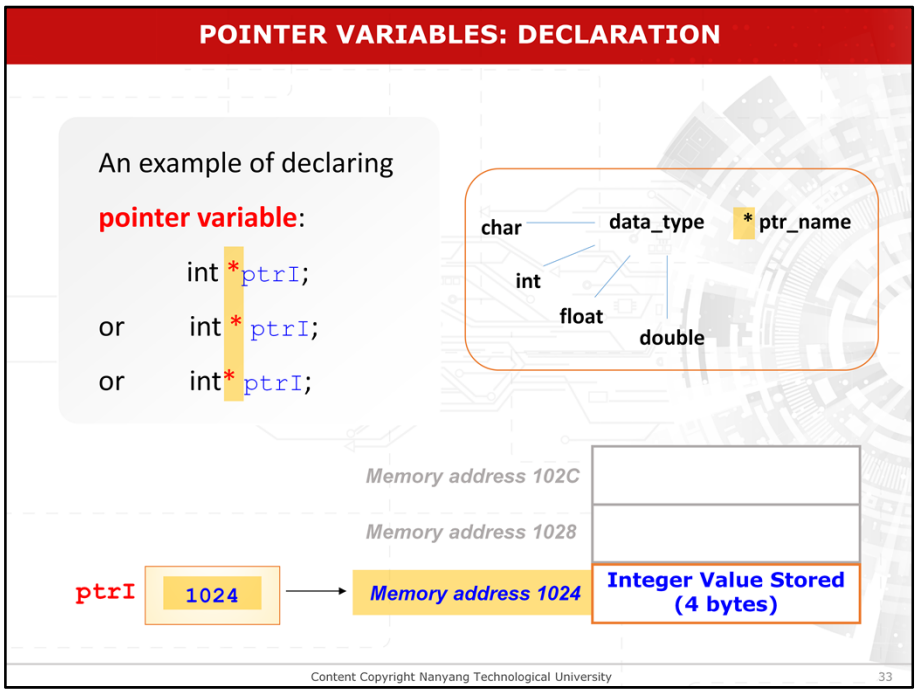
The pointer variable points to the address of a memory location.
The C P U has many memory spaces represented by the boxes as shown.



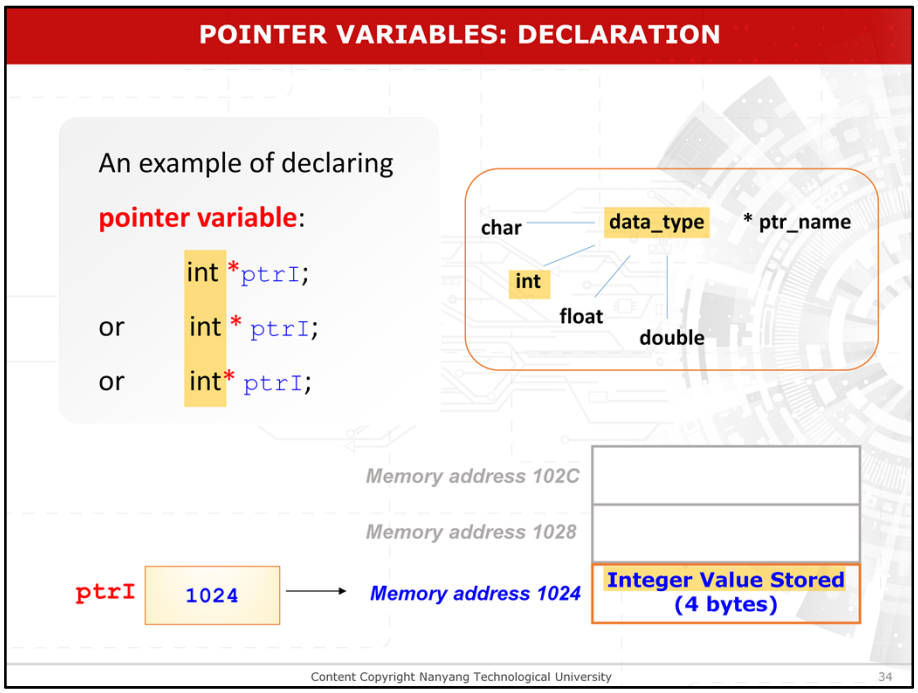
Each of these memory is unique and can be identified by memory address such as 1024, 1028, 102C etc.



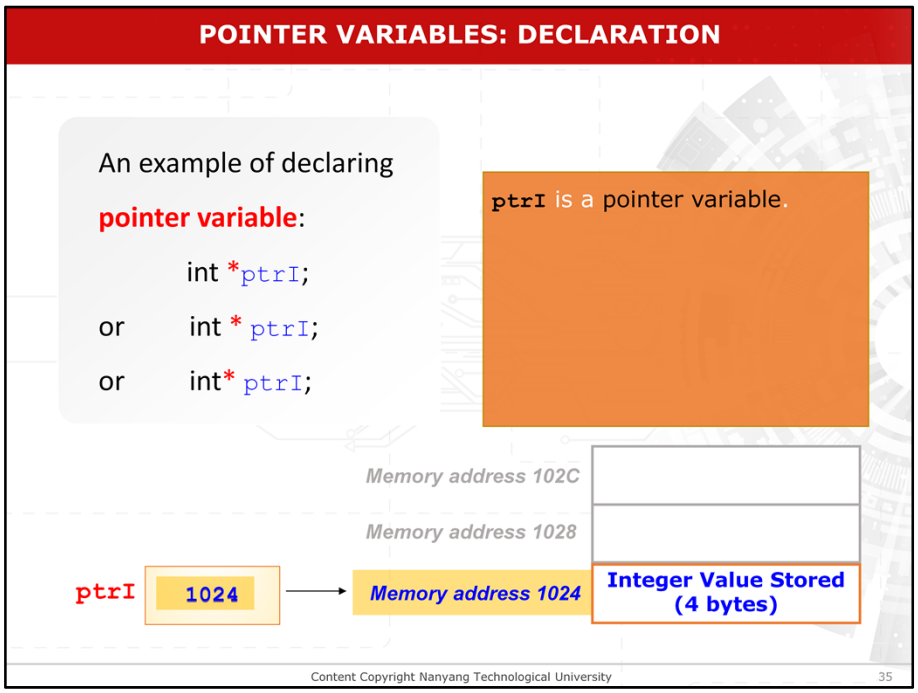
For this example, let's say the CPU has allocated memory address 1024 to store an integer value.
And the job for pointer I is to point to this memory location.



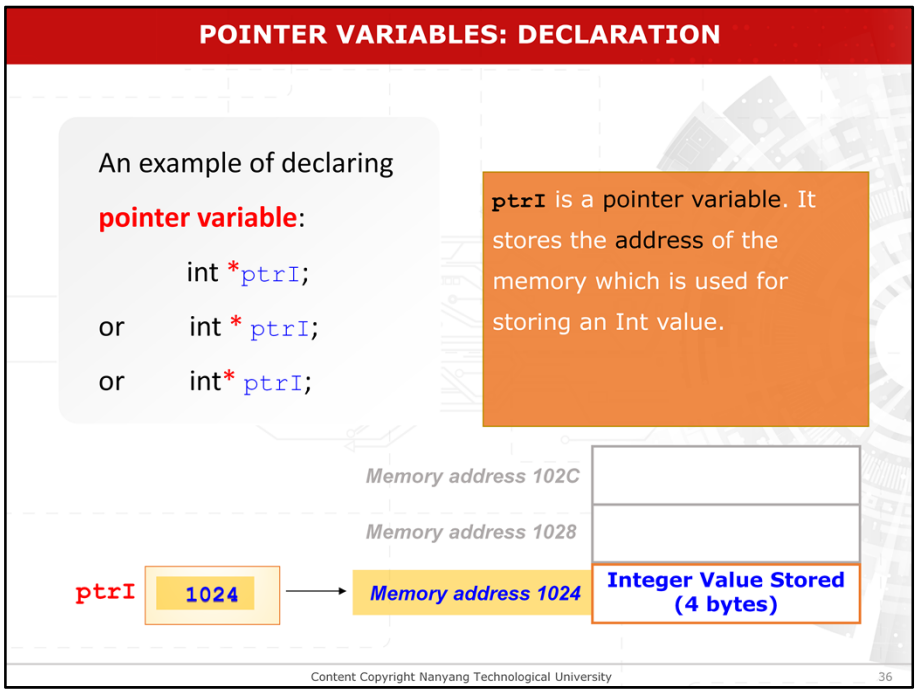
So the pointer I has the value 1024 stored in it.
It means that pointer I will point to the memory address of 1024 as shown in the diagram.



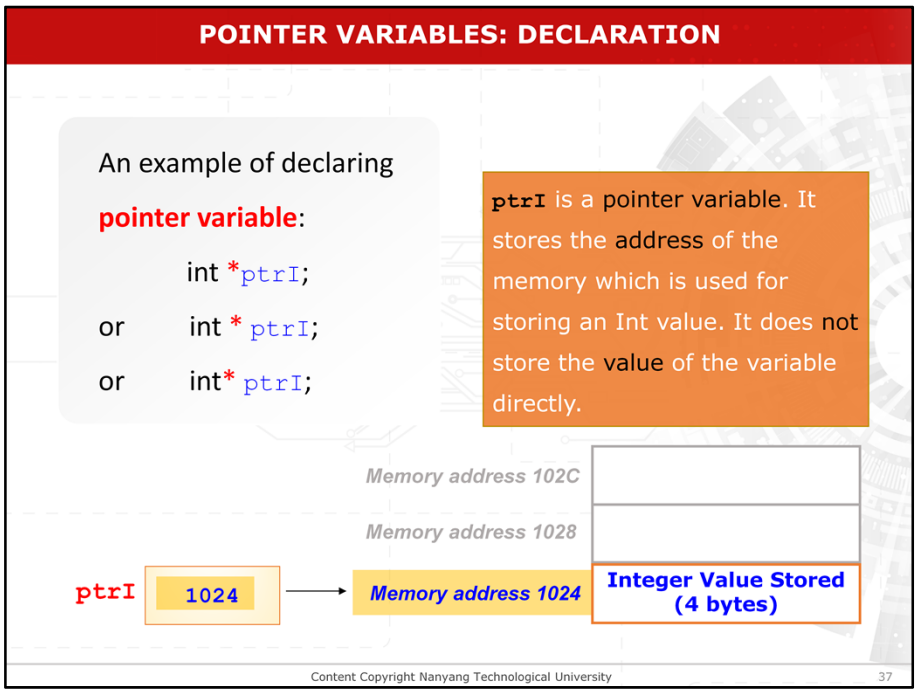
At the address that was pointed to, it is used to store an **int**eger. That is why the data type of the pointer I is integer



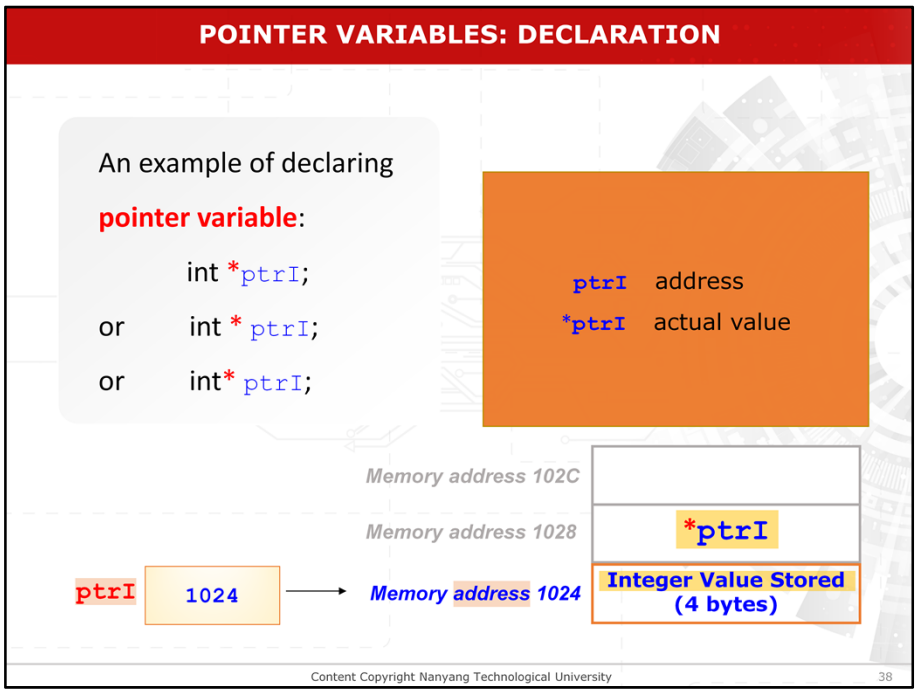
Note that pointer I is a pointer variable



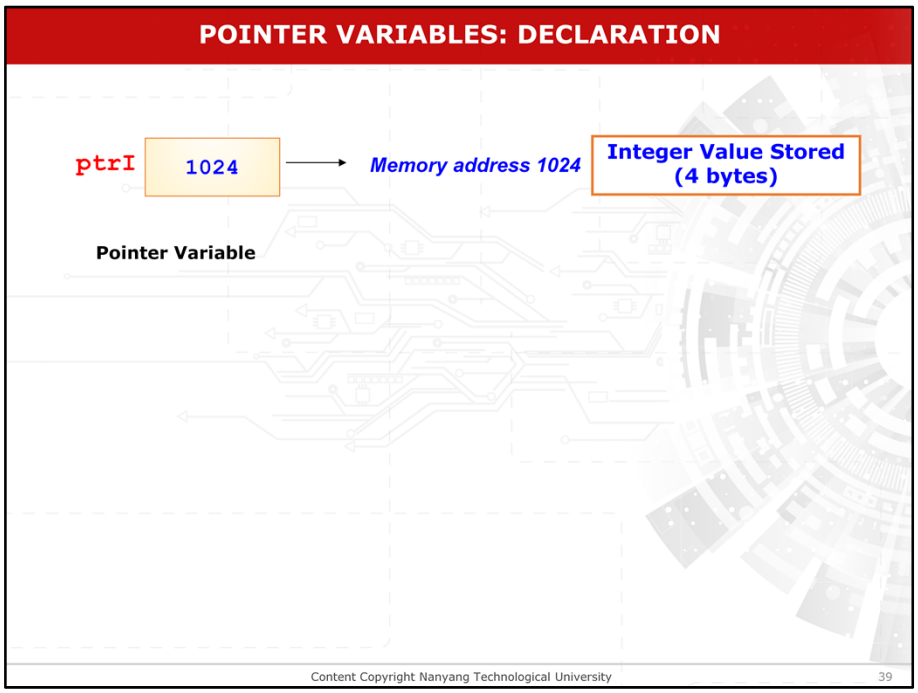
The value of a pointer variable is an ***address***, which is different from other variables of primitive data types that store the *data* directly.
In this example, it stores the **address** of the memory which is used for storing an integer value.



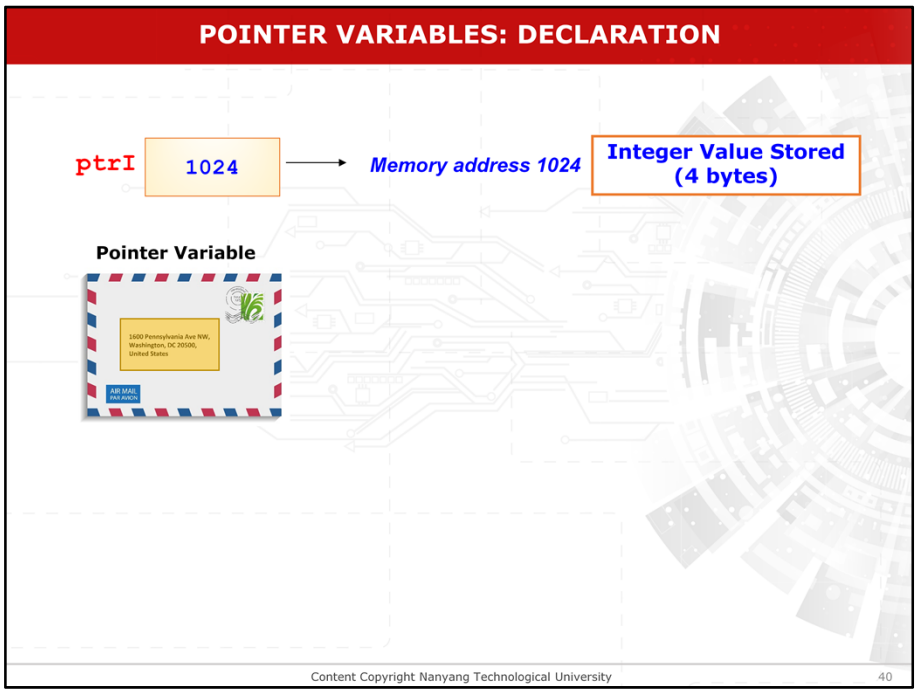
It does **not** store the **value** of the variable. It stores the address of the memory.



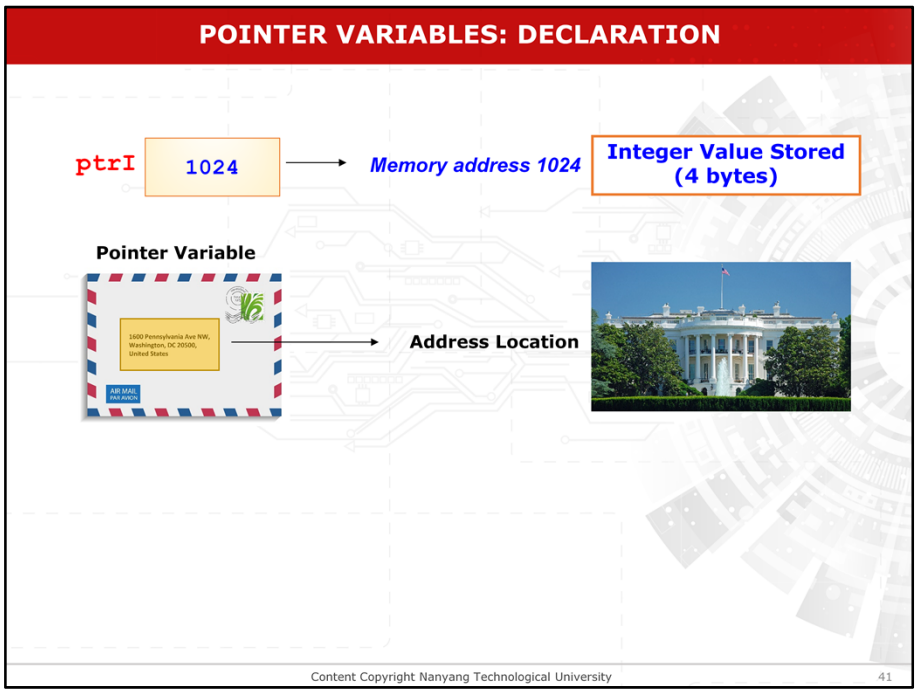
If we want to retrieve the actual value, we will need to use indirection operator (*), e.g. ***ptr i**.



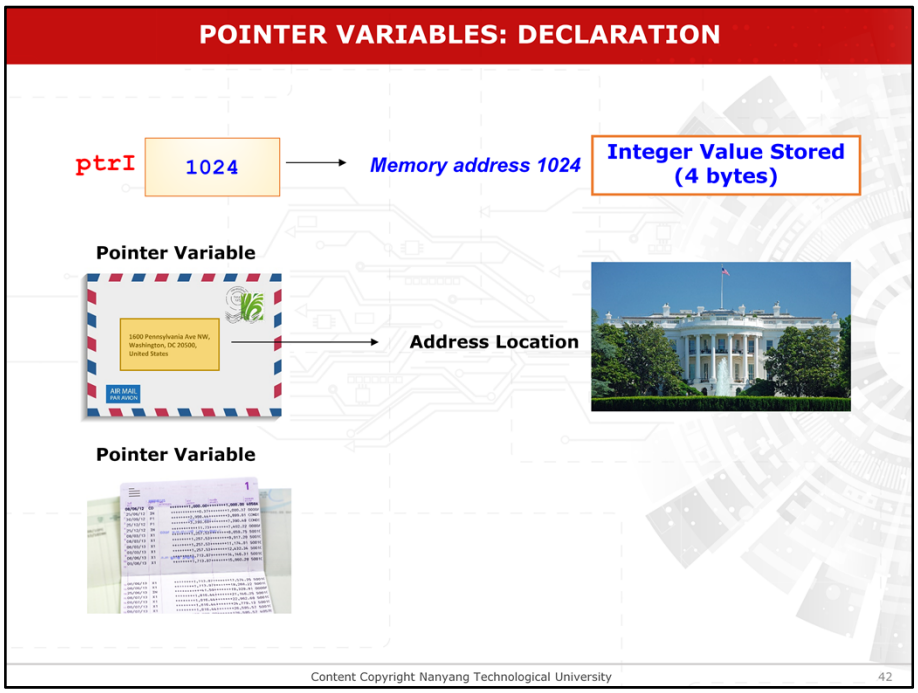
Pointer variable is similar to



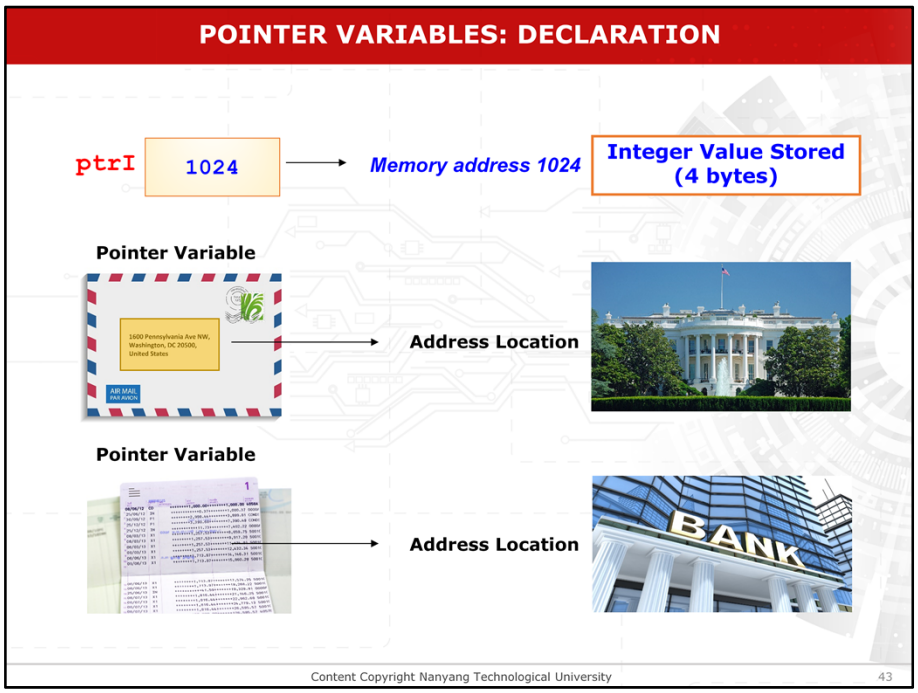
the address written on an envelope which stores the home address



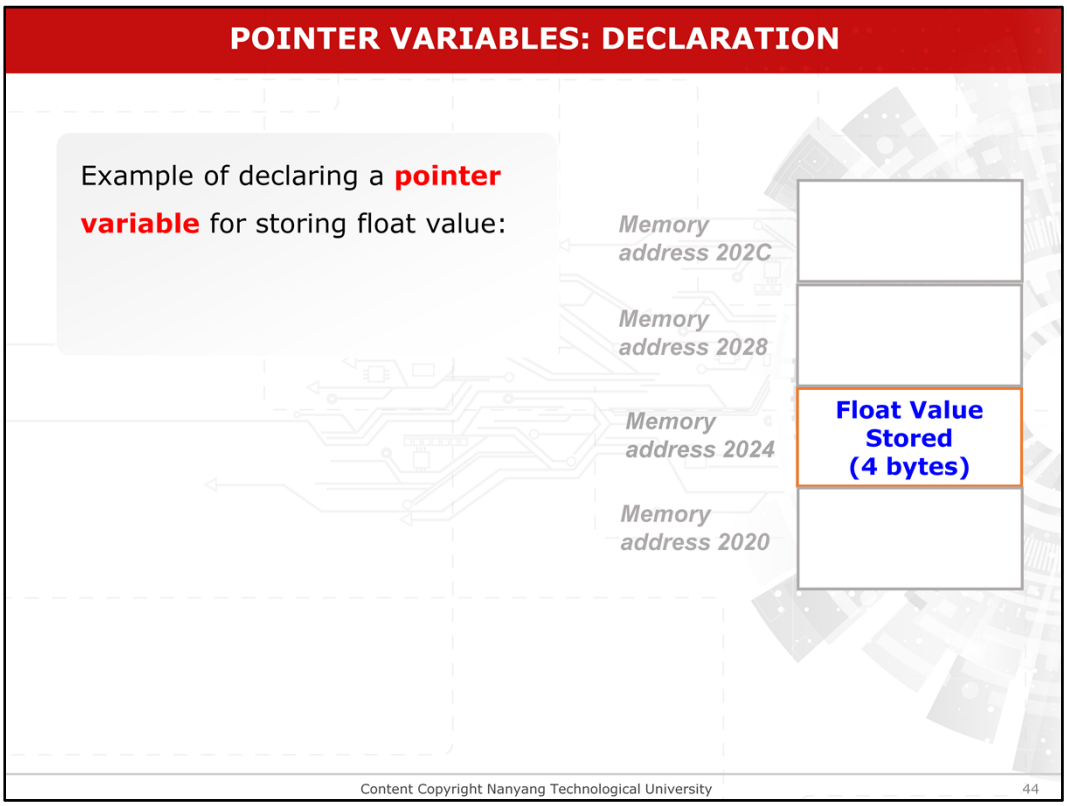
and the actual place can be referred to by the address



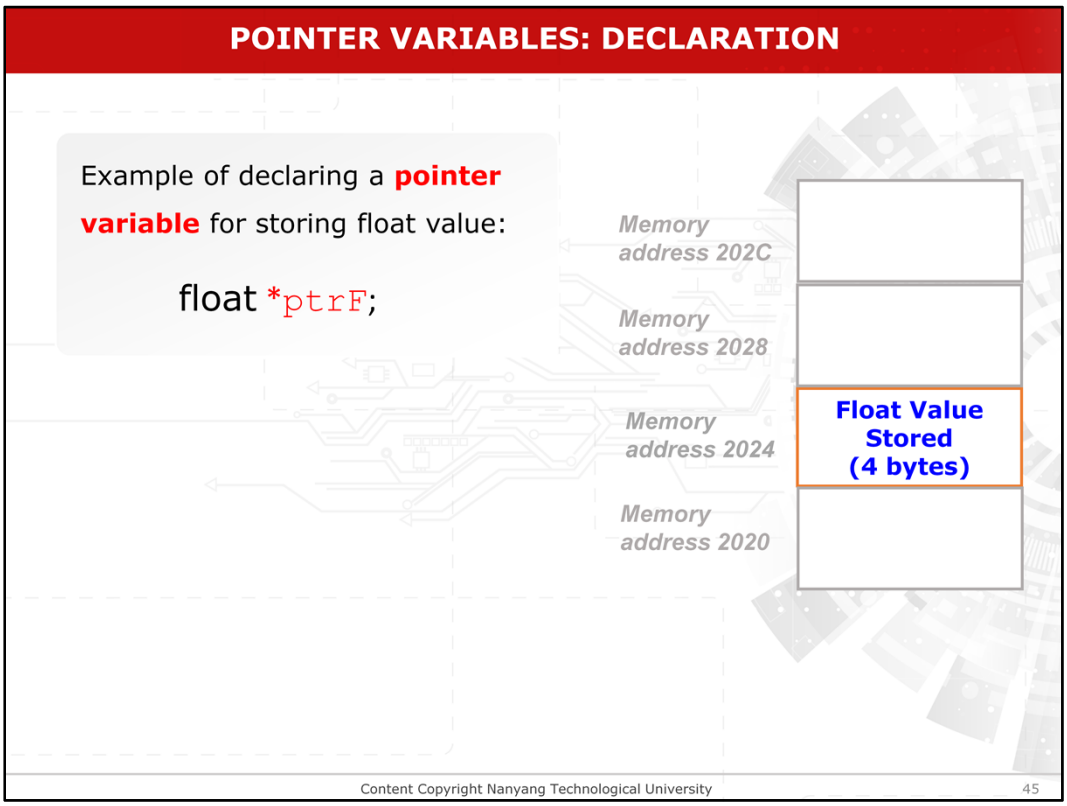
Pointer variable is also similar to a bank account, which contains the saving information



and the bank location that stores the real saving money. The saving money can be referred to via the bank account.

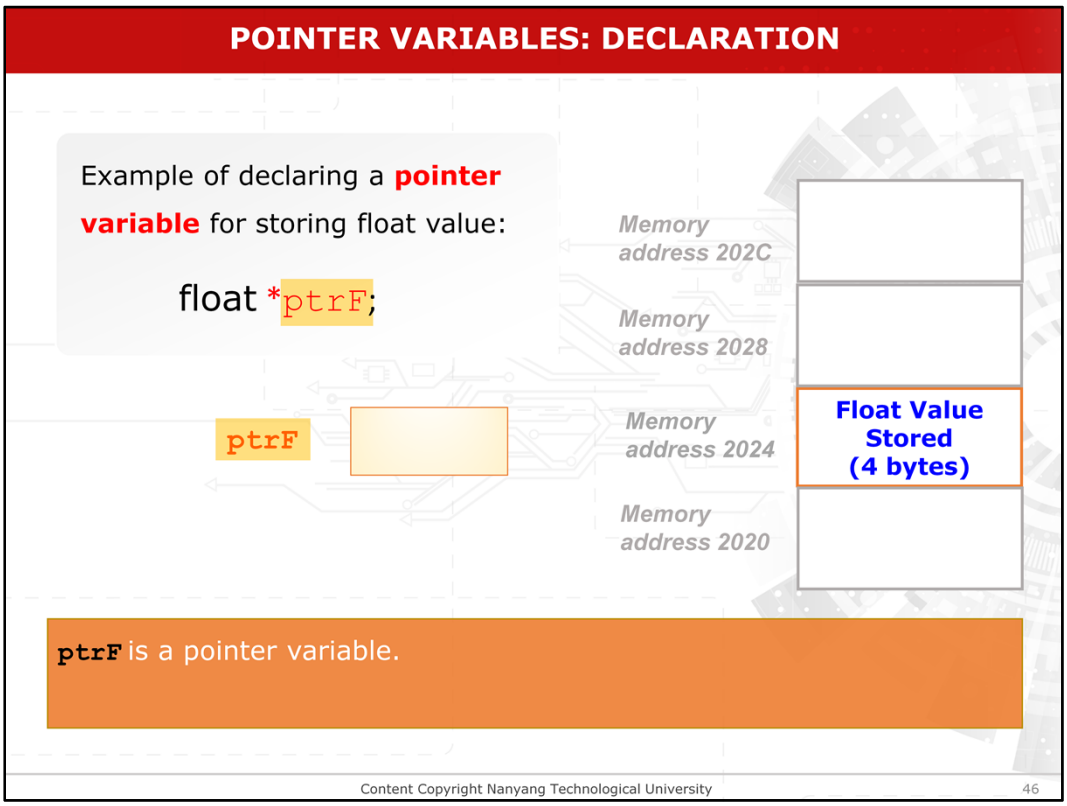


Let's see an example of declaring a **pointer variable** for storing float value.

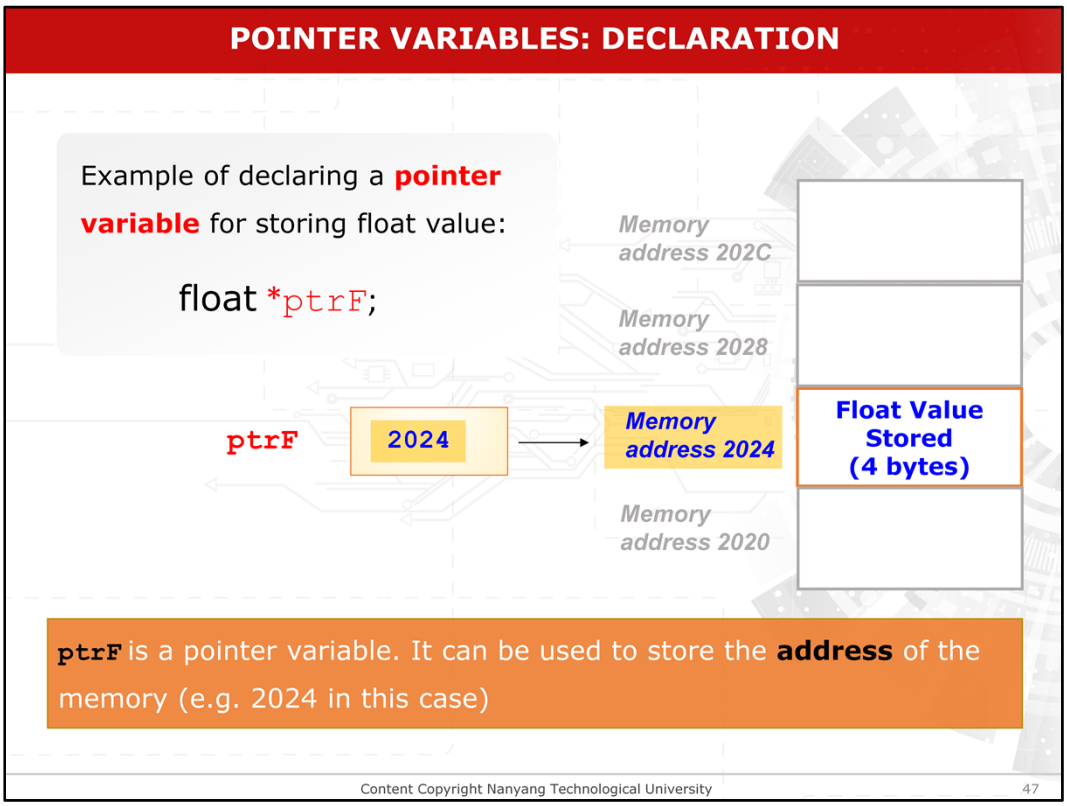


If we want to name the pointer variable as pointer F which we will use the short form p t r F, the statement of declaration would be

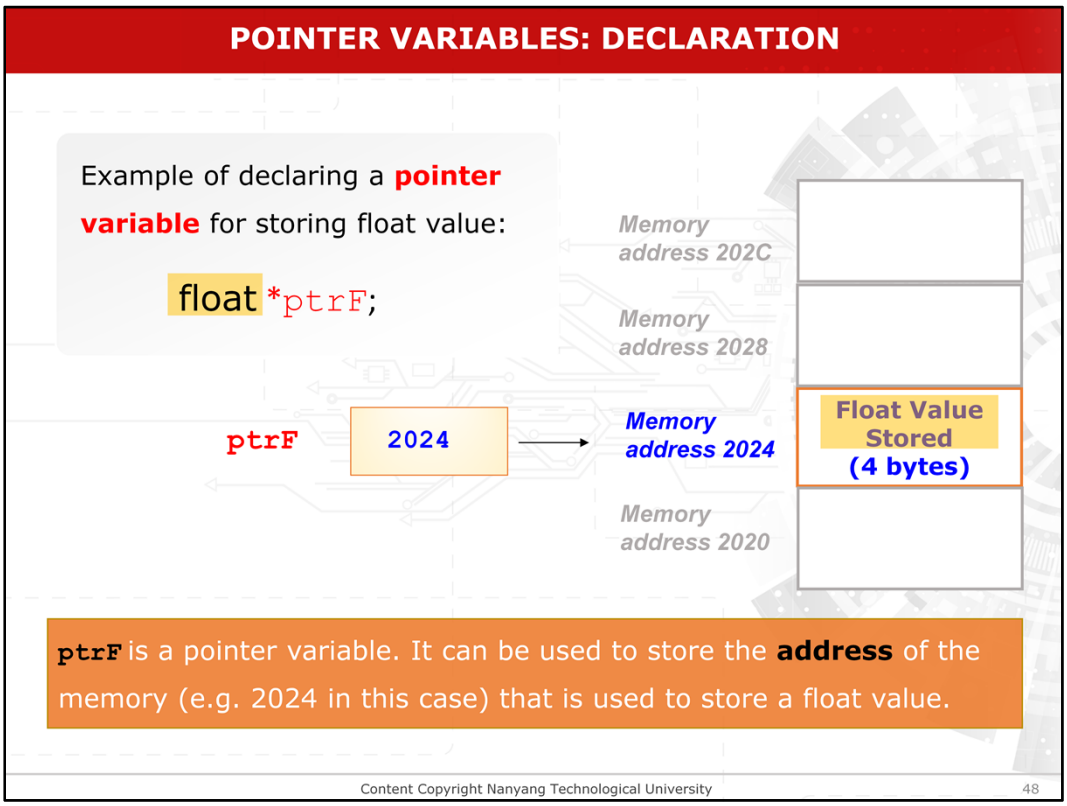
Float asterisk pointer F



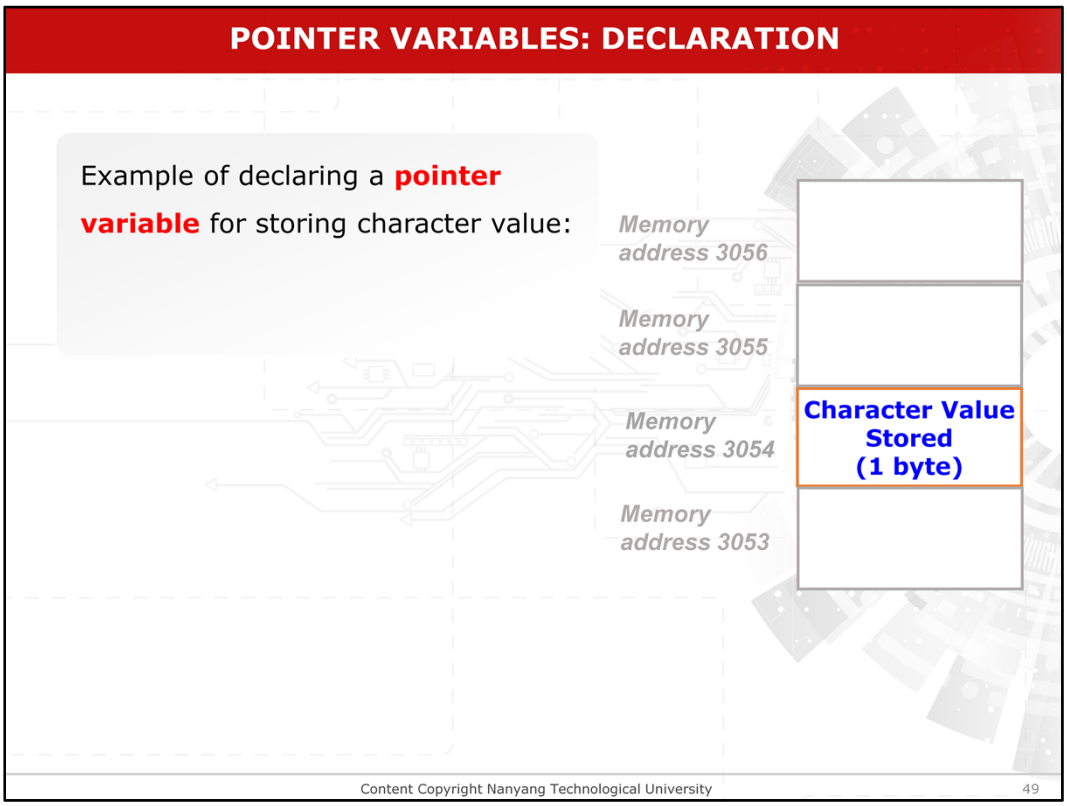
Pointer F is a pointer variable.



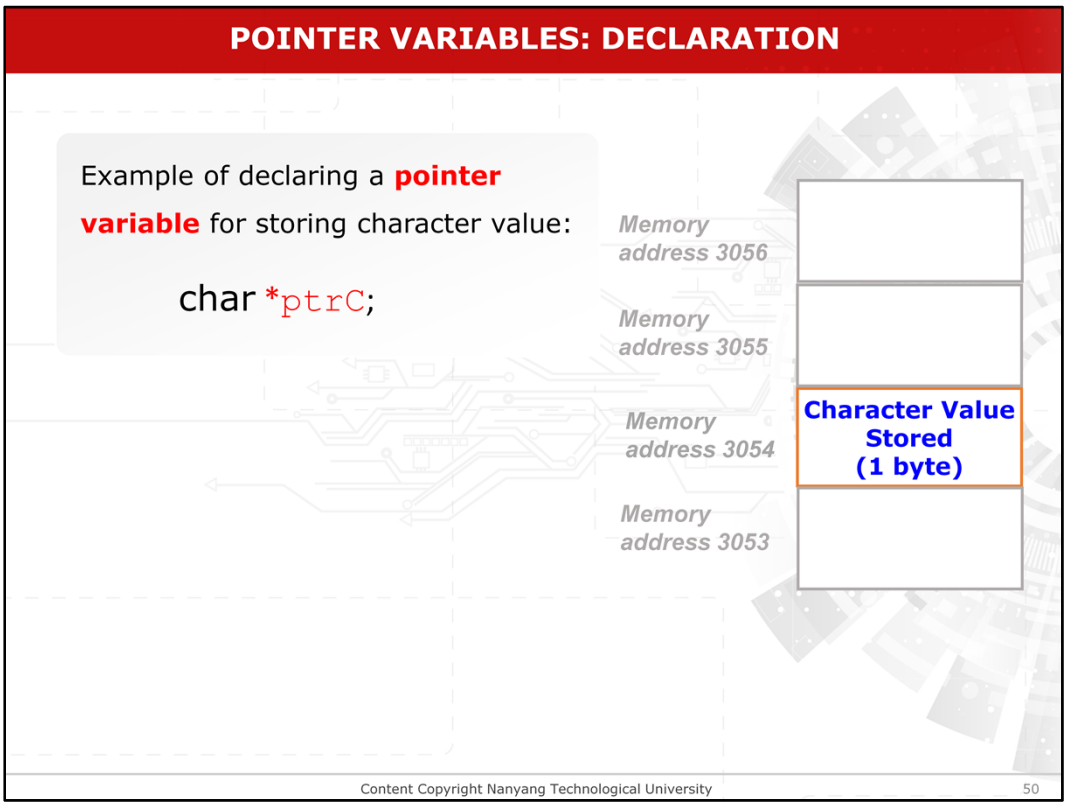
It can be used to store the address of the memory (for example 2024 in this case)



that is used to store a **float**.

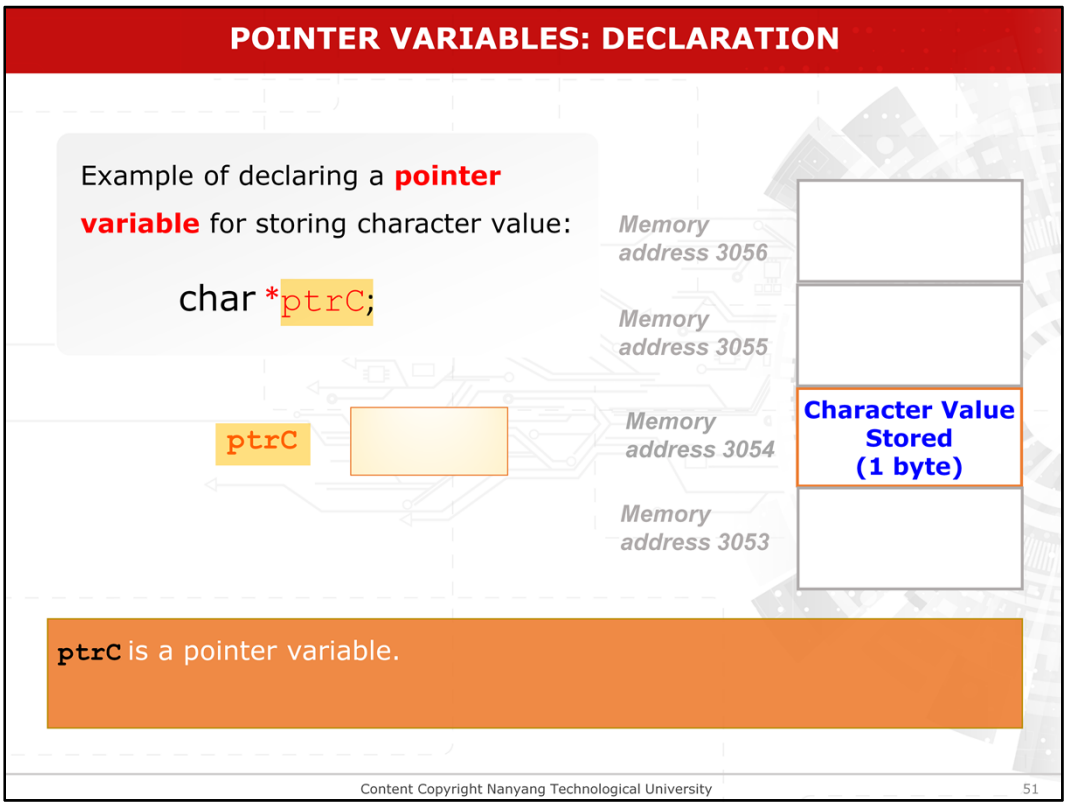


Let's see an example of declaring a **pointer variable** for storing character value.

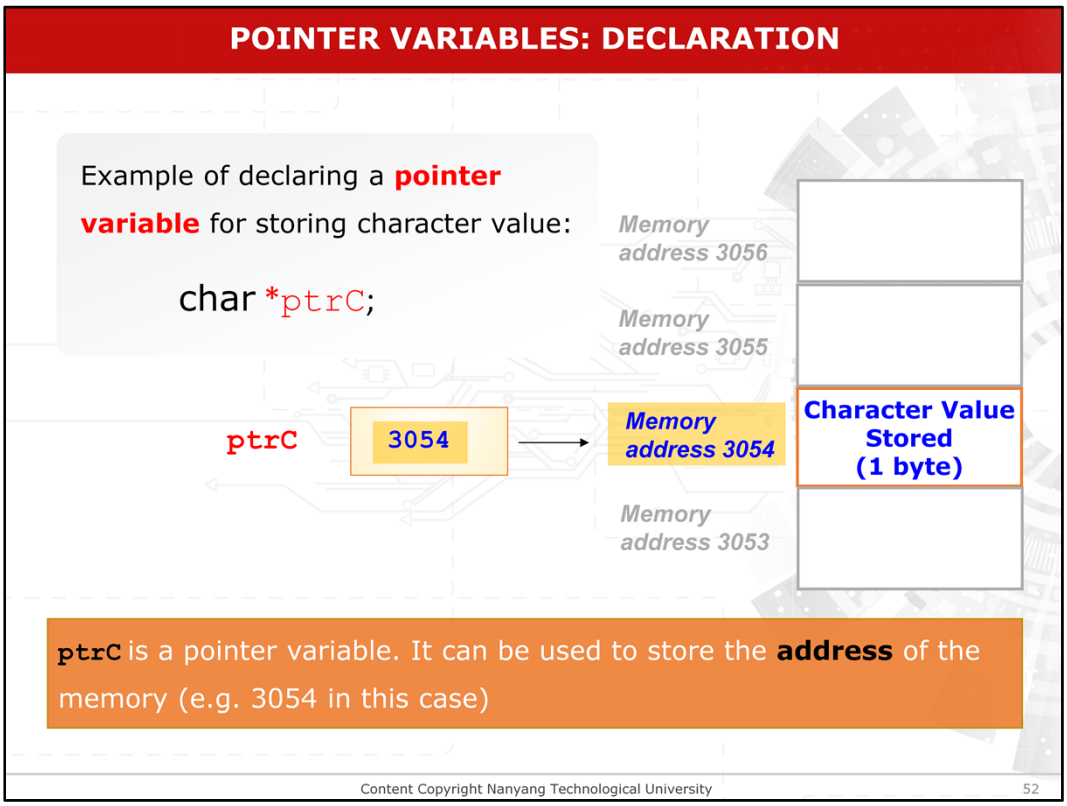


If we want to name the pointer variable as pointer C which we will use the short form p t r C, the statement of declaration would be

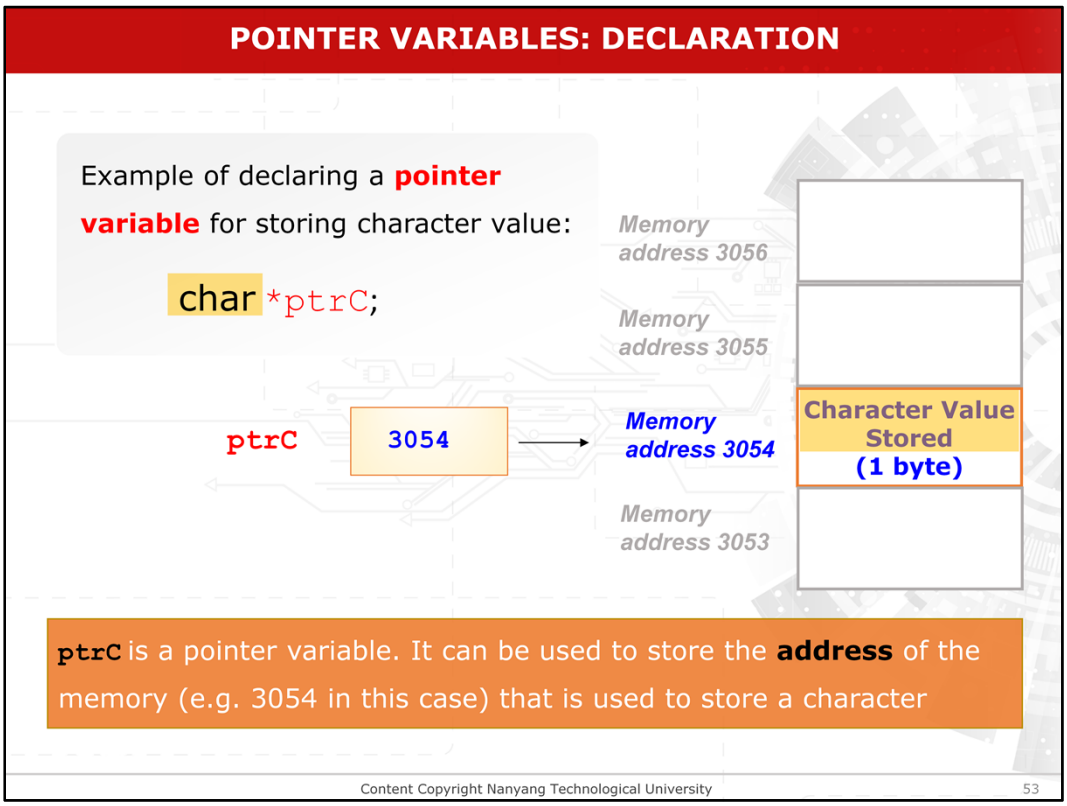
Character asterisk pointer C



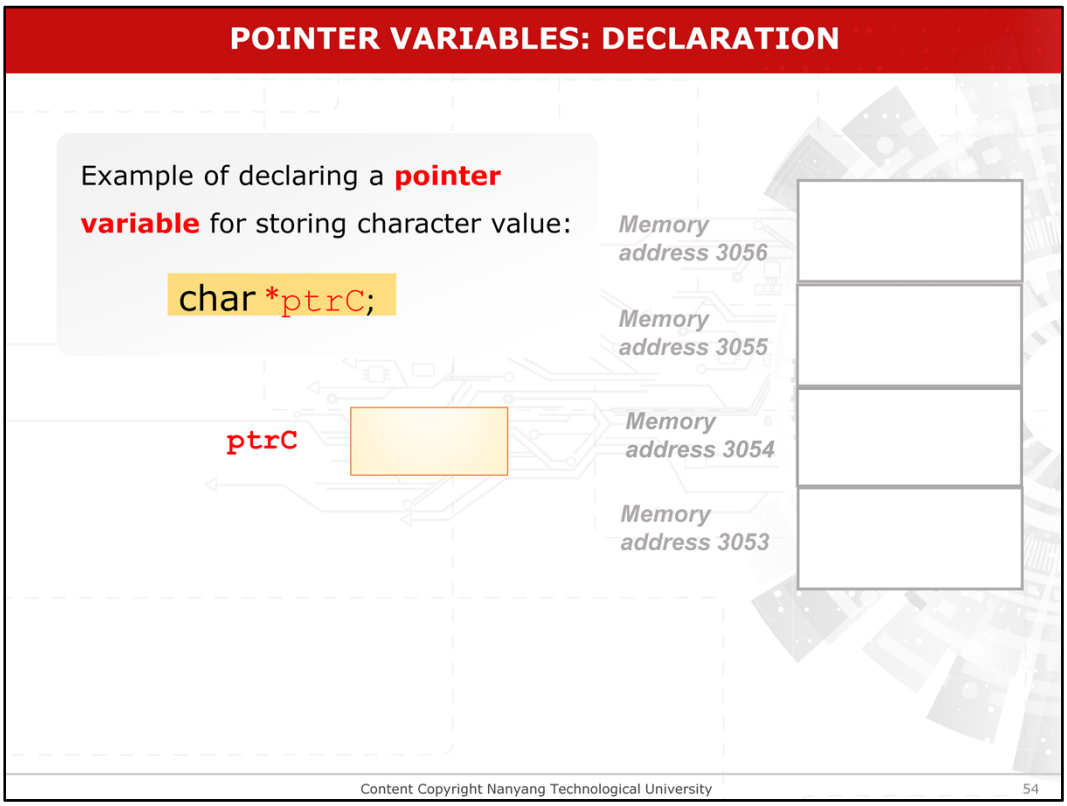
Pointer C is a pointer variable.



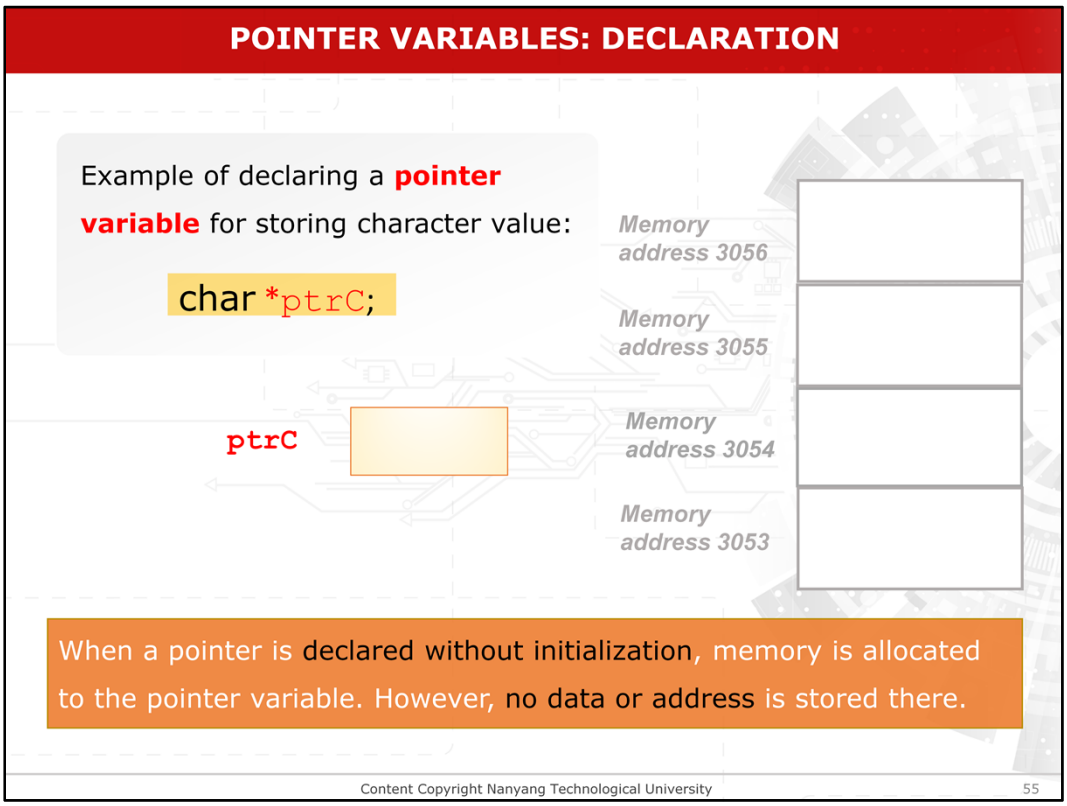
It can be used to store the address of the memory (for example 3054 in this case)



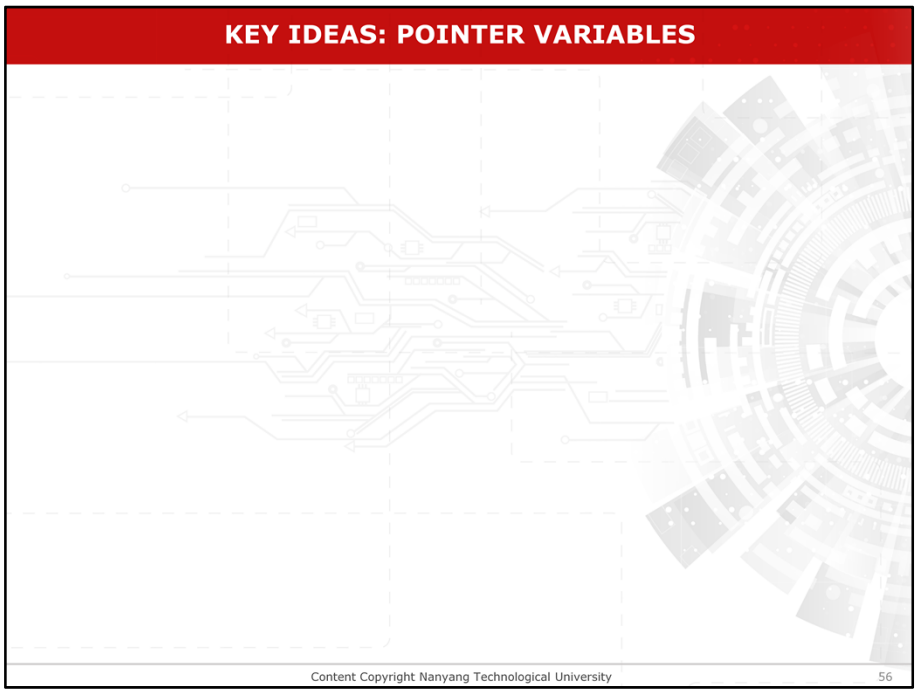
that is used to store a **character**.



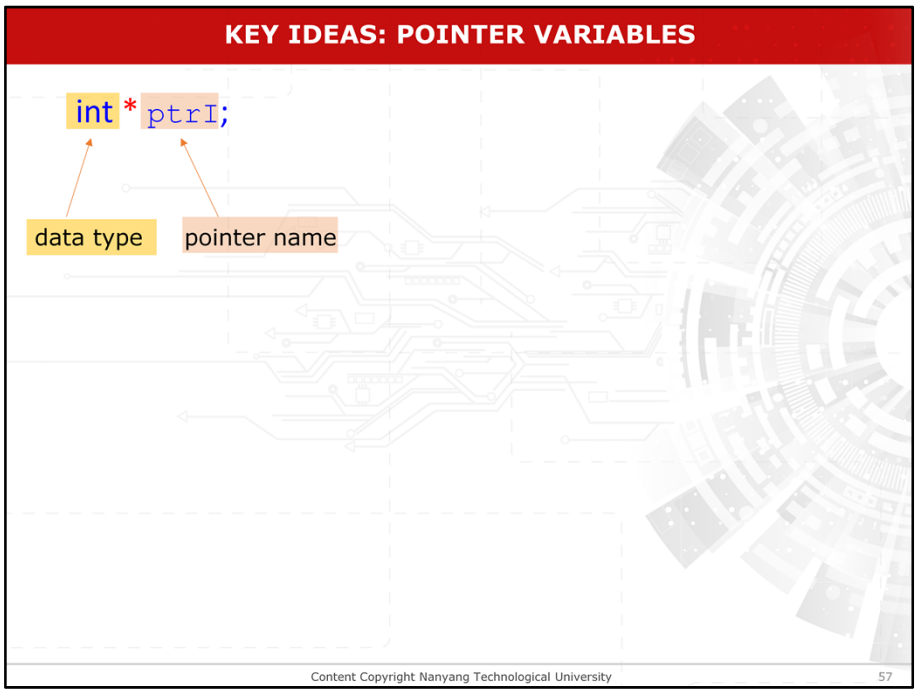
We need to note that in the statement Character asterisk pointer C, the pointer C is declared without initialization, that is no value has been assigned to it yet.



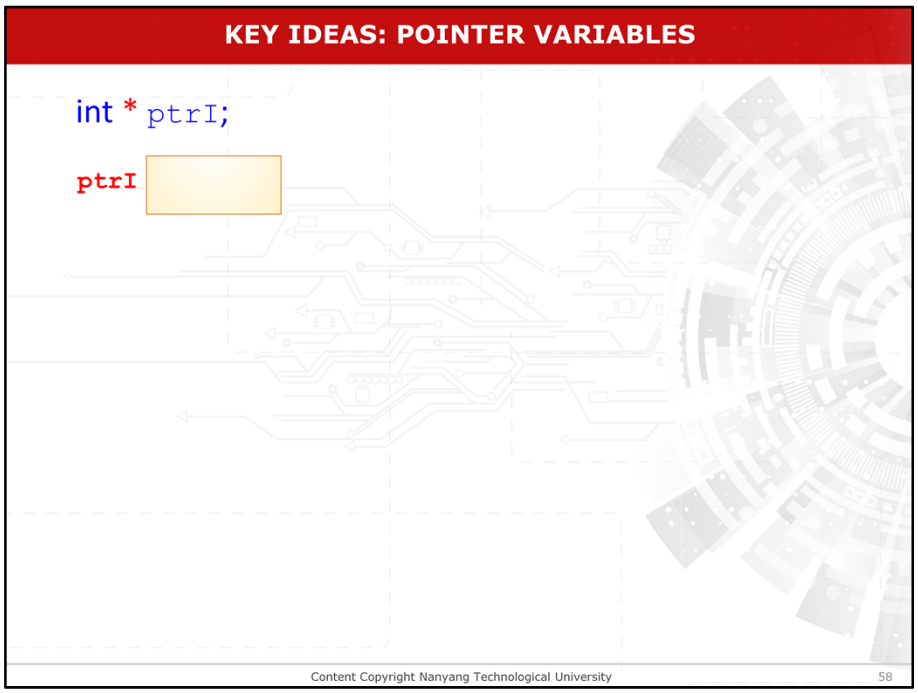
When a pointer is declared without initialization, memory is allocated to the pointer variable. However, no data or address is stored there yet.



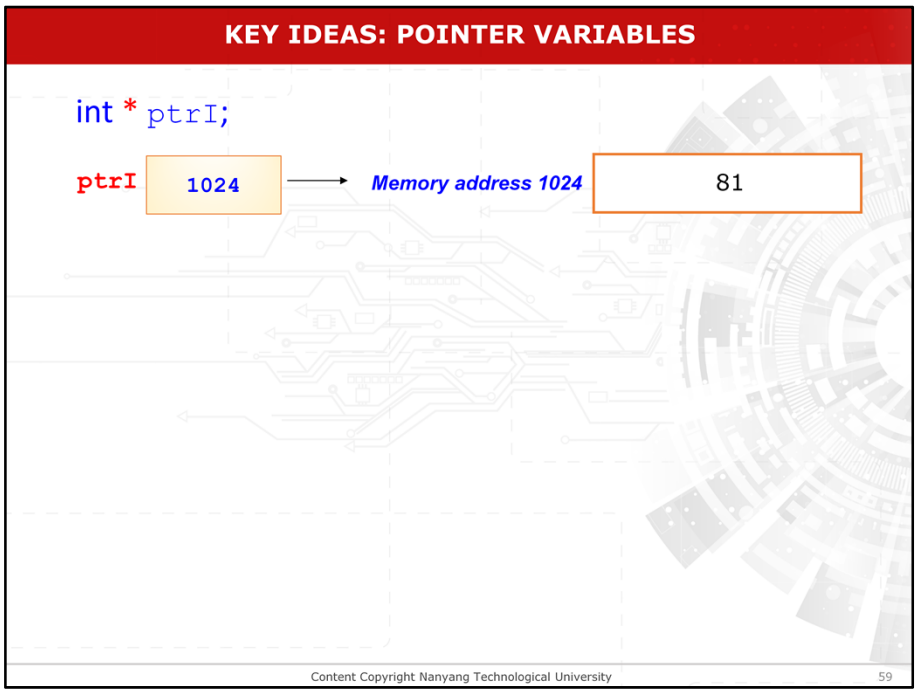
Let us recap the key ideas.



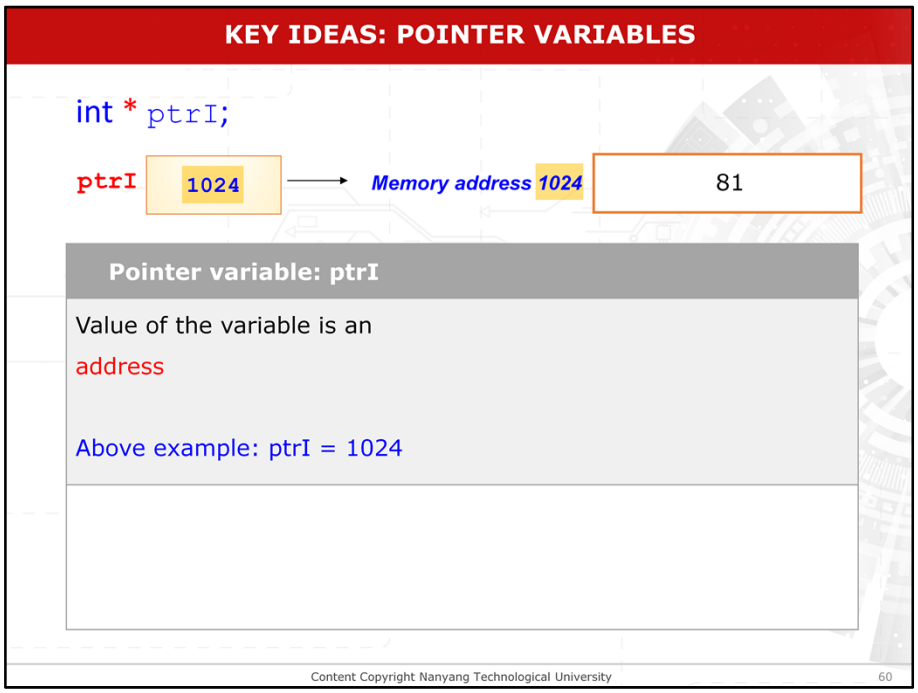
Pointer variable is declared with the format data type asterisk pointer name.



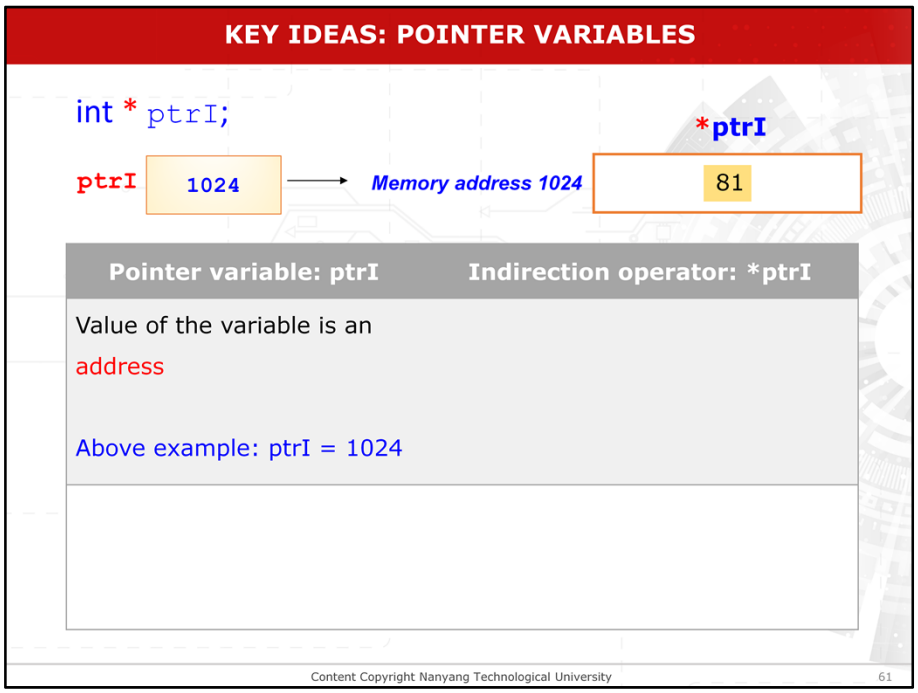
For example, integer asterisk pointer I. After this statement is declared, the CPU will allocate memory for pointer i.



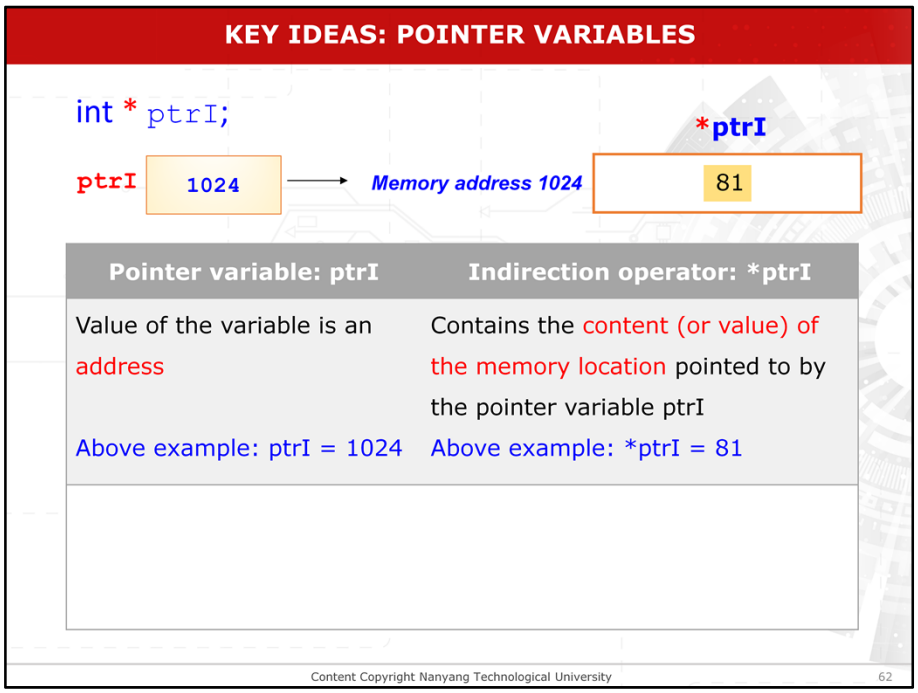
Pointer variable stores an address which refers to the location that stores the actual data.



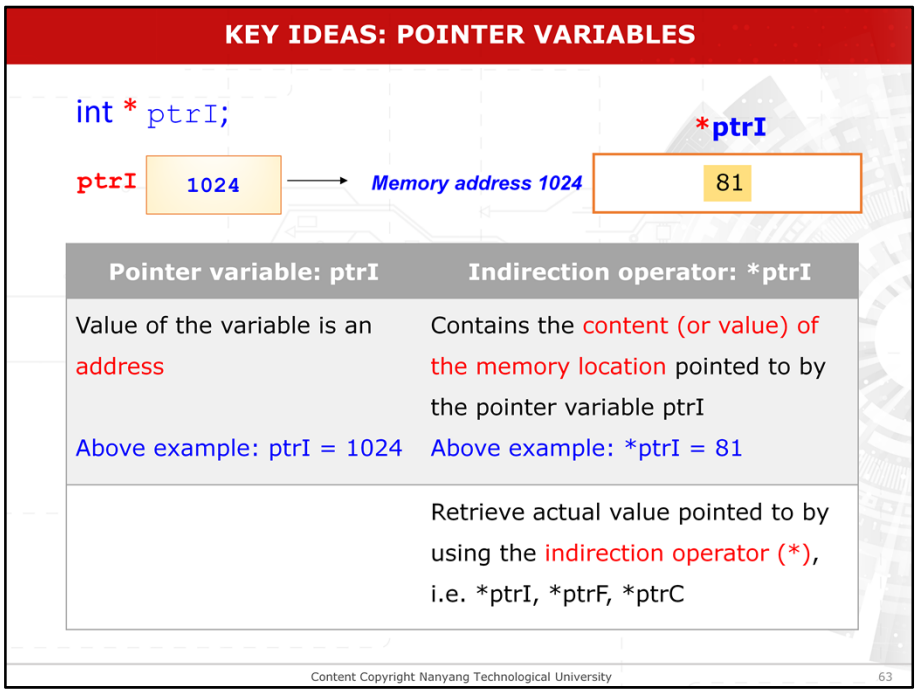
So pointer I is a pointer variable and the value of the pointer variable is an address. In this example, pointer I equals 1024.



To retrieve the value stored in the pointed memory address, we use indirection operator, for example, asterisk pointer i.



So asterisk pointer contains the content (or value) of the memory location pointed to by the pointer variable pointer i. In this case, the value of asterisk pointer I equals 81

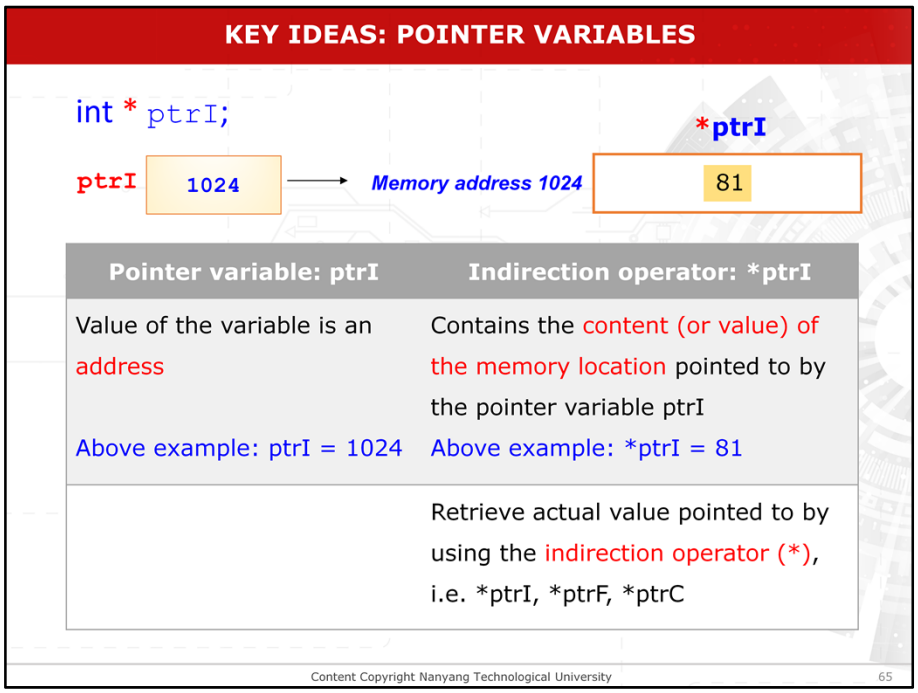


We can retrieve actual value pointed to by using the indirection operator asterisk that is asterisk pointer I, asterisk pointer F and asterisk pointer C and so on.

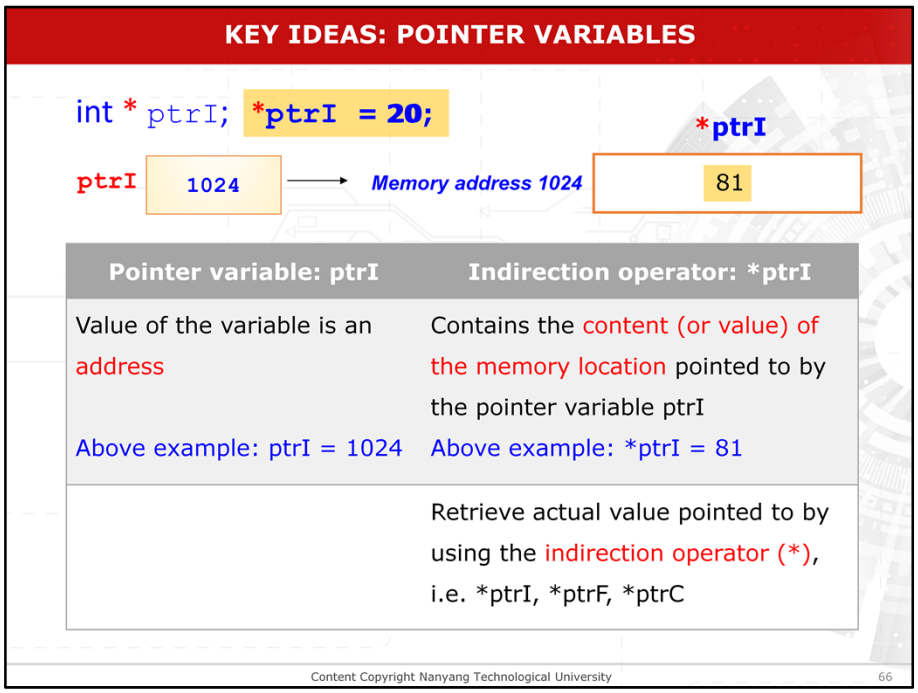
KEY IDEAS: POINTER VARIABLES

Pointer variable: ptrI	Indirection operator: *ptrI
Value of the variable is an address Above example: ptrI = 1024	Contains the content (or value) of the memory location pointed to by the pointer variable ptrI Above example: *ptrI = 81
	Retrieve actual value pointed to by using the indirection operator (*), i.e. *ptrI, *ptrF, *ptrC

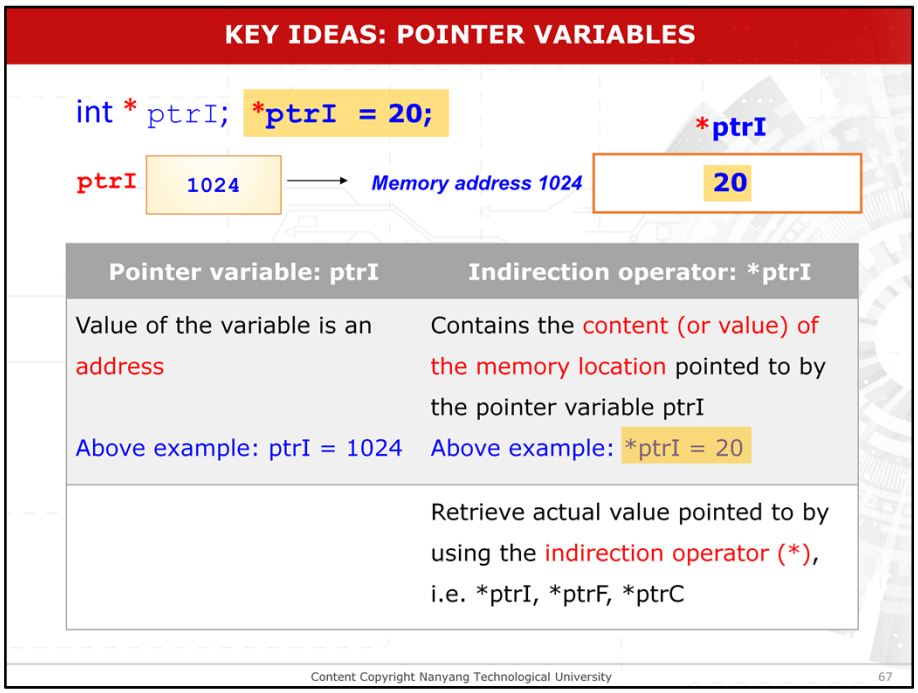
What if we want to change the value stored in the pointed memory?



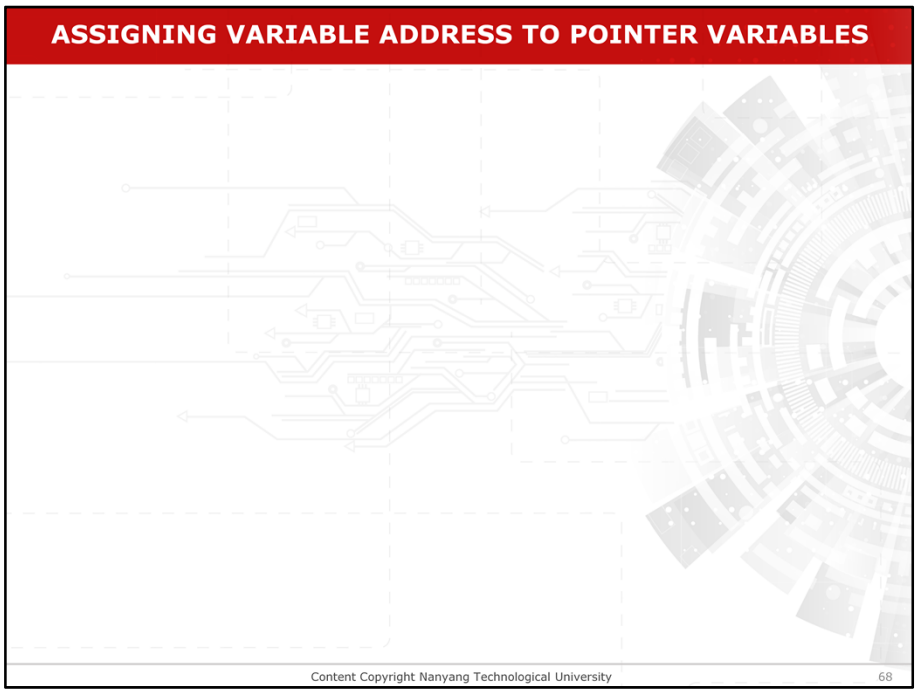
We can make use of indirection operator as well.



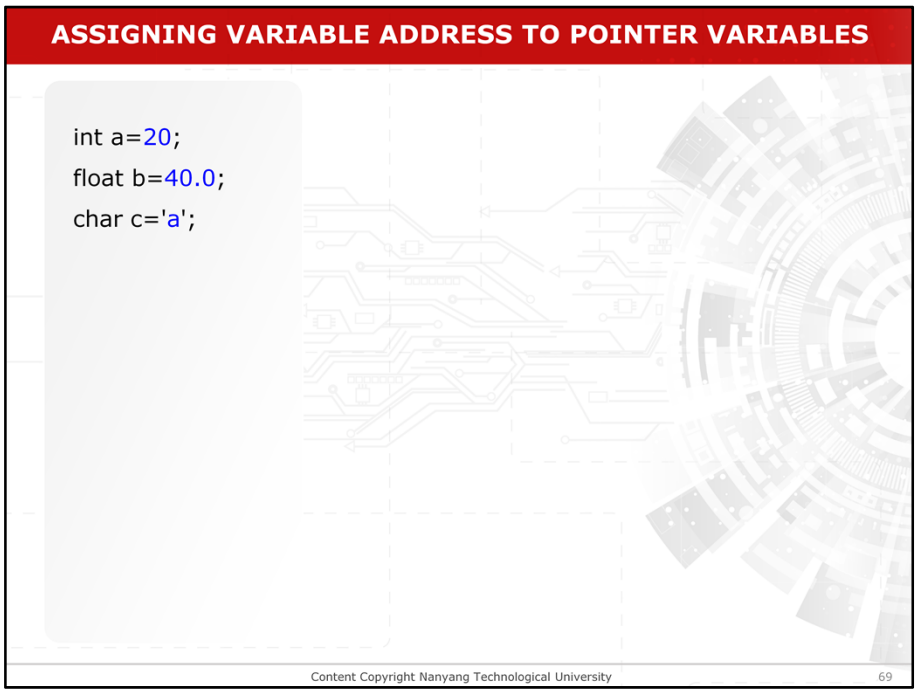
In this example, the statement asterisk pointer I equal 20 will update the value 81 to 20.



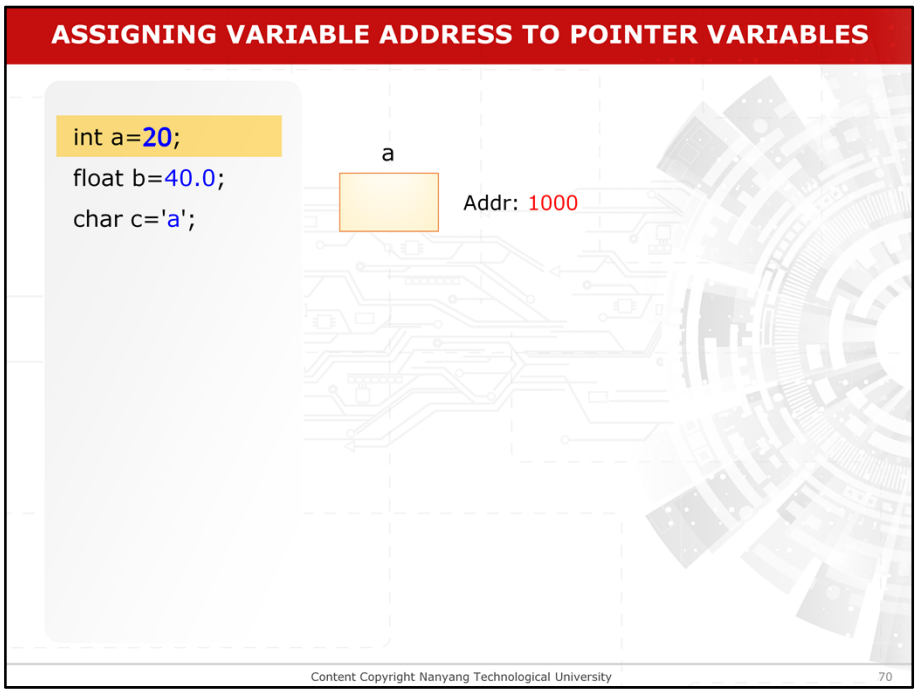
Therefore, asterisk pointer I equals 20 now.



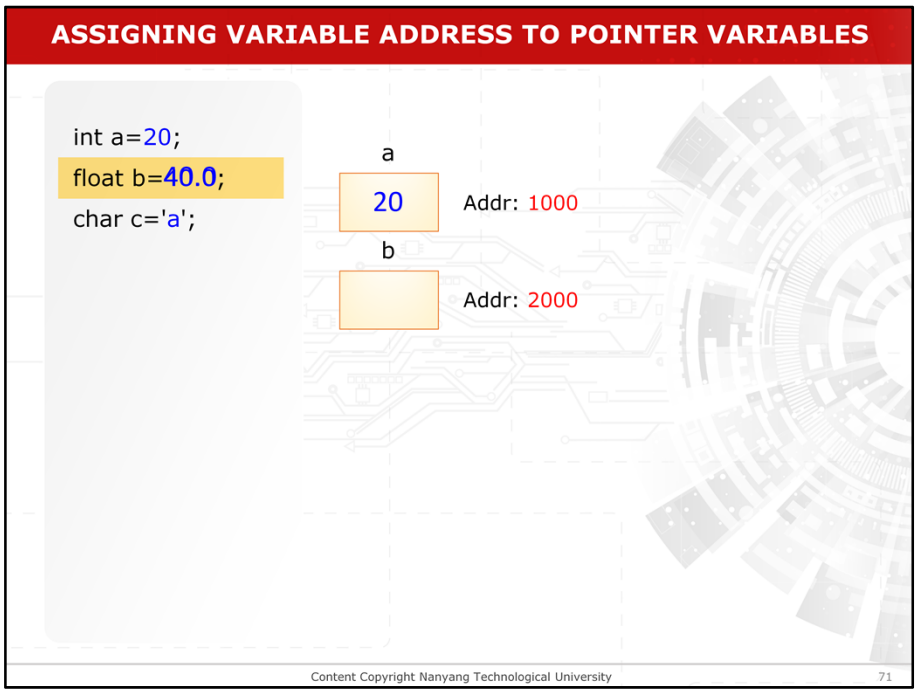
Let’s take a look at this example of Assigning Variable Address to Pointer Variable



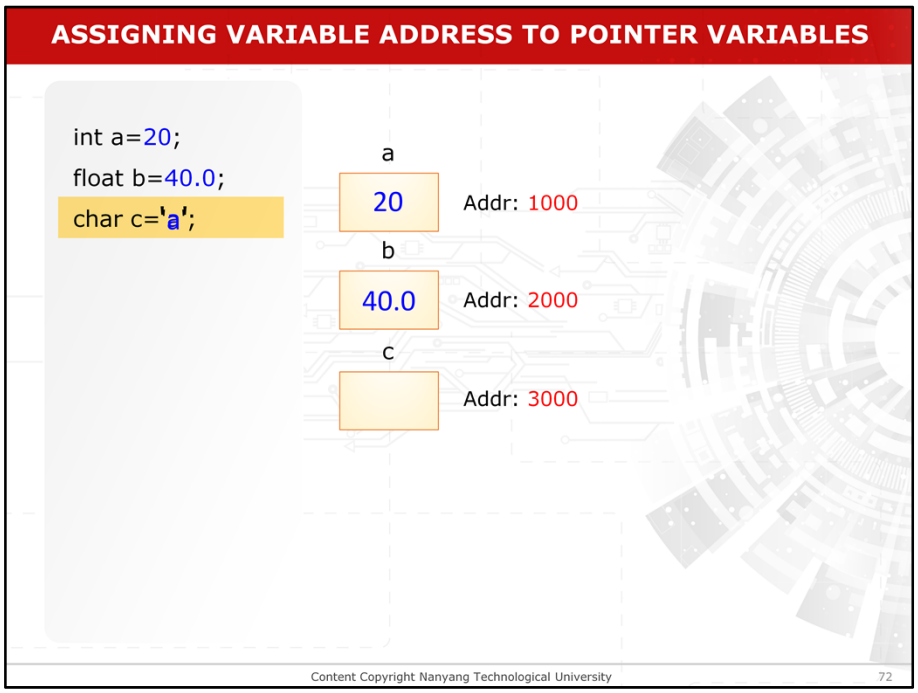
These 3 statements declared variables eh, b, and c of primitive types integer, float, and character respectively.



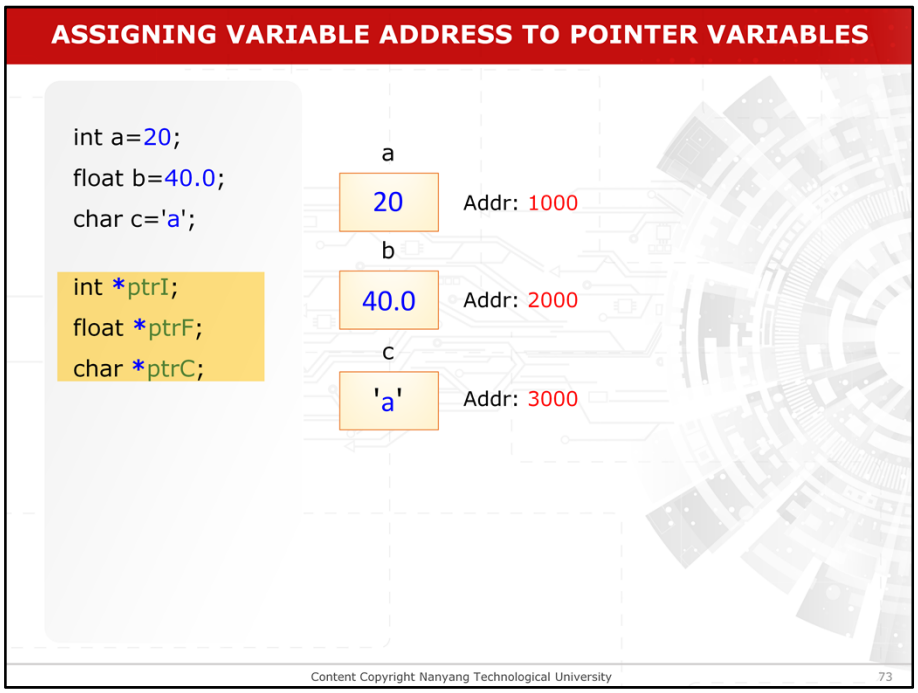
The CPU will allocate the memory address whenever variables are declared.
For simplicity, let's assume that the CPU allocated memory address 1000 for variable eh and store the integer value of 20.



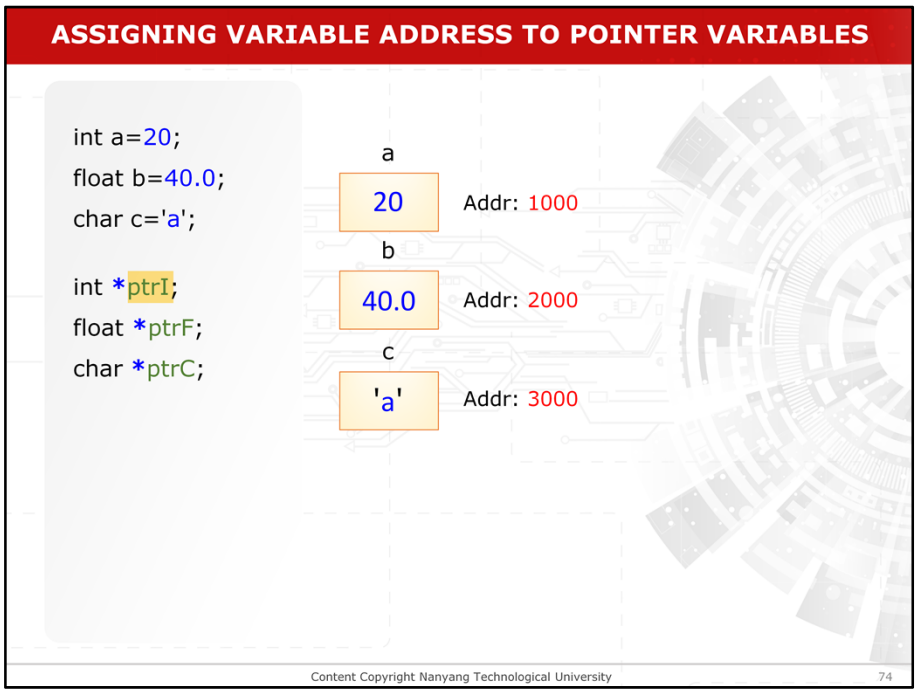
For variable b, the CPU allocated memory address 2000 for it to store a float value of 40.0.



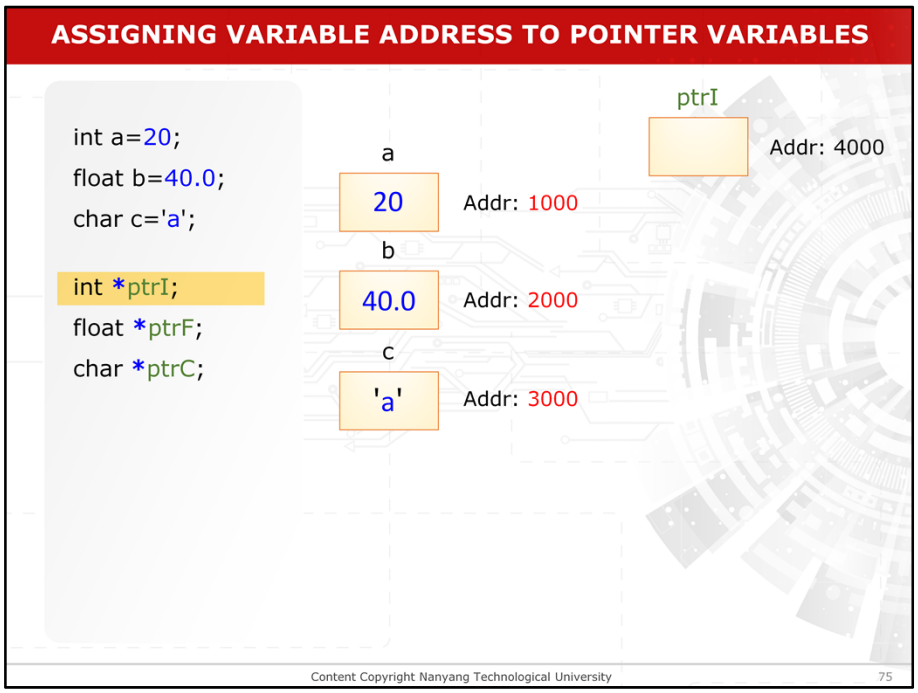
And the memory address 3000 is allocated for variable c to store character eh



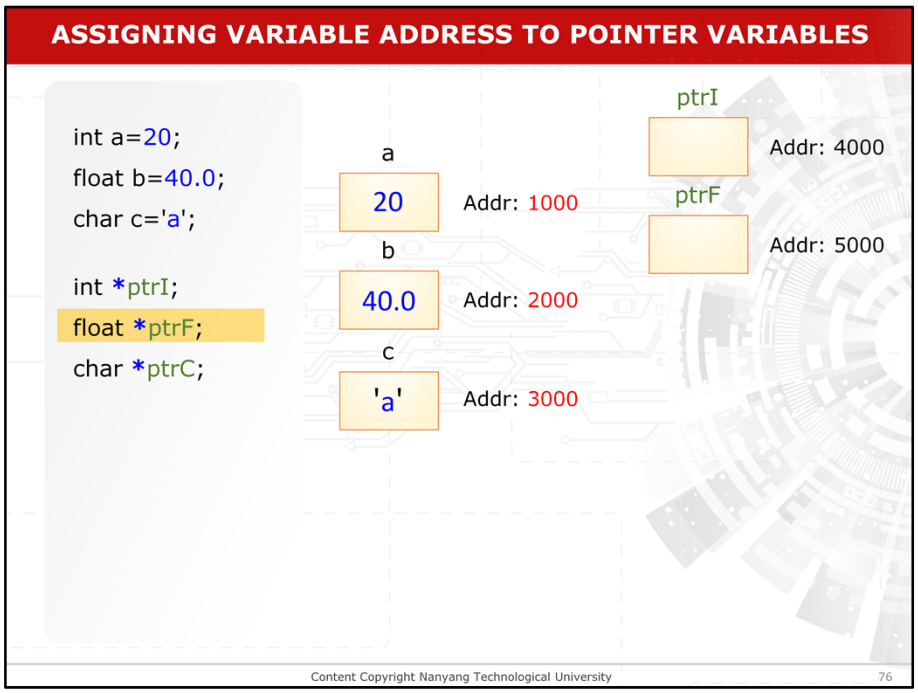
Theses next 3 statements declared pointer variable. Notice the use of the asterisk sign in the pointer variable declaration?



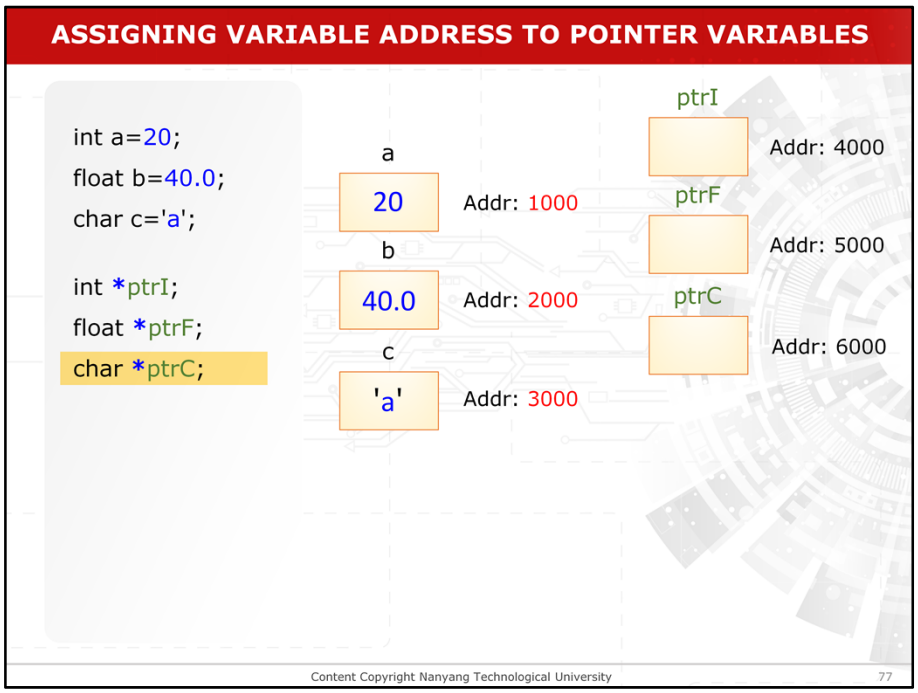
In this statement, the name of the pointer variable is pointer I and is represented by the short form p t r I which we will just call it by pointer i.



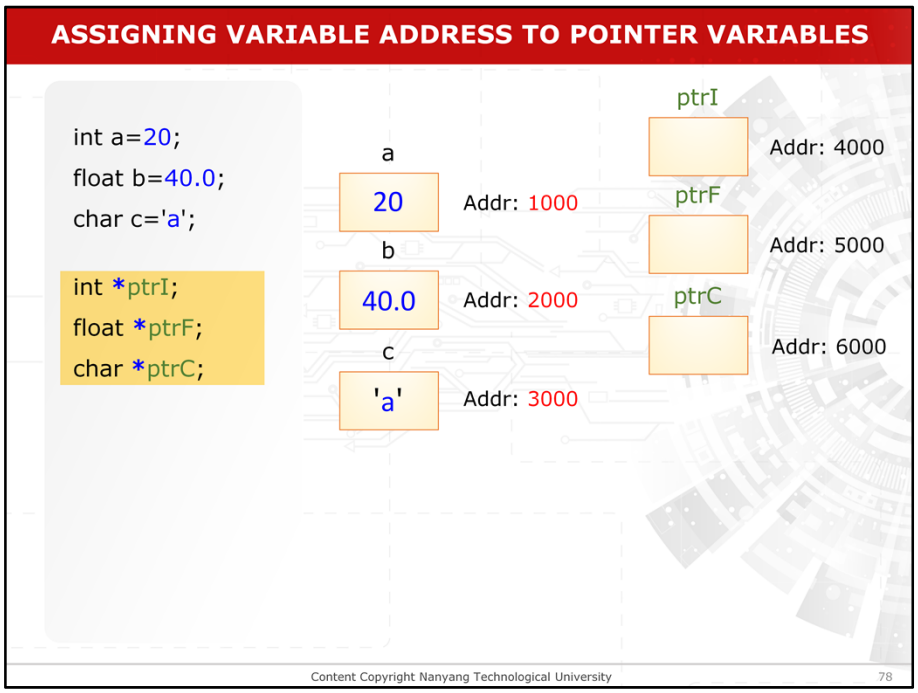
Basically, when this statement is executed, the CPU will just allocate a memory address, say 4000 in this example, for pointer i.



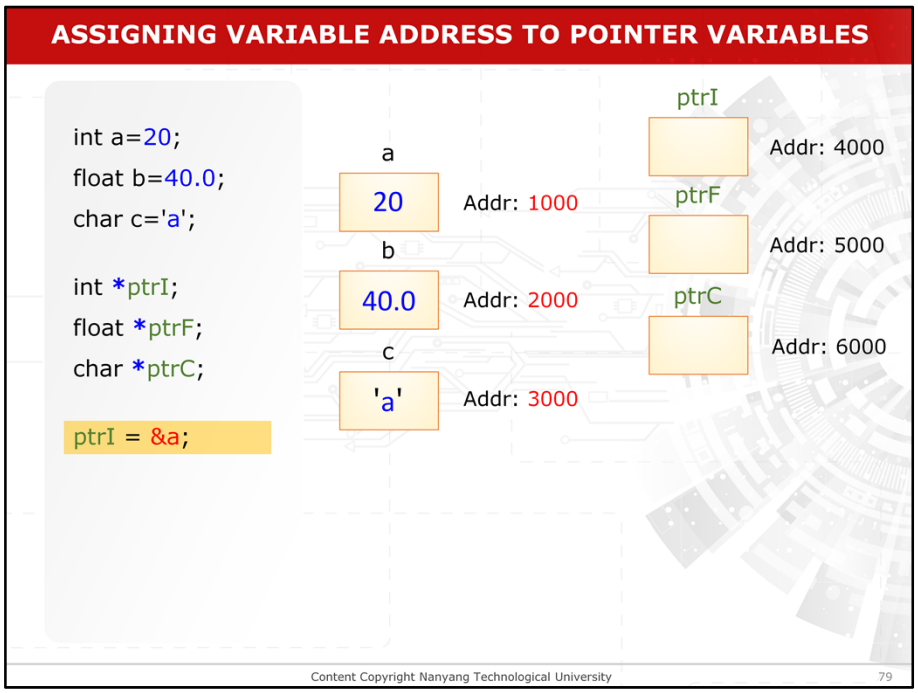
Similarly, pointer F is declared at memory address, say 5000 in this example.



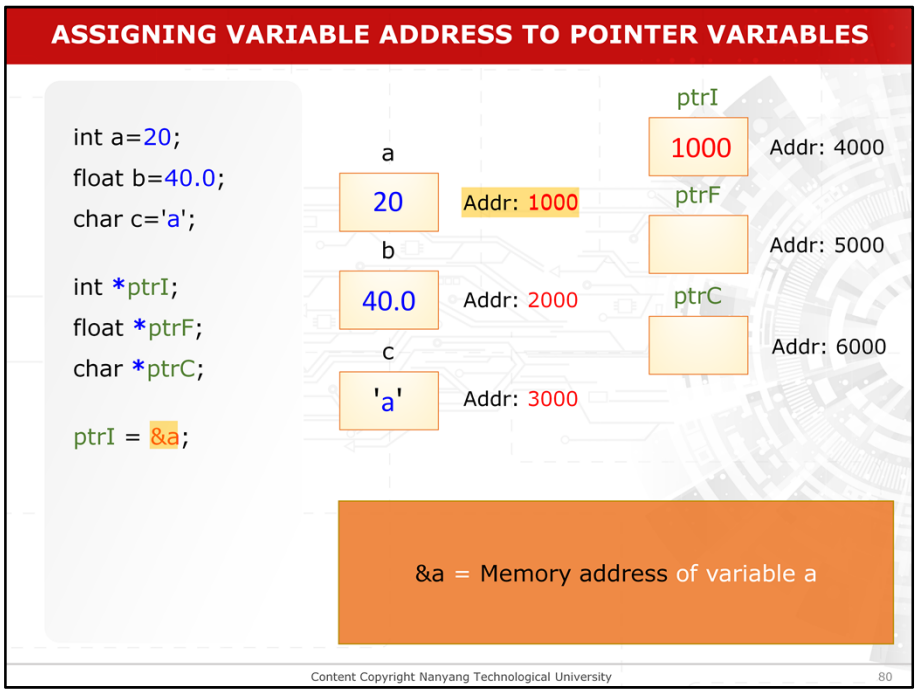
Pointer C is declared as well at memory address, say 6000 in this example.



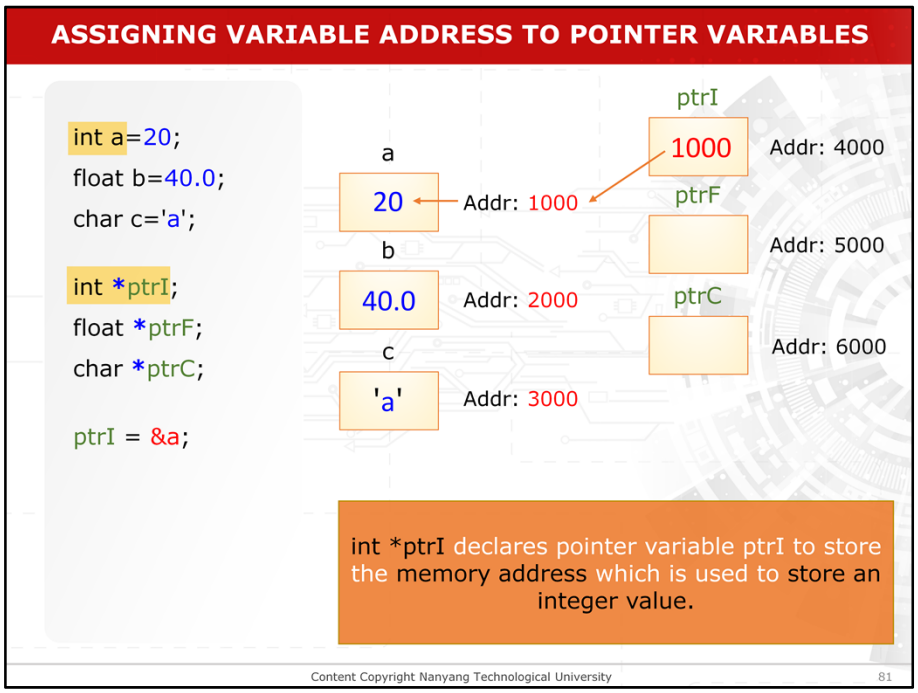
Notice that for these 3 statements, pointer variables: pointer I, pointer F and pointer C have been declared but not assigned any value yet.
We will now look into how to assign the values to these pointer variables.



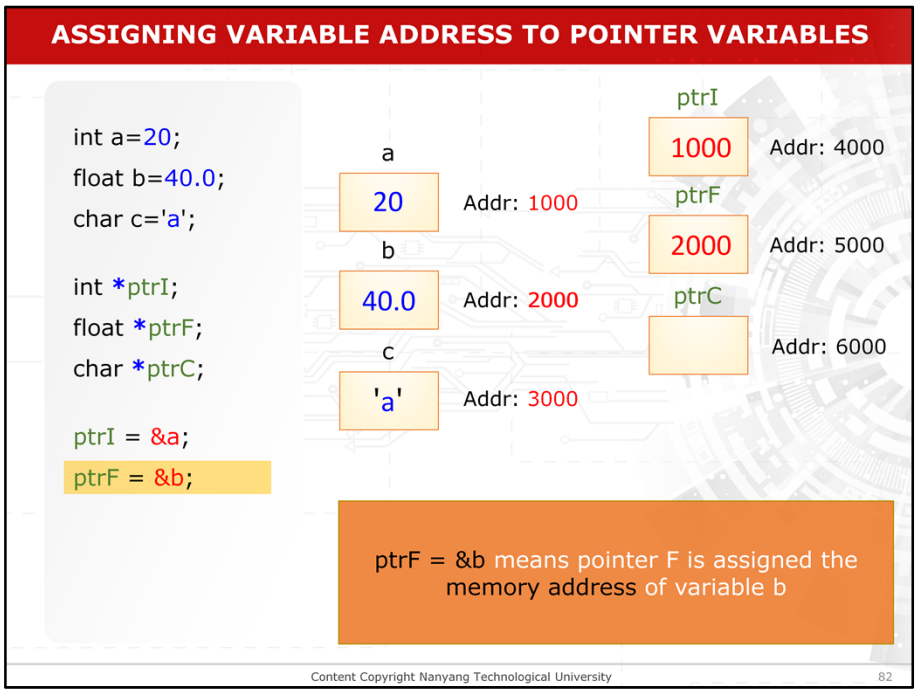
Let's start with pointer I. This statement basically assign the memory address of variable eh to pointer I.



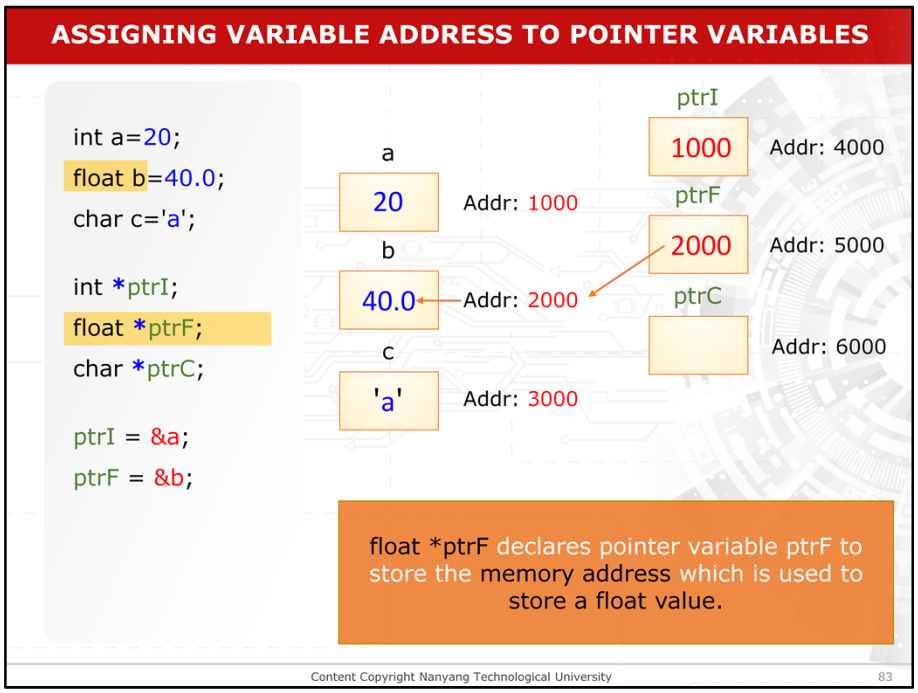
The ampersand sign denotes memory address. So ampersand eh means memory address of variable eh which is 1000 in this example. Thus, the value of 1000 is stored in pointer i.



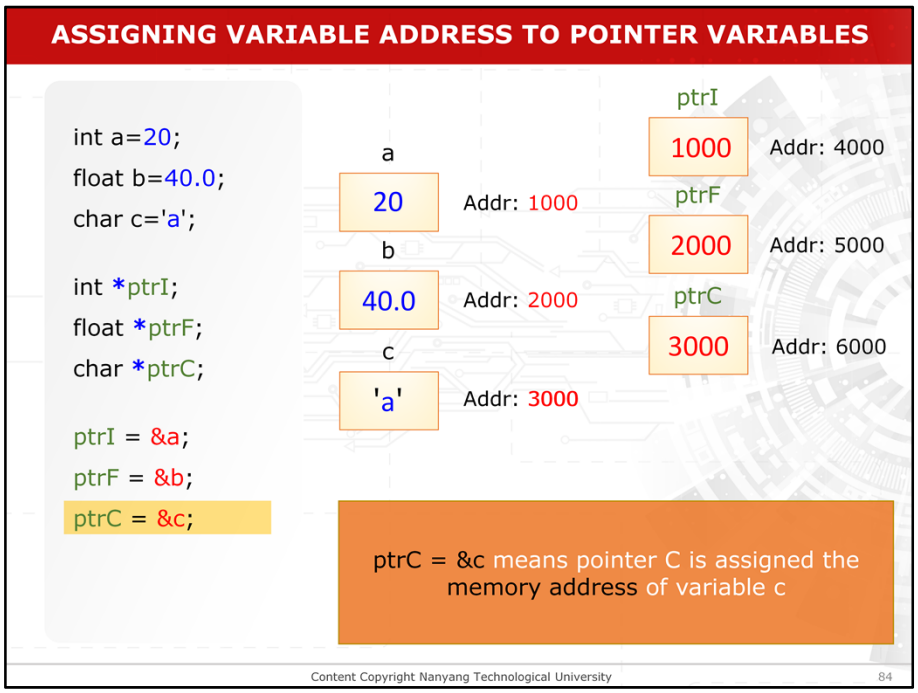
Note that the pointer I was originally declared with statement integer asterisk pointer I which declares pointer variable pointer I to store the memory address that is used to store an integer value.



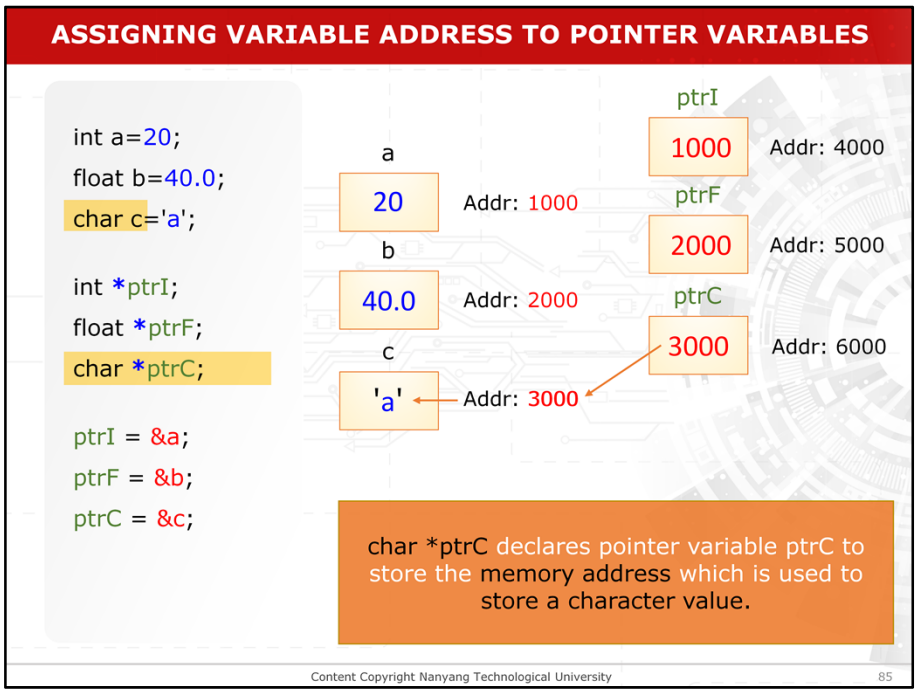
Pointer F equals ampersand b memory address



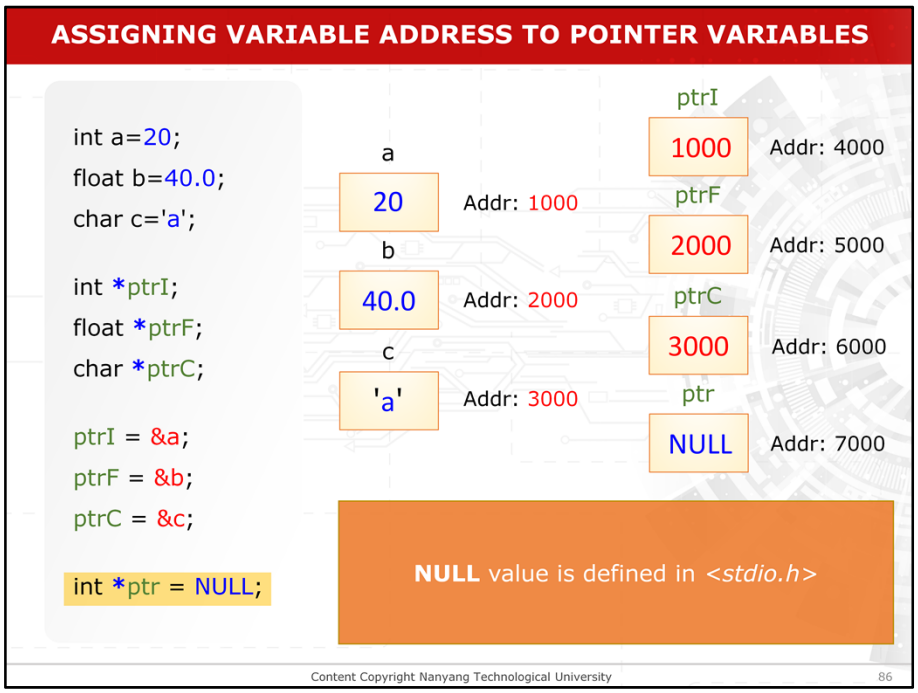
Note that the pointer F was originally declared with statement float asterisk pointer F which declares pointer variable pointer F to store the memory address that is used to store a float value.



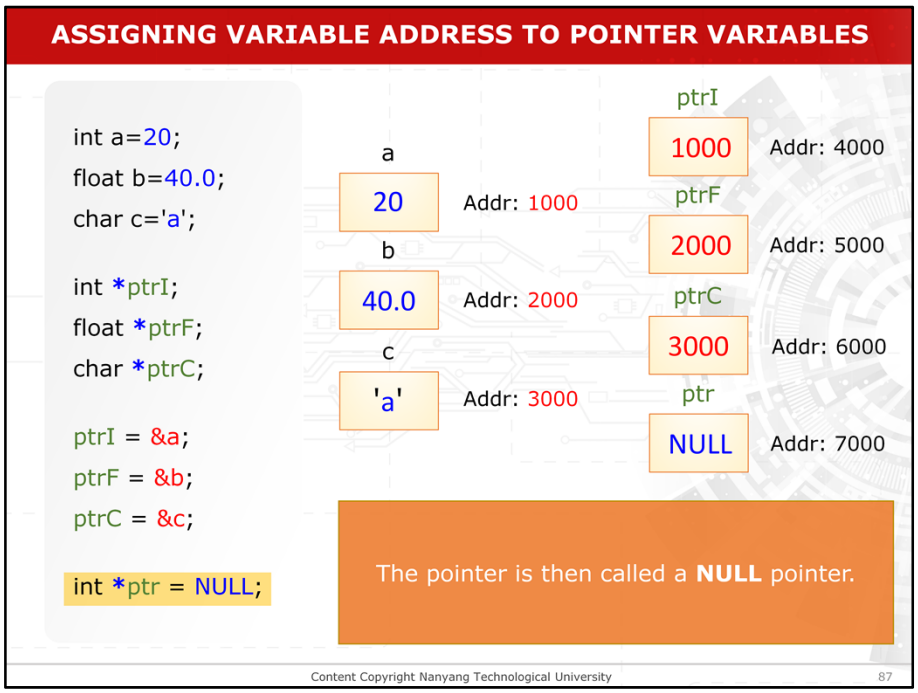
Pointer C equals ampersand c memory address



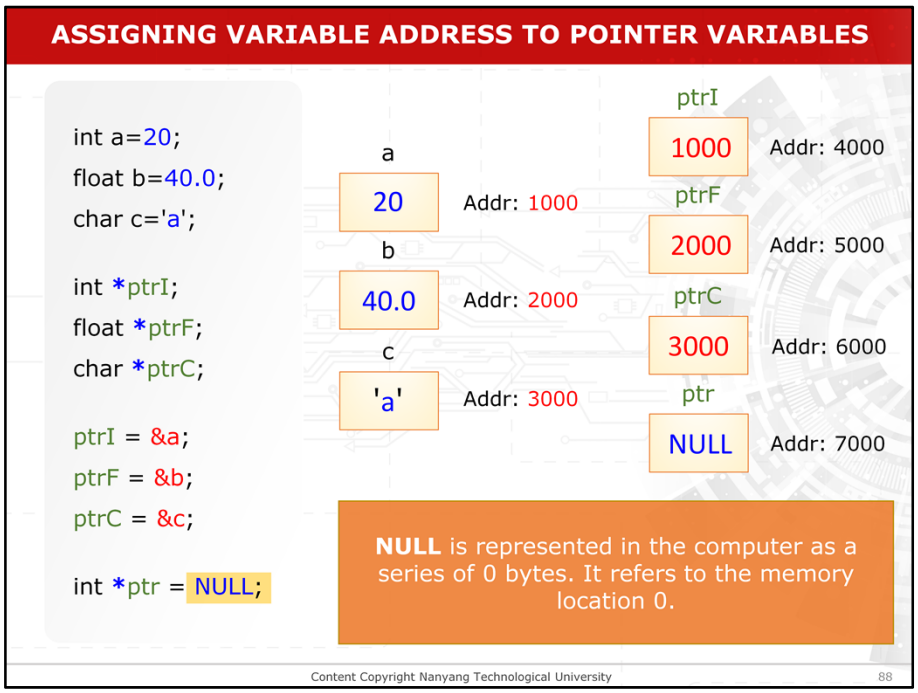
Note that the pointer C was originally declared with statement character asterisk pointer C which declares pointer variable pointer C to store the memory address that is used to store a character value.



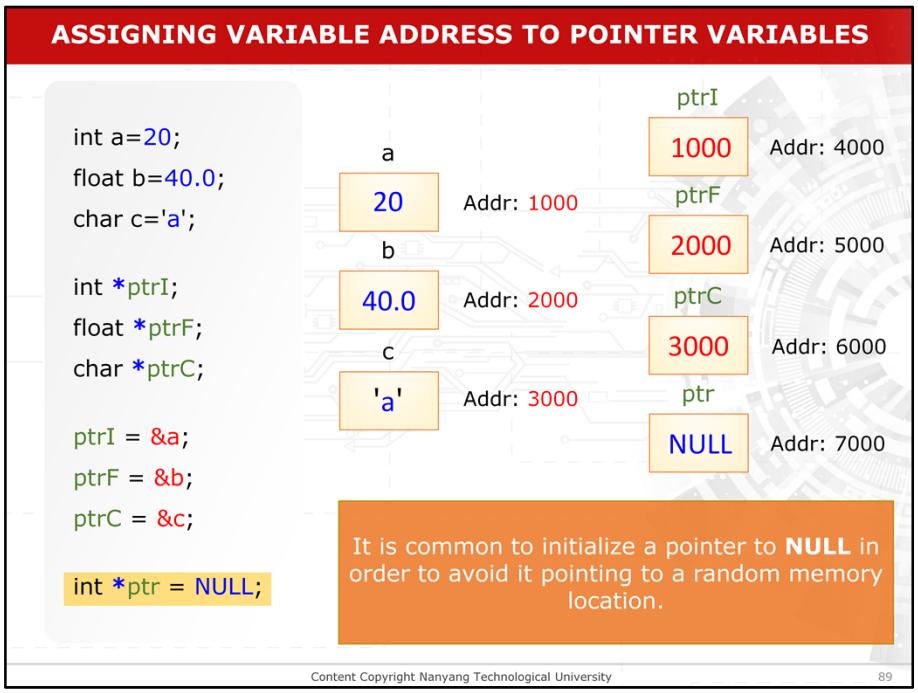
Apart from storing an address value in a pointer variable, we can also store the **NULL** value which is defined in standard eye oh dot aich in a pointer variable.



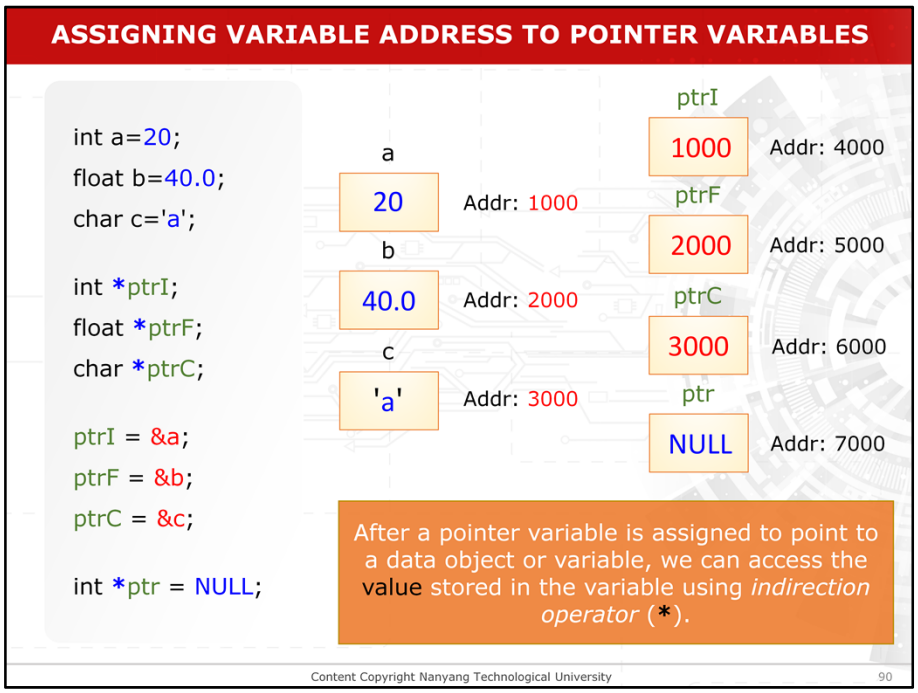
The pointer is then called a **NULL** pointer.



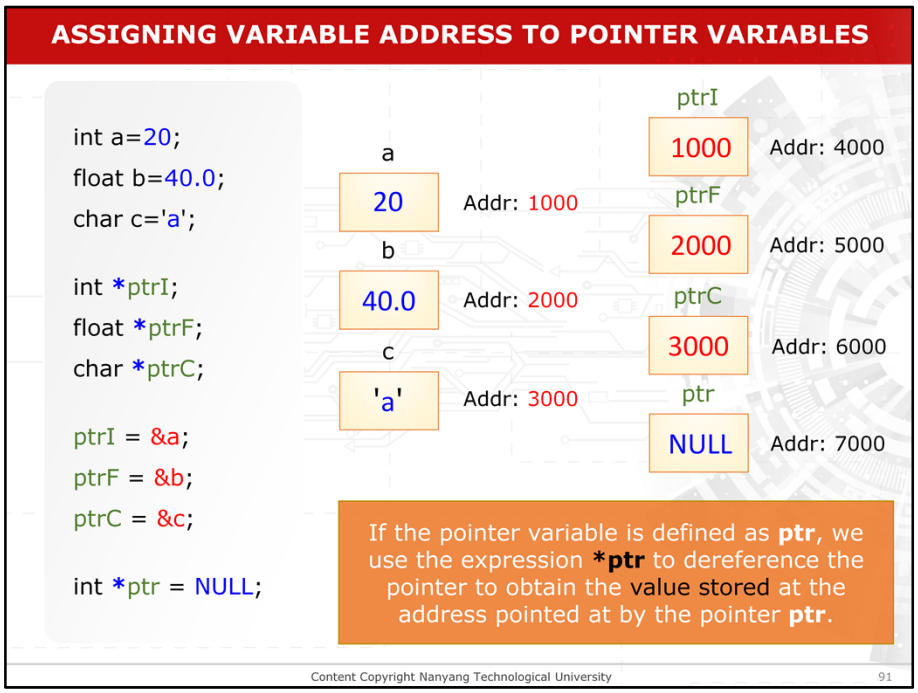
NULL is represented in the computer as a series of 0 bits. It refers to the memory location 0.



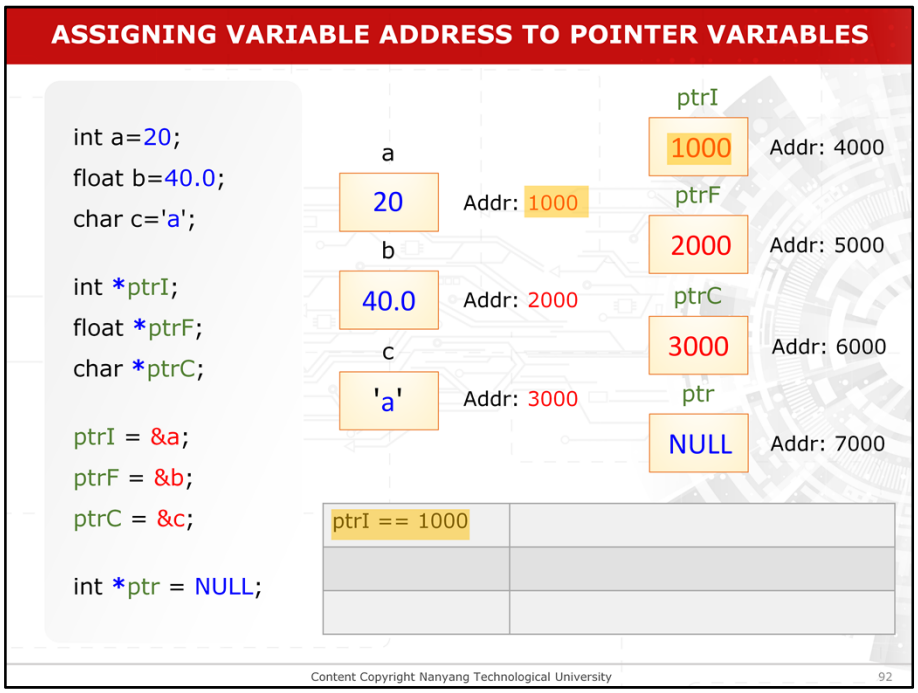
It is common to initialize a pointer to **NULL** in order to avoid it pointing to a random memory location.



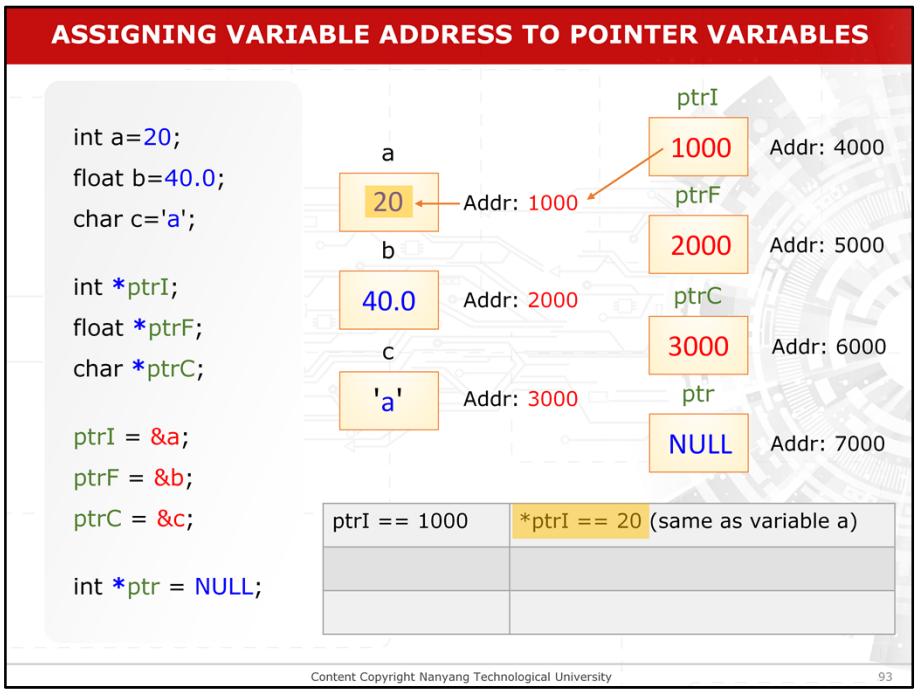
After a pointer variable is assigned to point to a data object or variable, we can access the value stored in the variable using *indirection operator* (**).



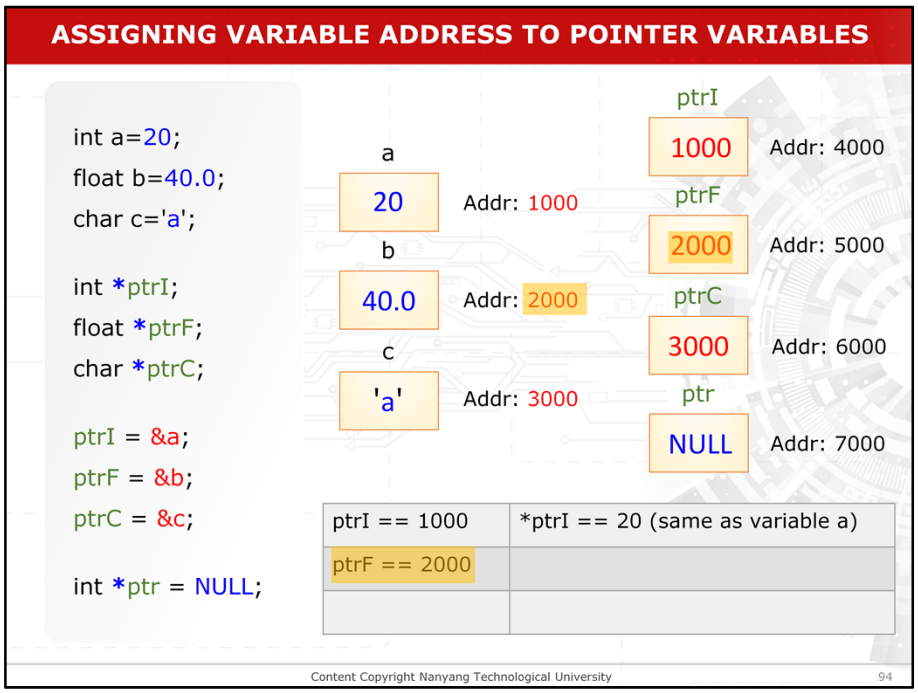
If the pointer variable is defined as **pointer,** we use the expression **asterisk pointer** to dereference the pointer to obtain the value stored at the address pointed at by the pointer. Let's see how this concept is applied in this example.



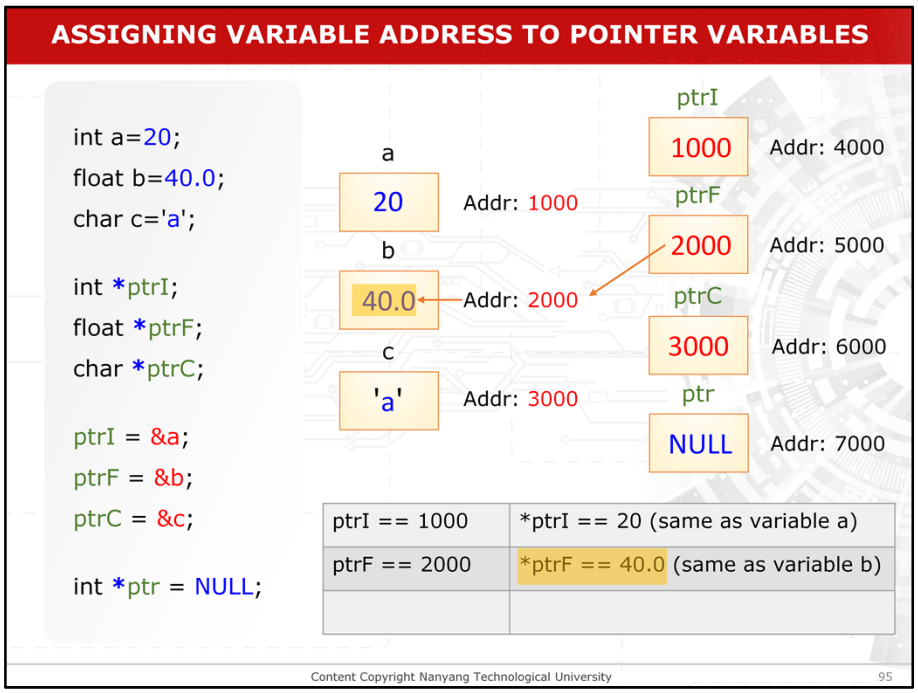
After the assignment operations, we will have **pointer I** stores the memory address of the variable **eh**.



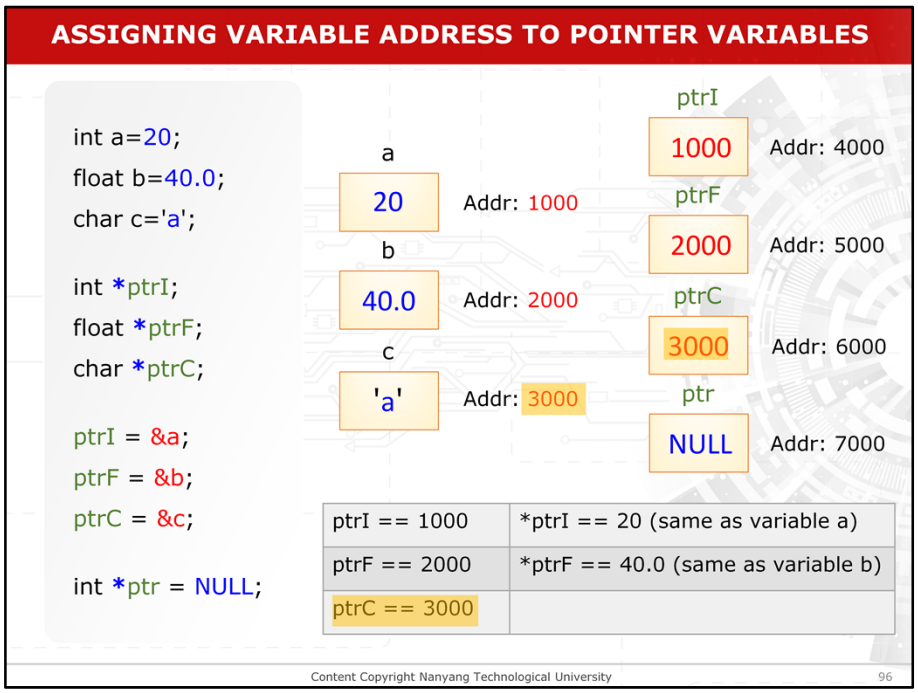
The value of asterisk pointer I is the value stored in memory address of eh which is 20. We can see that asterisk pointer I and variable eh have the same value.



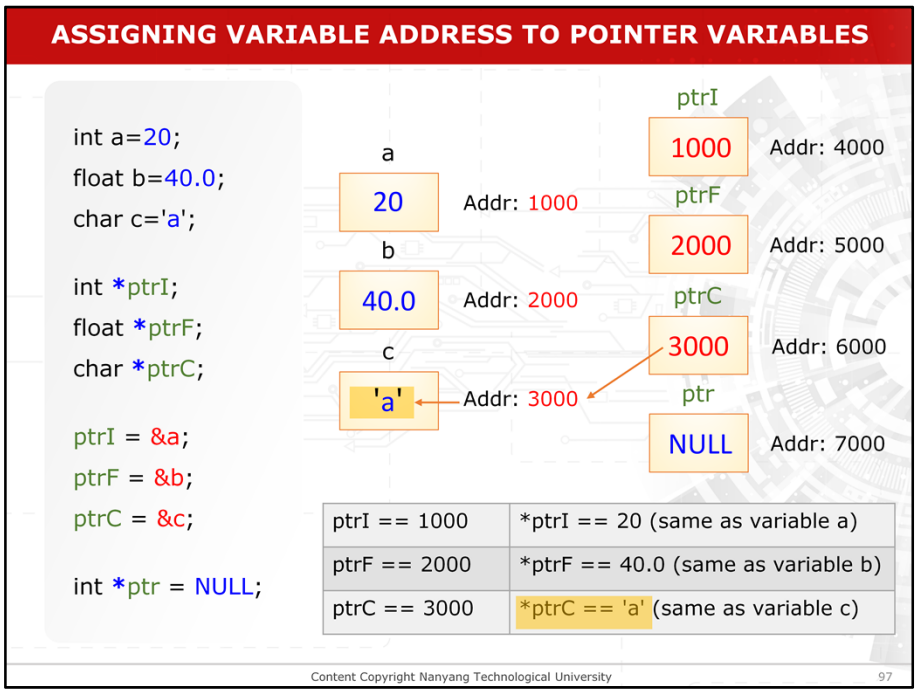
After the assignment operations, we will have **pointer F** stores the memory address of the variable **b**.



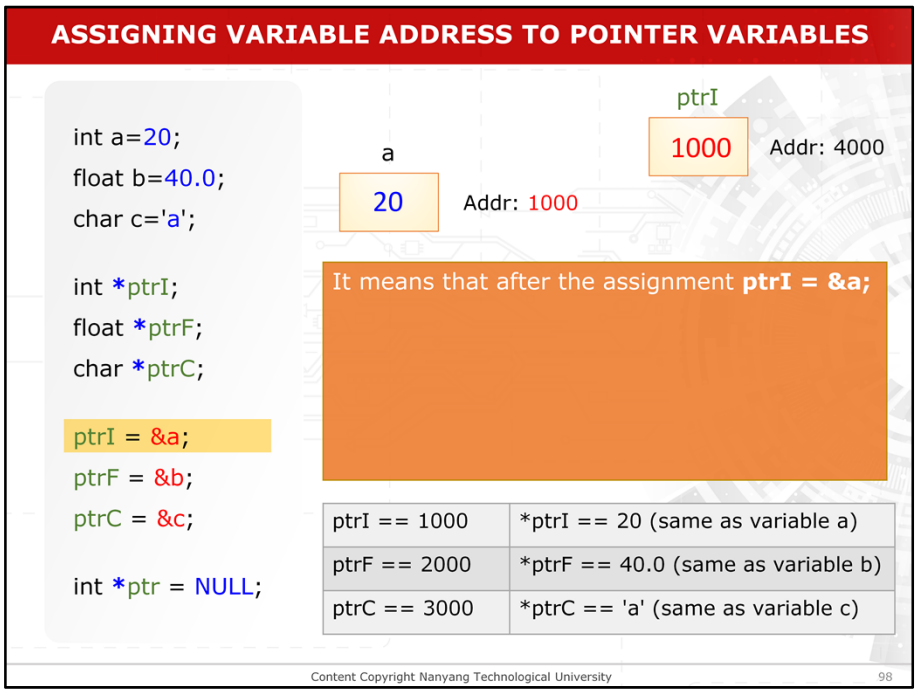
The value of asterisk pointer F is the value stored in memory address of b which is 40.0. We can see that asterisk pointer F and variable b have the same value.



After the assignment operations, we will have **pointer C** stores the memory address of the variable **c**.



The value of asterisk pointer C is the value stored in memory address of c which is the character eh. We can see that asterisk pointer C and variable c have the same value.



It means that after the assignment **pointer I equals ampersand eh;**

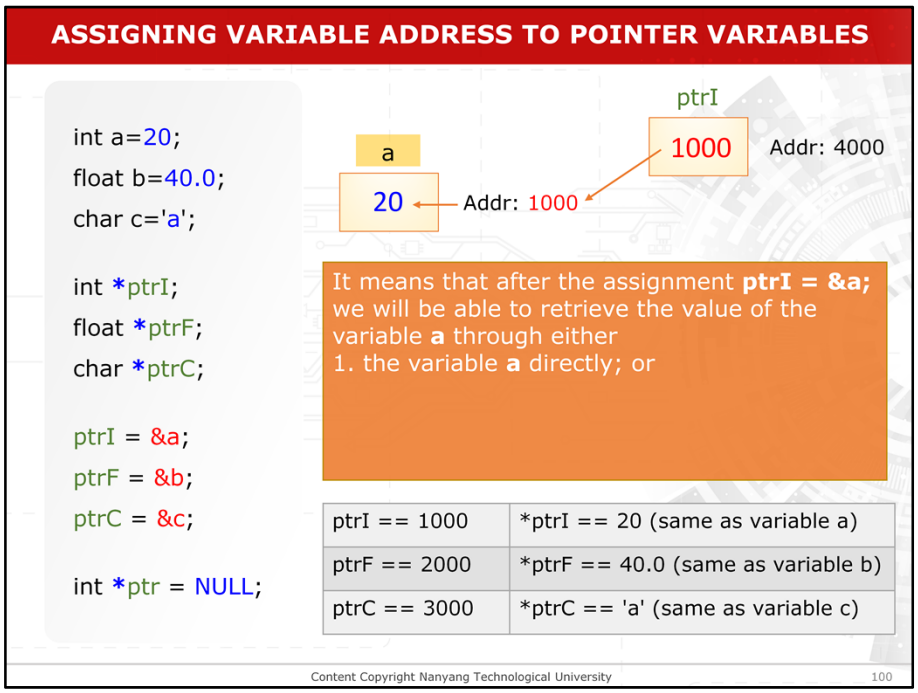
ASSIGNING VARIABLE ADDRESS TO POINTER VARIABLES

```
int a=20;
float b=40.0;
char c='a';

int *ptrI;
float *ptrF;
char *ptrC;

ptrI = &a;
ptrF = &b;
ptrC = &c;

int *ptr = NULL;
```

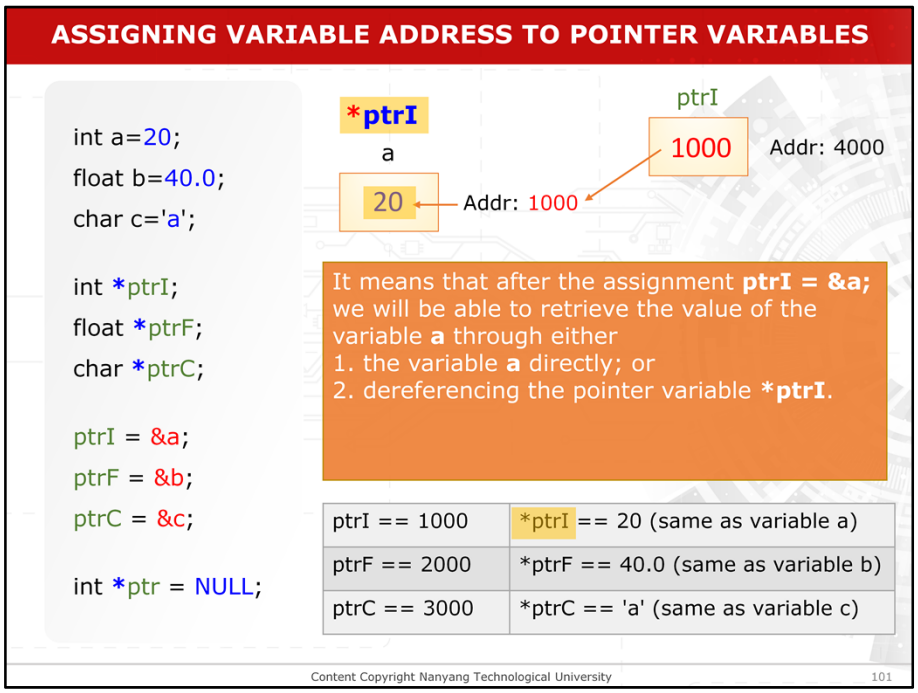
It means that after the assignment **ptrI = &a;** we will be able to retrieve the value of the variable **a** through either

ptrI == 1000	*ptrI == 20 (same as variable a)
ptrF == 2000	*ptrF == 40.0 (same as variable b)
ptrC == 3000	*ptrC == 'a' (same as variable c)

we will be able to retrieve the value of the variable **eh** through either



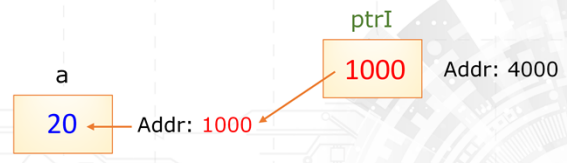
the variable eh directly; or



dereferencing the pointer variable **asterisk pointer i.**

ASSIGNING VARIABLE ADDRESS TO POINTER VARIABLES

```
int a=20;
float b=40.0;
char c='a';

int *ptrI;
float *ptrF;
char *ptrC;

ptrI = &a;
ptrF = &b;
ptrC = &c;

int *ptr = NULL;
```



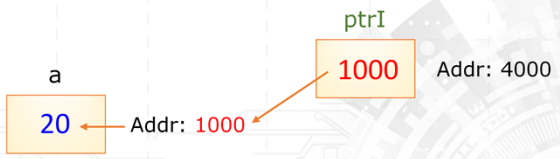
It means that after the assignment **ptrI = &a;** we will be able to retrieve the value of the variable **a** through either
1. the variable **a** directly; or
2. dereferencing the pointer variable ***ptrI**.

ptrI == 1000	*ptrI == 20 (same as variable a)
ptrF == 2000	*ptrF == 40.0 (same as variable b)
ptrC == 3000	*ptrC == 'a' (same as variable c)

This concept also applies to retrieving of pointer F and pointer C.

ASSIGNING VARIABLE ADDRESS TO POINTER VARIABLES

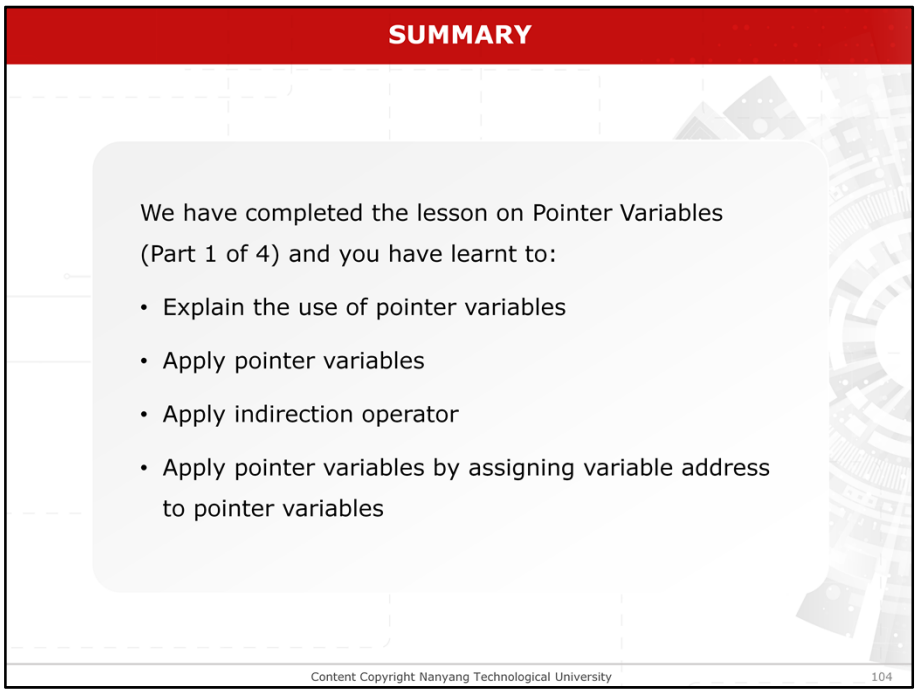
```
int a=20;
float b=40.0;
char c='a';

int *ptrI;
float *ptrF;
char *ptrC;

ptrI = &a;
ptrF = &b;
ptrC = &c;

int *ptr = NULL;
```



It means that after the assignment **ptrI = &a;** we will be able to retrieve the value of the variable **a** through either
1. the variable **a** directly; or
2. dereferencing the pointer variable ***ptrI**.
Therefore, we can write programs more flexibly by using pointer variable.

ptrI == 1000	*ptrI == 20 (same as variable a)
ptrF == 2000	*ptrF == 40.0 (same as variable b)
ptrC == 3000	*ptrC == 'a' (same as variable c)

Therefore, we can write programs more flexibly by using pointer variable.



We have completed the lesson on Pointer Variables (Part 1) and you have learnt the points listed. In the next lesson, we will be learning how these points work in proper programming codes with an example.