- Username : class-1
- Password :

- Check VS 2019 whether can use
- We will start our course in 18:30
- we will start demonstrate the exercises at 19:15.
- Do not use scanf_s
- Please make sure the TA has recorded your exercise score here before leaving.

Introduction to Computers and Programming

-- *POINTER –*

WEEK 09

What is a Pointer?

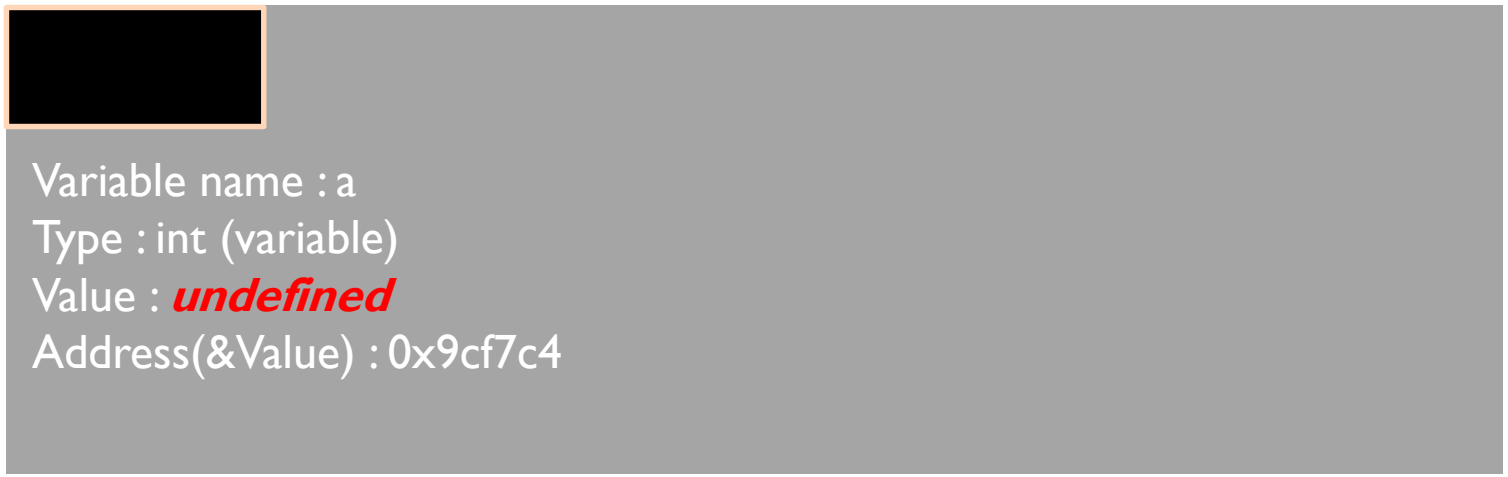
Before we start: Declaration

Once you declare a variable, the system will allocate a piece of memory space to store the value of this variable.

Before we start: Declaration

The system accesses the memory space via a “memory address”, so the program actually maps a “variable name” to its “memory address”.

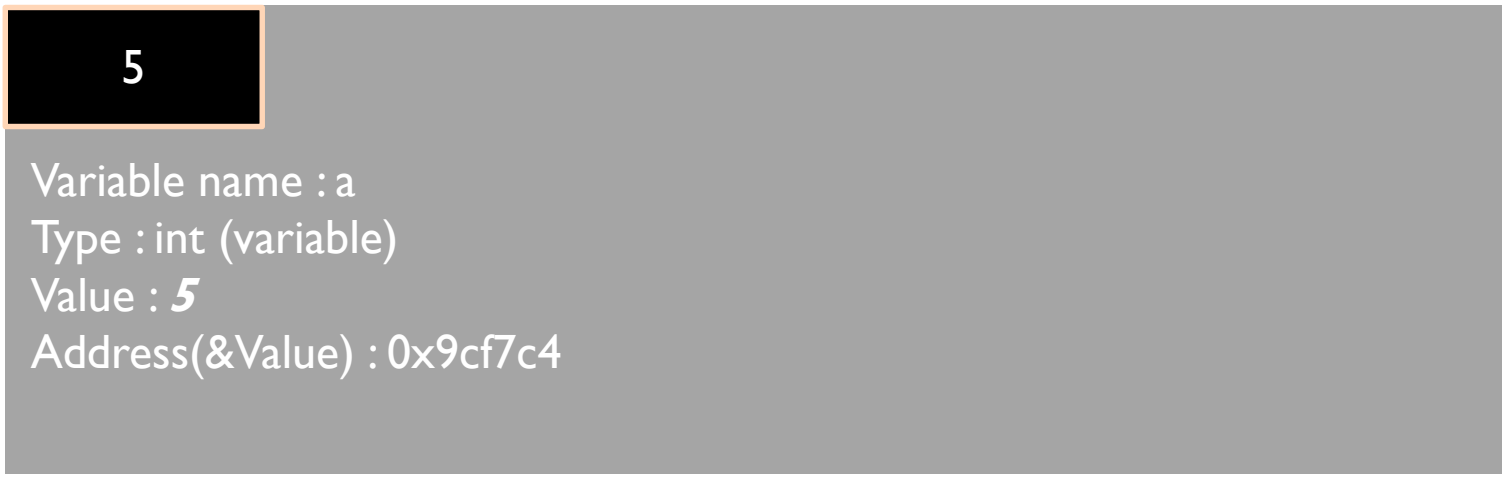
```
int a;
```



Before we start: Declaration

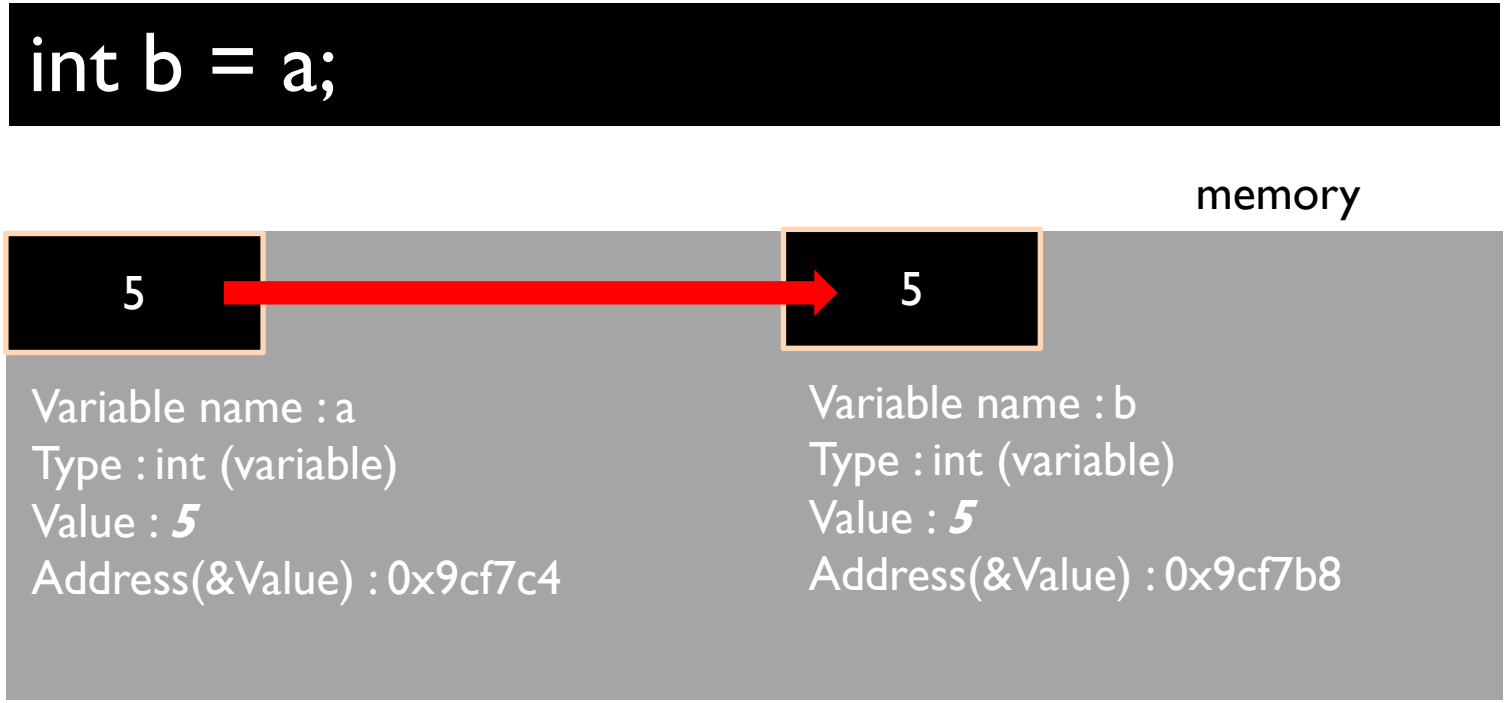
The variable has to be assigned a value before being used. Otherwise you will get an arbitrary value.

```
a = 5;
```



Before we start: Declaration

When we assign a variable to another, actually its value is copied into the new variable.



What is a Pointer?

- A pointer is a variable that holds the address of some other variable.

- Declare a pointer: use *

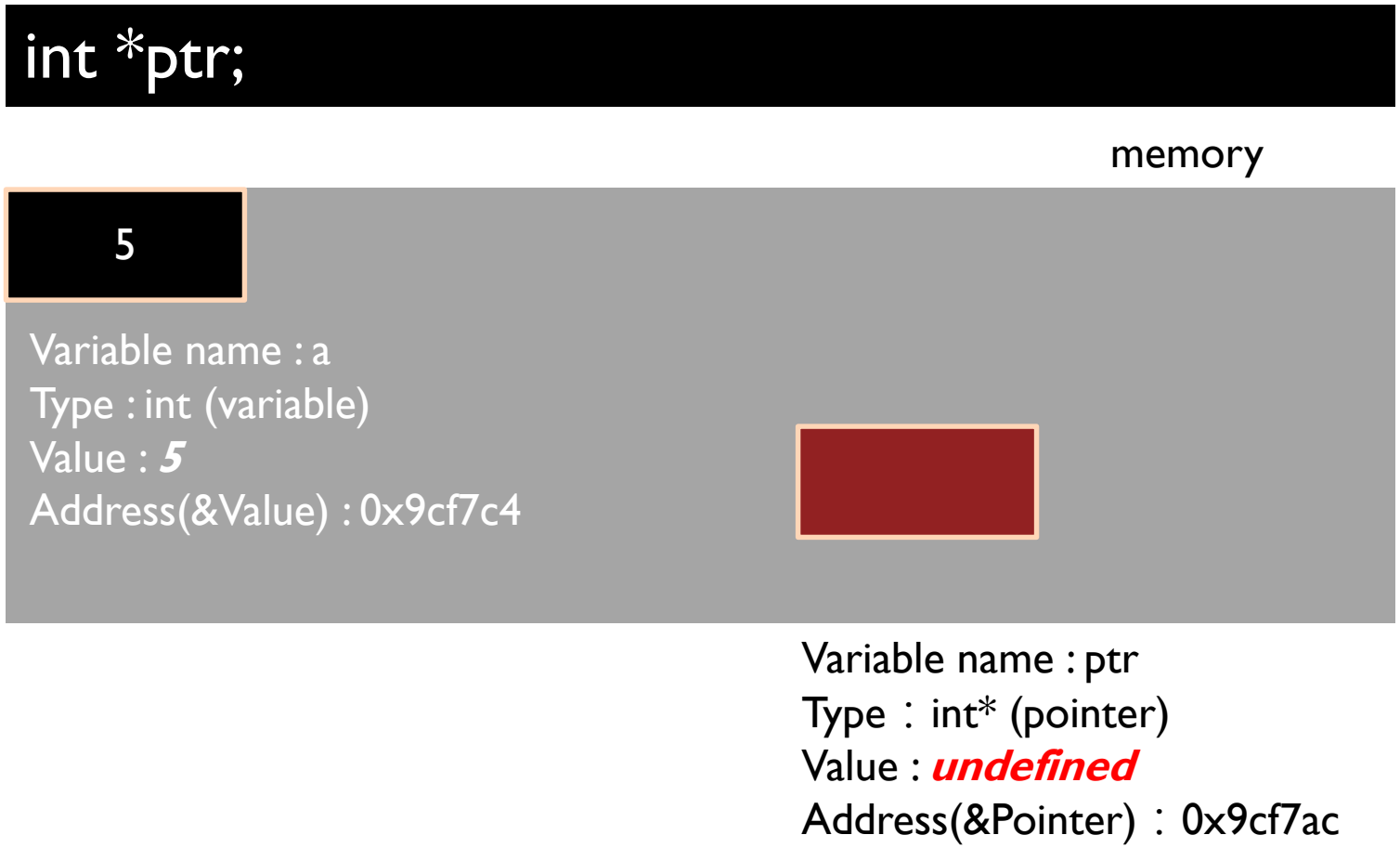
```
(some type) *PointerName
```

Example:

```
int *ptr;
```

How to Use a Pointer

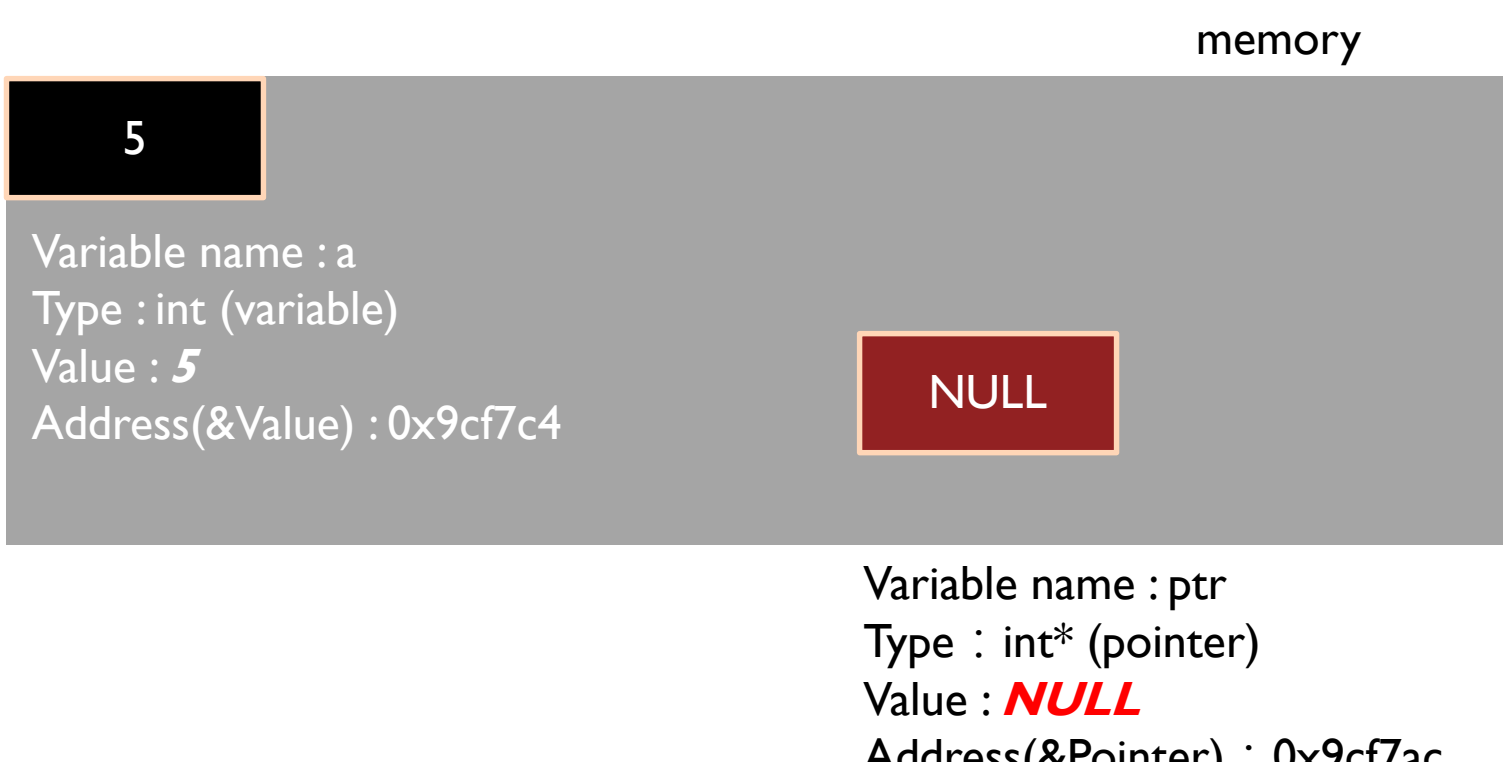
First, declare a pointer.



How to Use a Pointer

Good practice: initialize every pointer with NULL.

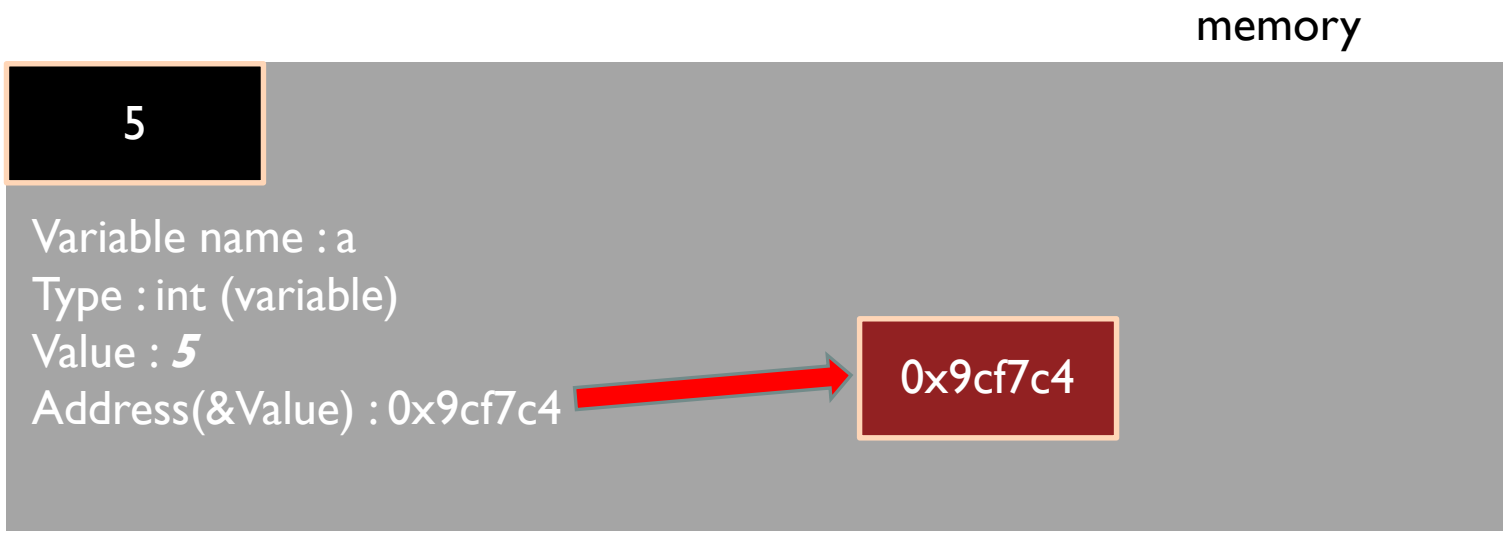
```
ptr = NULL;    // No need of * when assigning!
```



How to Use a Pointer

Then find the address of the variable and assign it to the pointer variable.

```
ptr = &a;      // "&a" is the address of "a",
               // which is 0x9cf7c4.
```

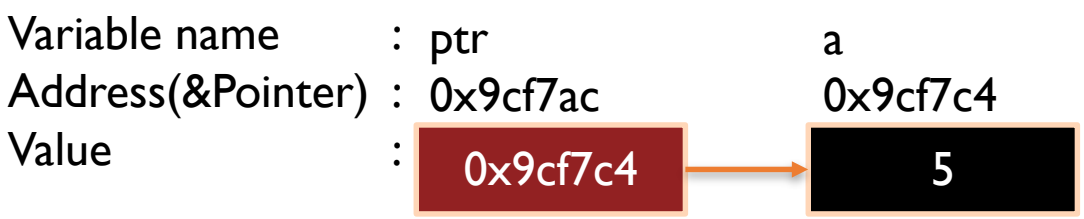


Variable name : ptr
Type : int* (pointer)
Value : ***0x9cf7c4***
Address(&Pointer) : 0x9cf7ac

How to Use a Pointer

```
ptr = &a;    // "&a" is the address of "a",
             // which is 0x9cf7c4.
```

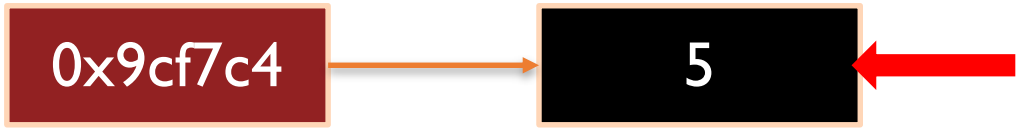
Variable name	:	ptr	a
Address(&Pointer)	:	0x9cf7ac	0x9cf7c4
Value	:	0x9cf7c4	5



How to Use a Pointer

*ptr: Value of the variable which the pointer is pointed

```
printf("%d\n", *ptr);   // print the content that Pointer is pointed
```



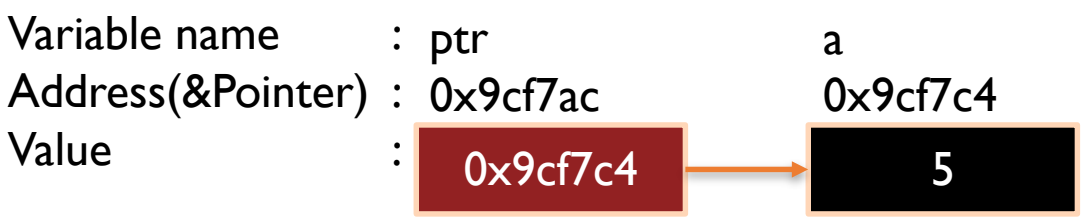
ptr: value of the pointer (same as normal variable)

```
printf("%x\n", ptr);    // print the content of the pointer variable ptr.
```

&ptr: address of the pointer (same as normal variable)

```
printf("%x\n", &ptr);    // print the address of the pointer variable ptr.
```

How to Use a Pointer

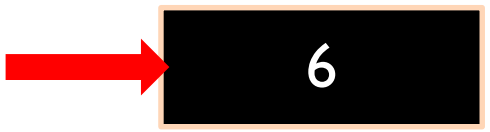


Assign 6 to the variable which Pointer is pointed.

```
*ptr = 6;       // assign 6 to the variable which Pointer is pointed
```

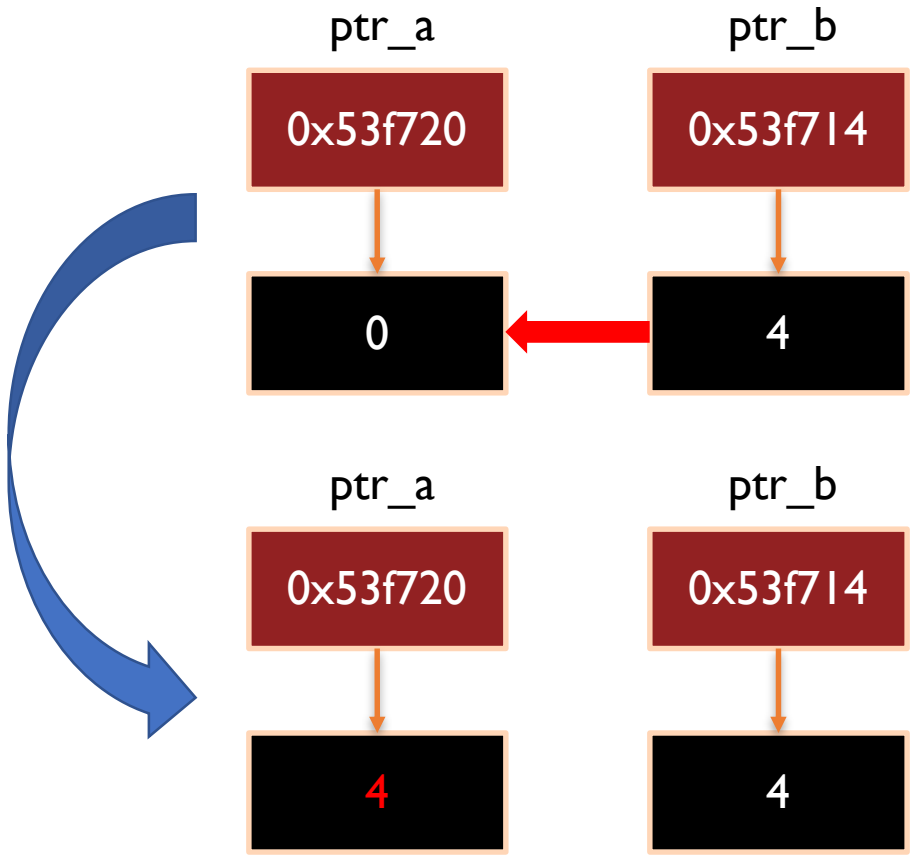
The variable which Pointer is pointed changed.

```
printf("%d\n", a);      // print the value of a
```

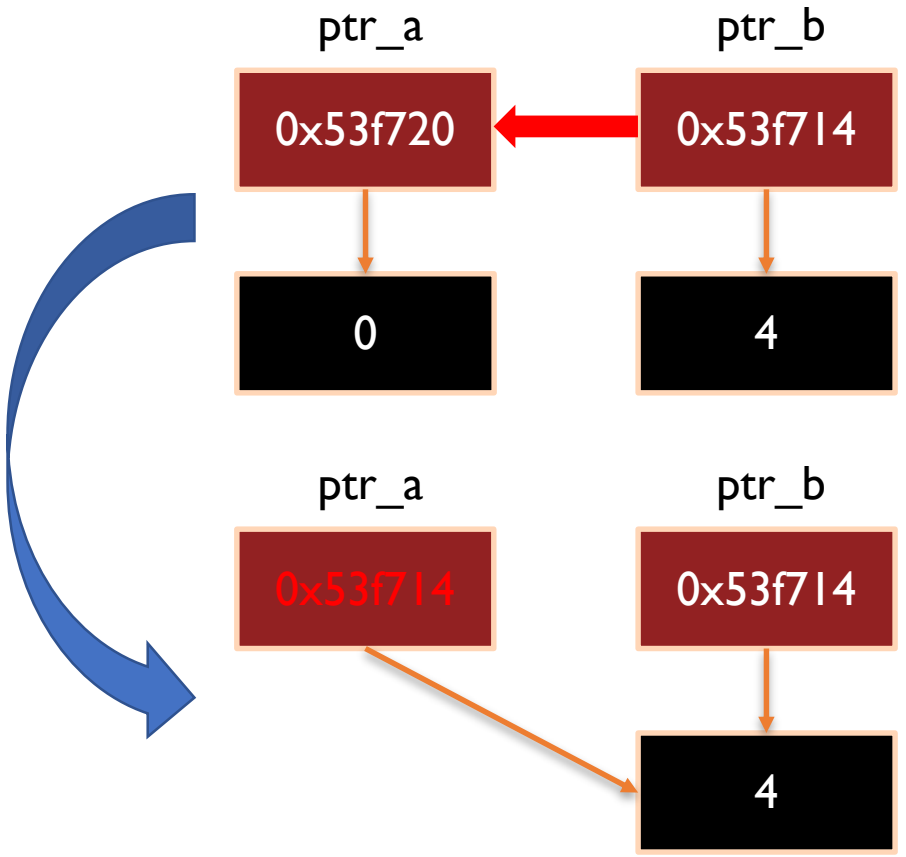


Interaction Between Pointers

```
*ptr_a = *ptr_b;     // assign the value pointed by ptr_b to the variable
                     // pointed by ptr_a, and they are different variables
```




```
ptr_a = ptr_b;     // assign the address in ptr_b to ptr_a, and
                   // they will point to the same memory cell.
```



Returning a Pointer

Just like any other types, the return value of a function could be a pointer.
All you need to do is to use a pointer type such as “int *” as the return type.

```c
int *return_pointer(int a)
{
    int *ptr = NULL;
    ptr = (int*)calloc(1, sizeof(int));
    *ptr = a;
    return ptr;
}

int main()
{
    int a = 1;
    int *ptr = NULL;

    ptr = return_pointer(a);

    printf("*ptr after call function = %d\n", *ptr);

    system("pause");
    return 0;
}
```

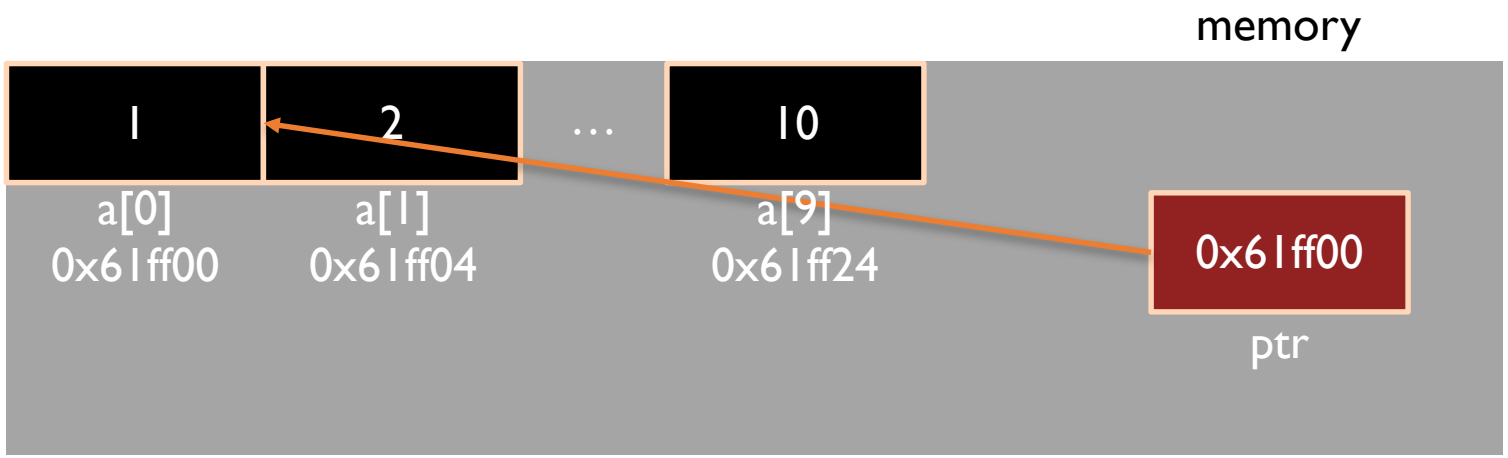
```
*ptr after call function = 1
請按任意鍵繼續 . . .
```

Pointers and Arrays

Pointer to an Array

The pointer can point to an array, and works as a normal pointer variable.

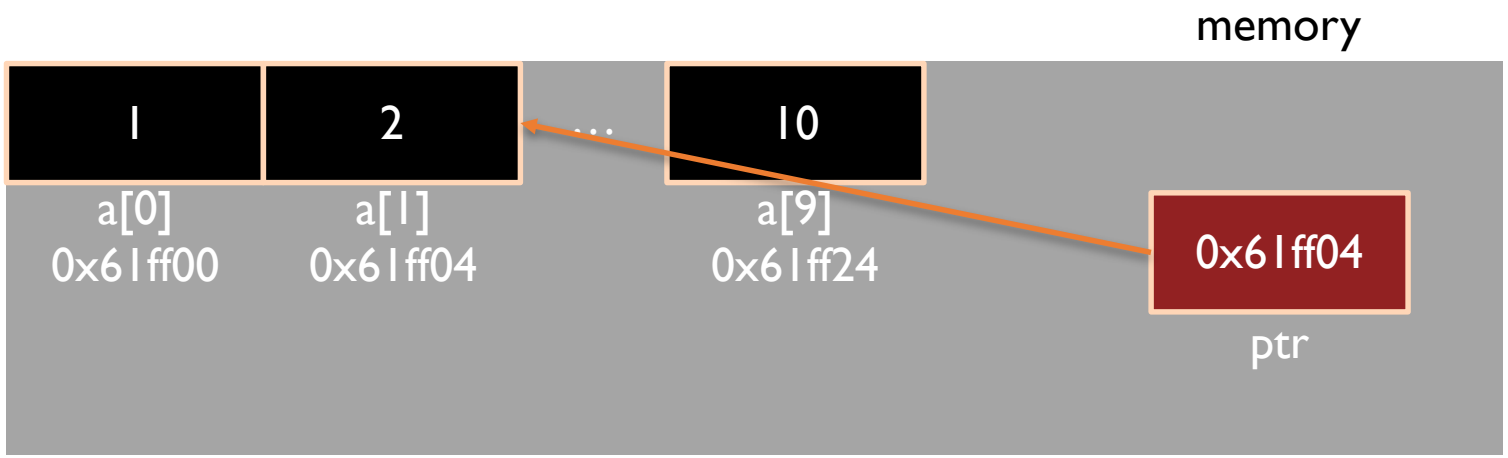
```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *ptr;
ptr = a;                    // Do not use "prt = &a;". This is a type error.
printf("%d", *ptr);         // output will be 1
```



Pointer to an Array

A pointer variable can point to an element in an array.

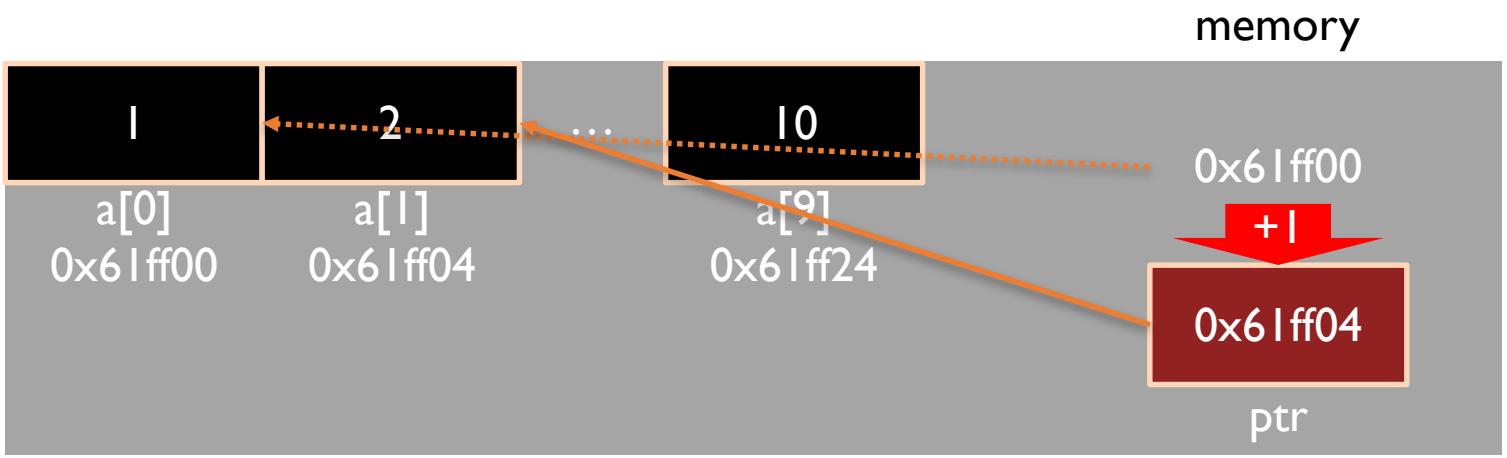
```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *ptr;
ptr = &a[1];              // notice that & is needed now
printf("%d", *ptr);       // output will be 2
```



Pointer Arithmetic

Addition and subtraction can be performed on a pointer variable.

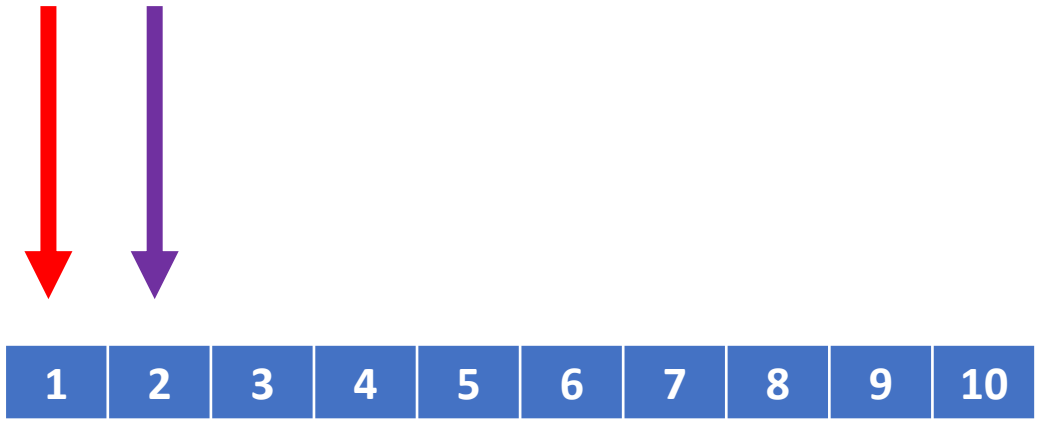
```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *ptr;
ptr = a;
ptr = ptr + 1;          // move ptr to the next element
```



Pointer to an Array and its Arithmetic -- Example

```c
int main()
{
    int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int *ptr;

    ptr = a;
    printf("*ptr = %d\n", *ptr);

    ptr = &a[1];
    printf("*ptr = %d\n", *ptr);

    ptr = a;
    ptr = ptr + 1;
    printf("*ptr = %d\n", *ptr);

    system("pause");
    return 0;
}
```

```
*ptr = 1
*ptr = 2
*ptr = 2
請按任意鍵繼續 . . .
```



Dynamic Memory Allocation

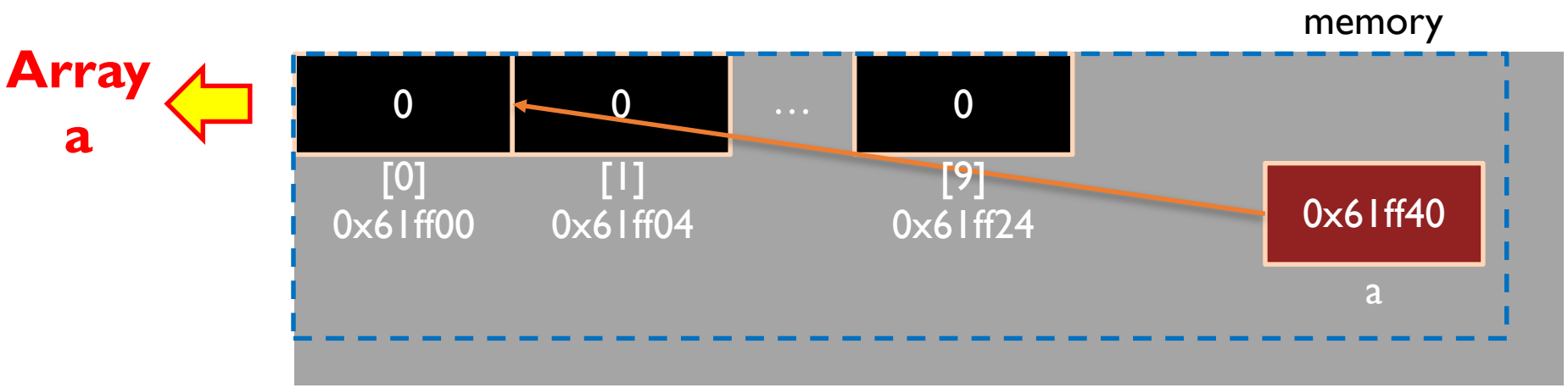
Allocate an Array

Usually we define an array as follows:

```
int a[10];
```

Another way to do it is to declare a pointer and allocate a piece of memory space to it:

```
int *a = NULL;
a = (int*)calloc(10, sizeof(int));
```

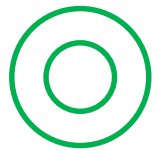


Benefit of using dynamic memory allocation

(1) We can create an array with user defined size.

```
int n;
n = 3;
int arr[n];      // compile error!!
```

```
int n;
n = 3;
int *arr = NULL;
arr = (int*)calloc(n, sizeof(int));    // valid expression
```



Benefit of using dynamic memory allocation

(2) Memory space will be kept until the program end or free() is called, so we can create an array inside the function and return it back to main()

```
int *func() {
    int arr[10];
    …
    return arr; }     //  arr is deleted when func() ends.
```

```
int *func() {
    int *arr = (int*)calloc(10, sizeof(int));
    …
    return arr; }   // will be kept in memory until the
program stops.
```

Functions related to allocation

- (type*)calloc(N, S)

Allocate a continuous space that contains N elements with size S each, and fill them with 0.

- (type*)malloc(N * S)

Allocate a continuous space that contains N elements with size S. Won't initialize it.

Functions related to allocation

- (type*)realloc(Ptr, N * S)

Allocate a continuous space that contains N elements with size S, and move the values in Ptr to the new space.

- free(Ptr)

Free the memory space which Ptr is pointed at now. Once you allocate a piece of memory space, they won’t be freed until the program terminates or calls free().

Exercise

Exercise 1

- You need to implement an array with dynamic size.

1. Initialize the array size to be 1*sizeof(int).
2. First input your action.
   - 0 for quitting the program
   - 1 for adding element to the array
   - 2 for removing the last one, you should set that value to -7777
3. If you choose action 1, then you have to input the element you want to add to the array.
4. If the array size is not enough for adding a new element, you should double the size of the array.
5. If the number of elements in the array is less then or equal to half of the size of the array, you need to reduce the array size to half of the size now.
6. Print the whole array.
7. Go back to step 2.

```
action = 1
element to add = 1
whole array = 1
action = 1
element to add = 2
whole array = 1 2
action = 1
element to add = 3
whole array = 1 2 3 -842150451
action = 1
element to add = 4
whole array = 1 2 3 4
action = 1
element to add = 5
whole array = 1 2 3 4 5 -842150451 -842150451 -842150451
action = 1
element to add = 7
whole array = 1 2 3 4 5 7 -842150451 -842150451
action = 1
element to add = 8
whole array = 1 2 3 4 5 7 8 -842150451
action = 2
whole array = 1 2 3 4 5 7 -7777 -842150451
action = 2
whole array = 1 2 3 4 5 -7777 -7777 -842150451
action = 2
whole array = 1 2 3 4
action = 0
```

Exercise Submission Format

Format:

- xxxxxxxxx_ex_w09.zip
  - xxxxxxxxx_ex_01.cpp

xxxxxxxxx is your student ID