計算機程式設計

Computer Programming

Pointers

Instructor: 林英嘉
2024/11/11

W10 Slido: #2306942

GitHub repo

Outline

- Recap of Pointers
- Pointers and Arrays

Recap of Pointers

[Definition & Declaration] Pointers

- A pointer is a **variable** that is used to store the memory address of another variable.
- A pointer also has a data type, which is the type of the pointed variable.
- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
Variable		
Stored value		
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	
Variable		
Stored value		
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

Standard variable

Pointer variable

```
int i = 10;
```

Variable

Variable i

Stored value

Memory address itself

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

Standard variable

```
int i = 10;
```

Pointer variable

Variable	Variable i
Stored value	10
Memory address itself	0x7fffffffdd6c

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

Standard variable

```c
int i = 10;
```

Pointer variable

```c
int *iptr;
iptr = &i;
```

Variable	Variable i
Stored value	10
Memory address itself	0x7fffffffdd6c

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

Standard variable

```c
int i = 10;
```

Pointer variable

```c
int *iptr;
iptr = &i;
```

Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
    - Standard variable: stores a specific data value directly
    - Pointer variable: stores the memory address of another variable

Standard variable

```c
int i = 10;
```

Pointer variable

```c
int *iptr;
iptr = &i;
```

Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

Standard variable

```c
int i = 10;
```

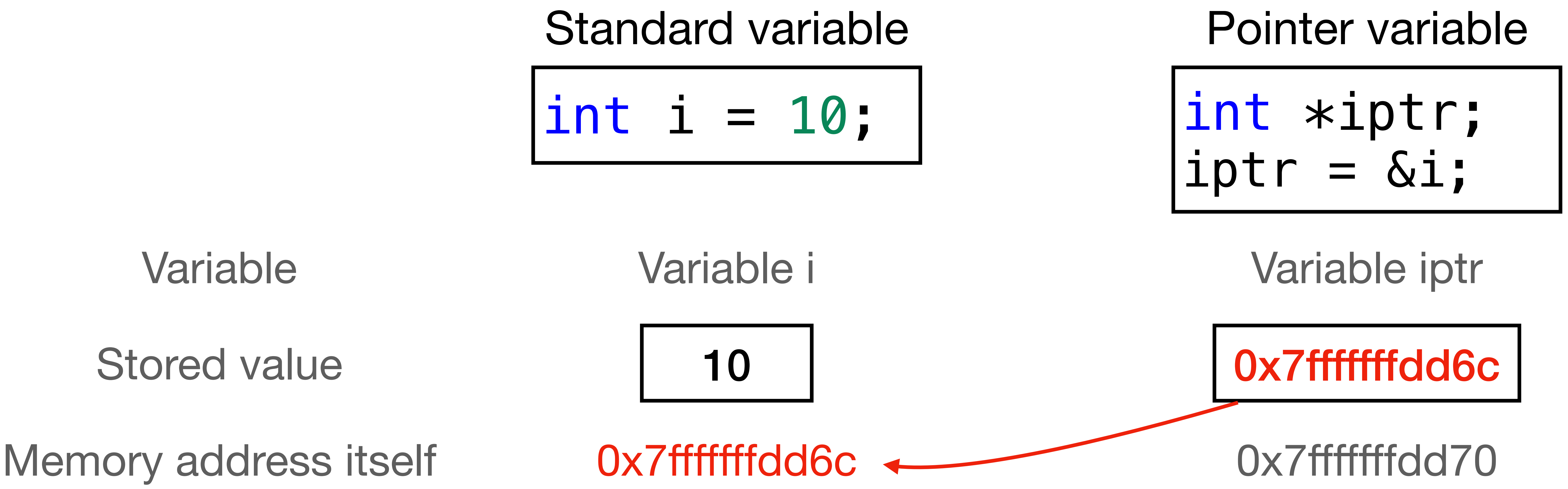
Pointer variable

```c
int *iptr;
iptr = &i;
```

Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable



Standard variable

```
int i = 10;
```

Pointer variable

```
int *iptr;
iptr = &i;
```

Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
Variable		
Stored value		
Memory address itself		

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```
int i = 10;
```

Pointer variable

Variable

Stored value

Memory address itself

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value		
Memory address itself		

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself		

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself	0x7fffffffdd6c	

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```
int i = 10;
```

Pointer variable

```
int *iptr;
iptr = &i;
```

```
int *iptr = &i;
```

Variable	Variable i
Stored value	10
Memory address itself	0x7fffffffdd6c

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```
int i = 10;
```

Pointer variable

```
int *iptr;
iptr = &i;
```

```
int *iptr = &i;
```

Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	

[Declaration] Make a pointer point to a variable in one line

```c
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```c
int i = 10;
```

Pointer variable

```c
int *iptr;
iptr = &i;
```

```c
int *iptr = &i;
```

Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

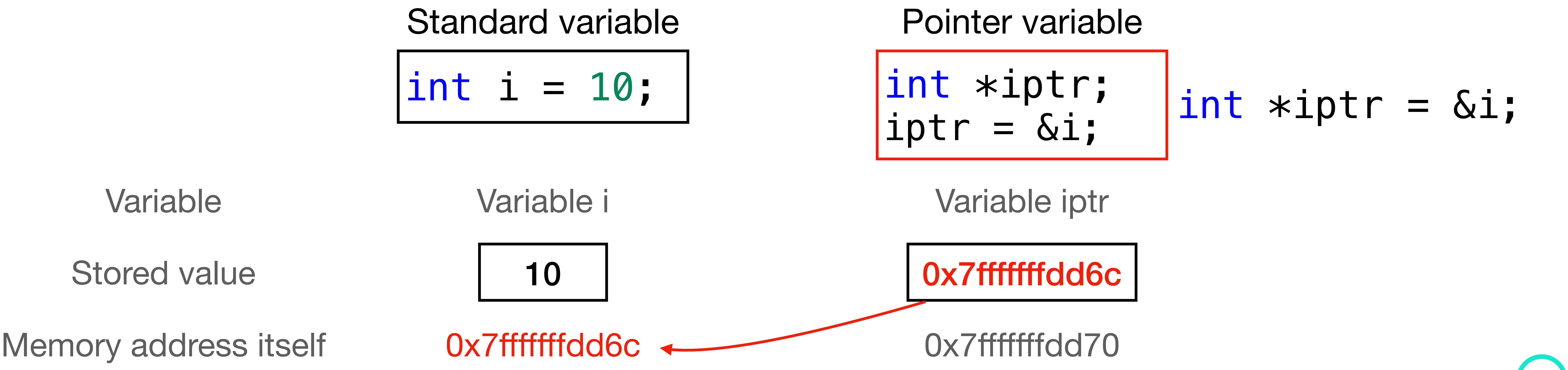
	Standard variable	Pointer variable	

	`int i = 10;`	`int *iptr;`
		`iptr = &i;`
			`int *iptr = &i;`

Variable	Variable i	Variable iptr	
Stored value	10	0x7fffffffdd6c	
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70	

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```



	Standard variable	Pointer variable

	`int i = 10;`	`int *iptr;`
		`iptr = &i;`

Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

```
int *iptr = &i;
```

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

  1. Declaration of a pointer variable

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

1. Declaration of a pointer variable

Standard variable:

```
int p;
```

Pointer variable:

```
int *p;
```

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

1. Declaration of a pointer variable

Standard variable:

```
int p;
```

Pointer variable:

```
int *p;
```

2. Dereference (解除參照)

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

1. Declaration of a pointer variable

Standard variable:

```
int p;
```

Pointer variable:

```
int *p;
```

2. Dereference (解除參照)

We can use an asterisk to a pointer (*p) to obtain the value of the pointed variable. Here, the asterisk is an **indirection** operator (間接運算子).

Use an Asterisk (*) for Dereference

C-course-materials/06-Pointers/dereference.c

2. Dereference (解除參照)

We can use an asterisk to obtain the value of the pointed variable (取值).

```c
#include <stdio.h>
int main(void){
    int *p;
    int i = 10;
    p = &i;
    printf("The value of i is: %d", *p);
}
```

Use an Asterisk (*) for Dereference

C-course-materials/06-Pointers/dereference.c

2. Dereference (解除參照)

We can use an asterisk to obtain the value of the pointed variable (取值).

```
#include <stdio.h>
int main(void){
    int *p;
    int i = 10;
    p = &i;
    printf("The value of i is: %d", *p);
}
```

Assign the address of i to p

Use an Asterisk (*) for Dereference

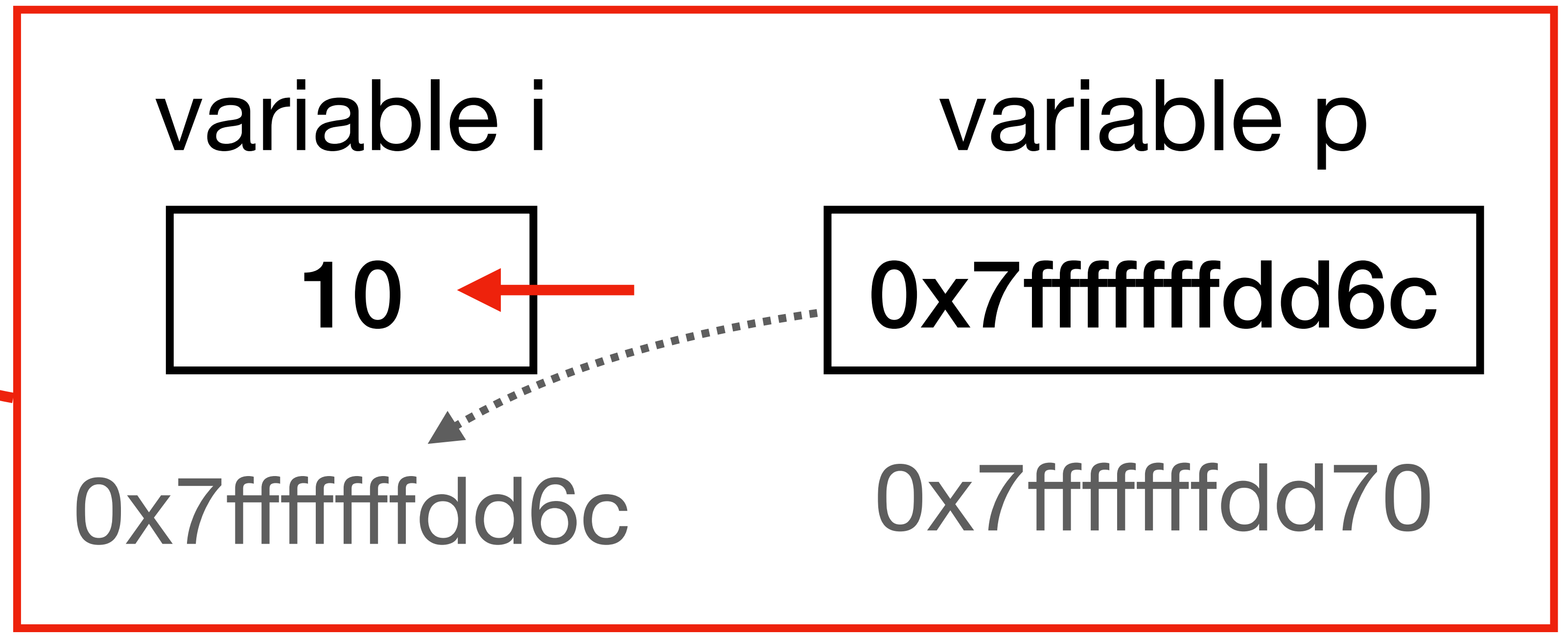
C-course-materials/06-Pointers/dereference.c

2. Dereference (解除參照)

We can use an asterisk to obtain the value of the pointed variable (取值).

```
#include <stdio.h>
int main(void){
    int *p;
    int i = 10;
    p = &i;
    printf("The value of i is: %d", *p);
}
```

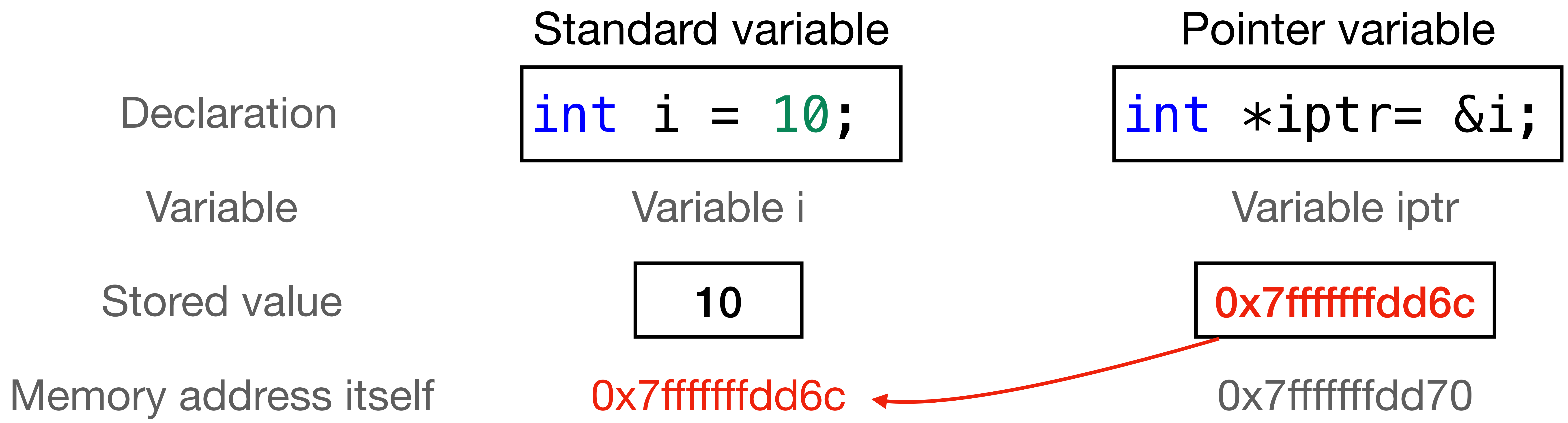
Assign the address of i to p



[Usage] Summary of & and *

- C provides a pair of operators designed specifically for use with pointers.
  - (取址) To find the address of a variable, we use the & (address) operator.
  - (取值) To gain access to the object that a pointer points to, we use the * (indirection) operator.

[Usage] Summary of stored values



	Standard variable	Pointer variable
Declaration	`int i = 10;`	`int *iptr= &i;`
Variable	Variable i	Variable iptr
Stored value	10	0x7ffffffdd6c
Memory address itself	0x7ffffffdd6c	0x7ffffffdd70

		位置運算子		位置運算子	間接運算子
	i	&i	iptr	&iptr	*iptr
Value	10	0x7ffffffdd6c	0x7ffffffdd6c	0x7ffffffdd70	10

Same

Practical Properties of Pointers

[Important Notes] Properties of Pointers

- Two pointers can point the same variable.
- A pointer can be redirected to point other variables.
    - The memory address of a variable itself cannot be changed
- Generally, a pointer with a specific data type can only point to the variable with the same data type

Complicated Pointer Operations

C-course-materials/06-Pointers/complicated_pointer_ops.c

```
#include <stdio.h>
int main(void){
    int i = 10;
    int *iptr = &i;
    // Operation 1
    printf("&(*iptr) = %p\n", &(*iptr));
    // Operation 2
    printf("*(&iptr) = %p\n", *(&iptr));
    // Operation 3
    printf("*(*(&iptr)) = %d\n", *(*(&iptr)));
    // Operation 4
    printf("*(&(*iptr)) = %d\n", *(&(*iptr)));
    // Operation 5
    printf("&(*(&iptr)) = %p\n", &(*(&iptr)));
}
```

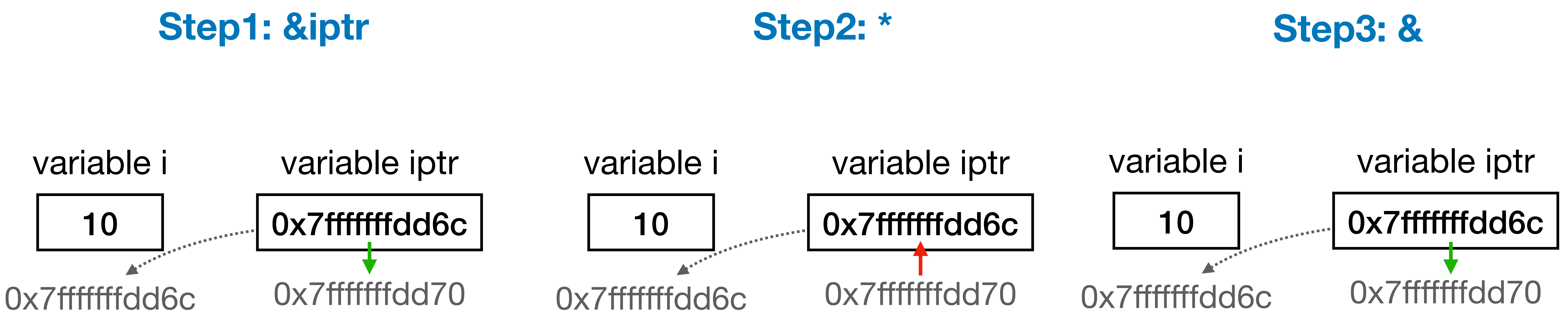
[Illustration] Operation 5

C-course-materials/06-Pointers/complicated_pointer_ops.c

```
printf("&(*(&iptr)) = %p\n", &(*(&iptr)));
```

The output is **the value of the pointed variable**.

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path



運算子優先順序說明

C-course-materials/slides/data_types_op.pdf (p.37)

優先順序	Operator	Meaning	連在一起用？
1	( )	大於	由左至右
2	[ ]	小於	由左至右
3	! + - * &	非、取正負、解除參照、取位址	由右至左
4	++ --	遞增、遞減	由右至左
5	* / %	算數運算子	由左至右
6	+ -	算數運算子	由左至右
7	> >= < <=	關係運算子	由左至右
8	= = !=	關係運算子	由左至右
9	&&	邏輯運算子	由左至右
10	||	邏輯運算子	由左至右
11	=	設定運算子	由右至左

Pointer Sizes

C-course-materials/06-Pointers/pointer_sizes.c

```
#include <stdio.h>
int main(void){
    int i = 10;
    float f = 10.0;
    double d = 10.0;
    int *iptr = &i;
    float *fptr = &f;
    double *dptr = &d;
    printf("Size of the int pointer p is: %d\n", sizeof(iptr));
    printf("Size of the value of the pointed varaiable is: %d\n", sizeof(*iptr));
    printf("Size of the float pointer p is: %d\n", sizeof(fptr));
    printf("Size of the value of the pointed varaiable is: %d\n", sizeof(*fptr));
    printf("Size of the double pointer p is: %d\n", sizeof(dptr));
    printf("Size of the value of the pointed varaiable is: %d", sizeof(*dptr));
}
```

On a 64-bit system, a pointer has a size of 8 bytes for all data types (1 byte = 8 bits.)

This is because that a pointer stores a **memory address** of its pointed variable, and

a memory address is **64 bits wide** on a 64-bit system.

Pointers and Arrays

[Declaration] Pointer to an Array

C-course-materials/06-Pointers-arrays/declaration_and_print.c

We can create a pointer to an array with any initial address of an pointer:

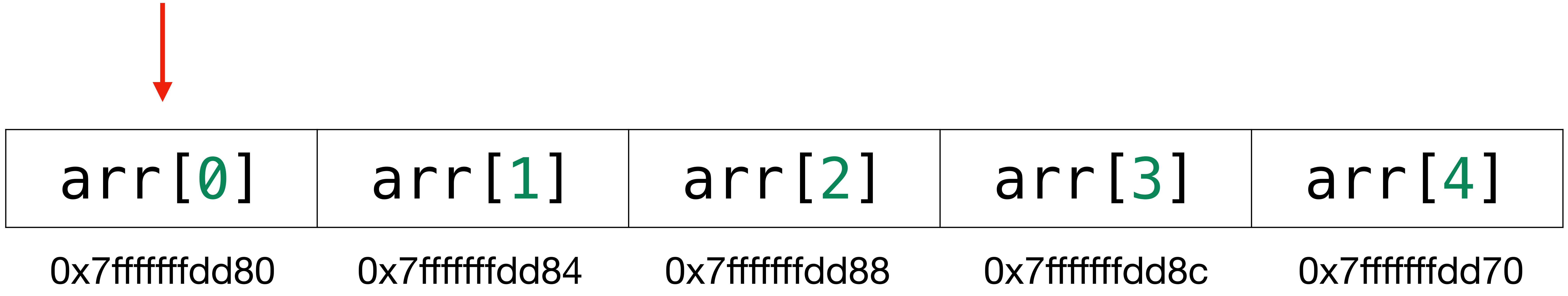
```
int arr[5] = {1, 2, 3, 4, 5}, *p;
p = &arr[0];
```

Or you can create a pointer for an array in one line:

```
int arr[5] = {1, 2, 3, 4, 5};
int *p = &arr[0];
```

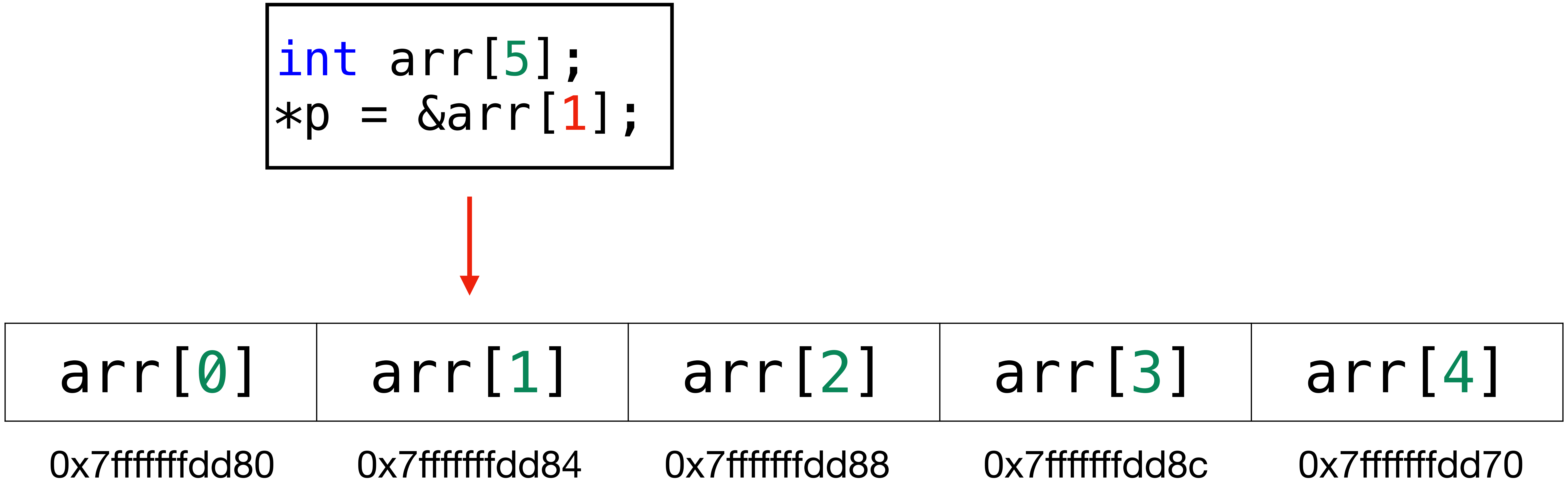
[Illustration] Pointer for an Array

```
int arr[5];
*p = &arr[0];
```



arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd70

[Illustration] Pointer for an Array



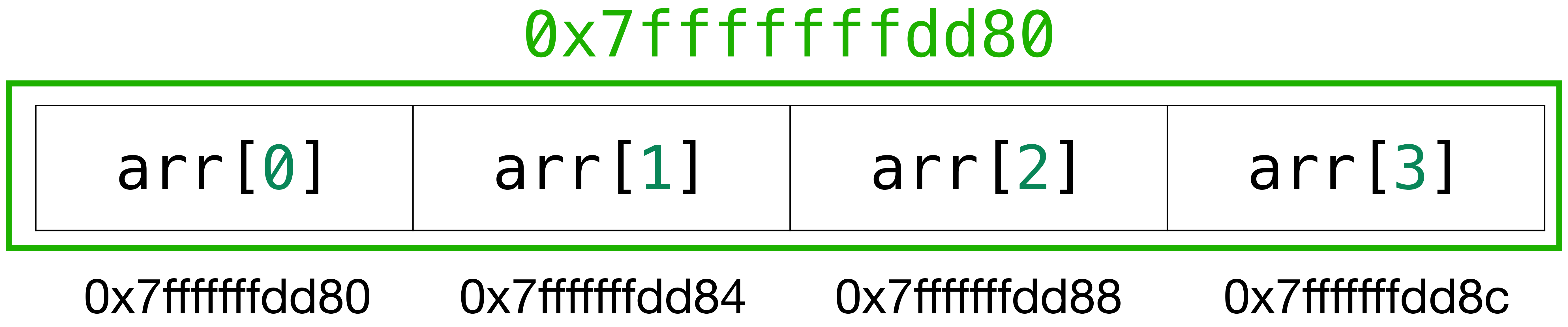
```
int arr[5];
*p = &arr[1];
```

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd70

[Recap] Properties of Array Address

C-course-materials/slides/arrays.pdf (p60)
C-course-materials/slides/functions_2.pdf (p17)

- The first element of an array shares the address of the array.
- The memory addresses of an array in C are contiguous. For an int array:
    - arr[0] is at address A
    - arr[1] is at address A + 4
    - arr[2] is at address A + 8
    - arr[3] is at address A + 12



0x7fffffffdd80

arr[0]	arr[1]	arr[2]	arr[3]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c

[Declaration] Pointer to an Array

C-course-materials/06-Pointers-arrays/declaration_and_print.c

We can create a pointer to an array with any initial address of an pointer:

```
int arr[5] = {1, 2, 3, 4, 5}, *p;
p = &arr[0];
```

equal to `p = arr;`

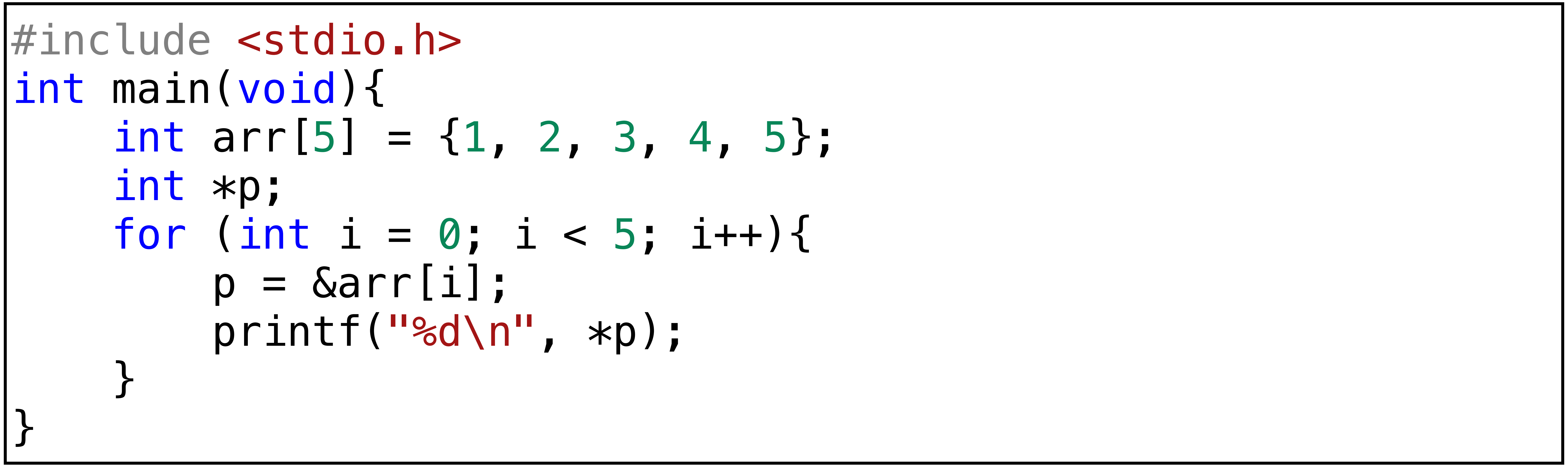
Or you can create a pointer for an array in one line:

```
int arr[5] = {1, 2, 3, 4, 5};
int *p = &arr[0];
```

equal to `int *p = arr;`

Print an Array with a Pointer

C-course-materials/06-Pointers-arrays/print_1D.c



```c
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p;
    for (int i = 0; i < 5; i++){
        p = &arr[i];
        printf("%d\n", *p);
    }
}
```

Print an Array with a Pointer

C-course-materials/06-Pointers-arrays/print_1D.c

```c
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p;
    for (int i = 0; i < 5; i++){
        p = &arr[i];
        printf("%d\n", *p);
    }
}
```

- i will increase (from zero to four)
- p points to an address of an array element for each time

Dereference of a Pointer Pointed to an Array

C-course-materials/06-Pointers-arrays/dereference.c

```c
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p = arr;
    printf("arr[0]: %d\n", *p);
    p = &arr[1];
    printf("arr[1]: %d\n", *p);
    p = &arr[2];
    printf("arr[2]: %d\n", *p);
}
```

Output

```
arr[0]: 1
arr[1]: 2
arr[2]: 3
```

Pointers Arithmetic

(指標的算數運算)

[Definition] Pointer Arithmetics

- Pointer arithmetics indicate arithmetic operations (as following) to pointers by performing addition or subtraction to the stored memory address.

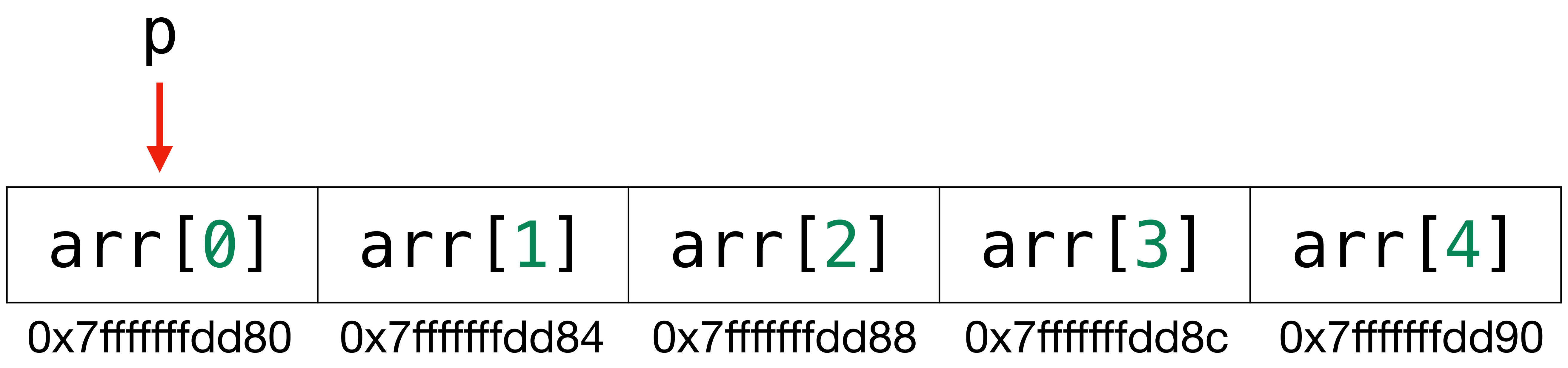
Pointer Arithmetics	Example	Result
Adding an integer[1] to a pointer	iptr + 1	Address
Subtracting an integer[1] from a pointer	iptr - 1	Address
Subtracting one pointer from another	iptr_2 - iptr_1	Difference in number of elements
Pointer comparison	iptr_1 <= iptr_2	True (1) or False (0)

[1] Only integers are allowed.

[Illustration] Adding an integer to a pointer

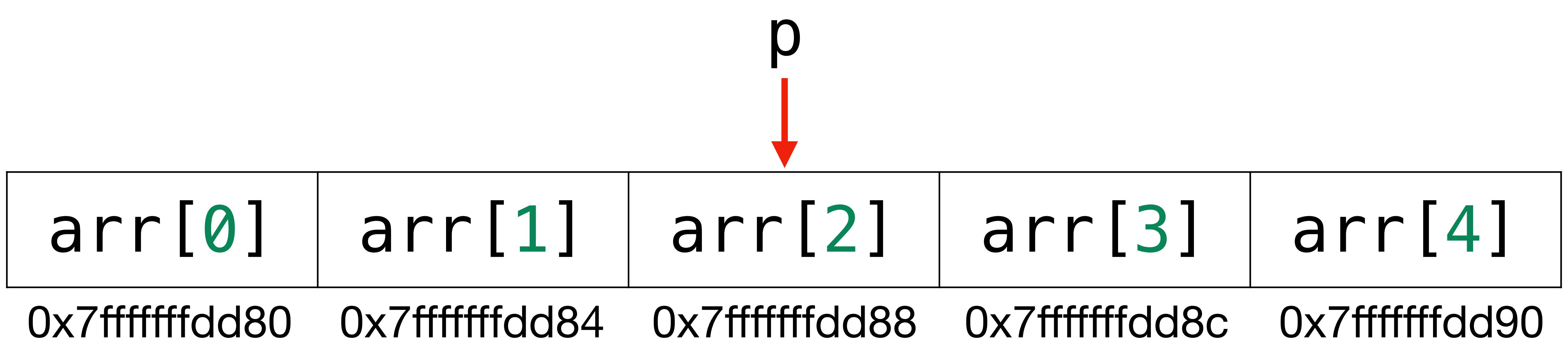
```
int arr[5], *p, *q;
```

```
p = &arr[0];
```



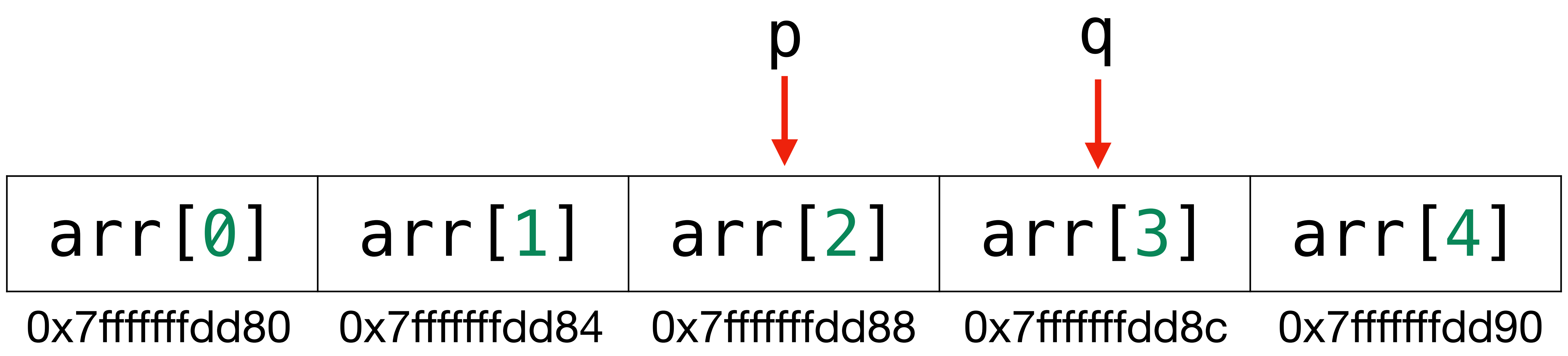
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

```
p = p + 2;
```



arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

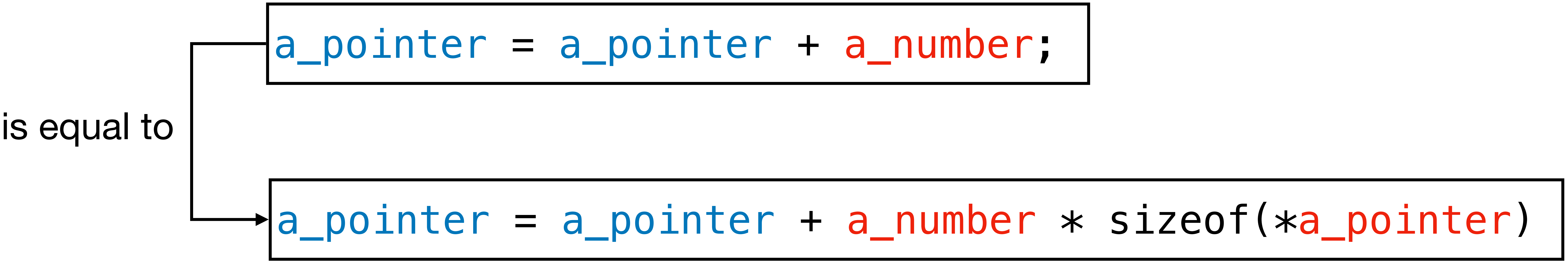
```
q = p + 1;
```



arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

[Illustration] Internal Procedures of Pointer Arithmetics

- Take “Adding an integer to a pointer” as an example:



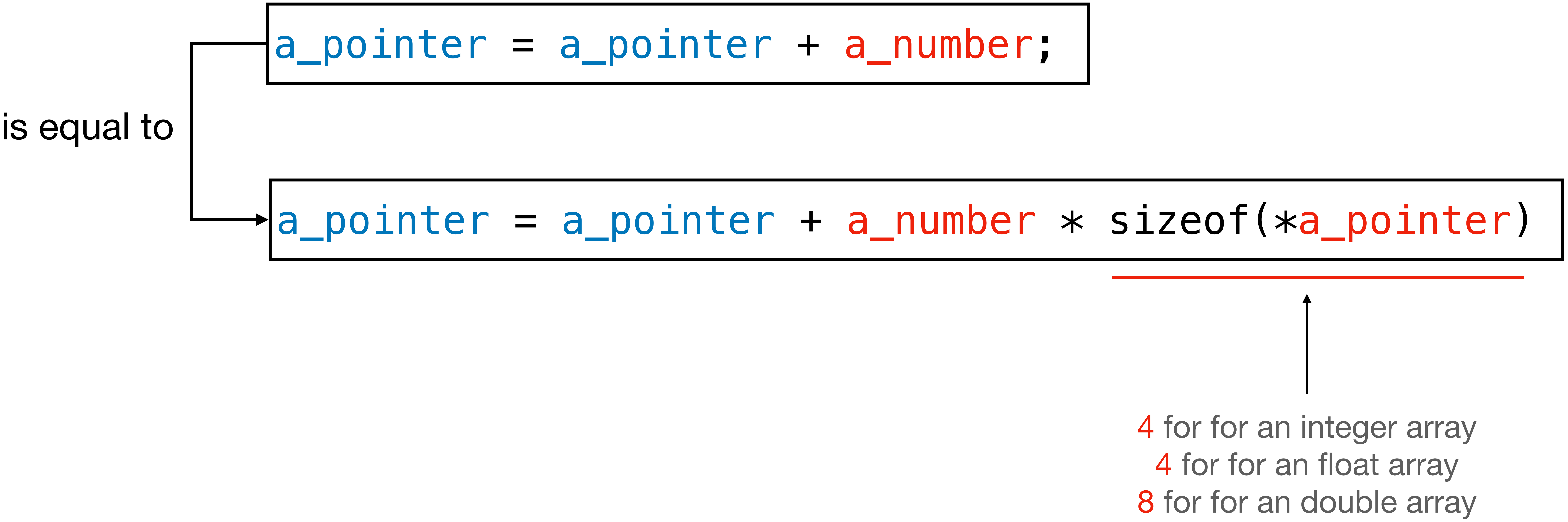
```
a_pointer = a_pointer + a_number;
```

is equal to

```
a_pointer = a_pointer + a_number * sizeof(*a_pointer)
```

[Illustration] Internal Procedures of Pointer Arithmetics

- Take “Adding an integer to a pointer” as an example:



```
a_pointer = a_pointer + a_number;
```

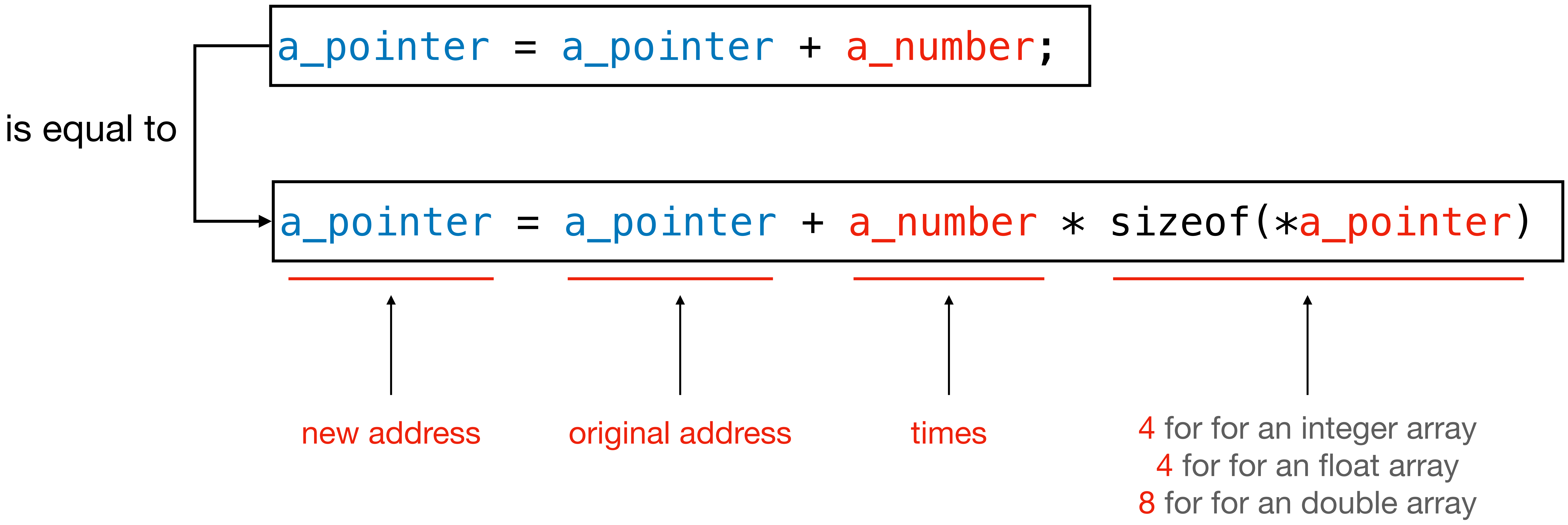
is equal to

```
a_pointer = a_pointer + a_number * sizeof(*a_pointer)
```

4 for for an integer array
4 for for an float array
8 for for an double array

[Illustration] Internal Procedures of Pointer Arithmetics

- Take “Adding an integer to a pointer” as an example:



```
a_pointer = a_pointer + a_number;
```

is equal to

```
a_pointer = a_pointer + a_number * sizeof(*a_pointer)
```

new address

original address

times

4 for for an integer array
4 for for an float array
8 for for an double array

Print an Array via Addition

C-course-materials/06-Pointers-arrays/print_1D_arithmetic.c
C-course-materials/06-Pointers-arrays/print_1D_increment.c

```
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p;
    for (int i = 0; i < 5; i++){
        p = &arr[i];
        printf("%d\n", *p);
    }
}
```

Output

```
1
2
3
4
5
```

```
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p = &arr[0];
    for (int i = 0; i < 5; i++){
        printf("%p\n", *p+i);
    }
}
```

Add an integer to the pointer each time
But the pointer p does not change in this case

Print an Array via Increment

C-course-materials/06-Pointers-arrays/print_1D.c
C-course-materials/06-Pointers-arrays/print_1D_increment.c

```c
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p;
    for (int i = 0; i < 5; i++){
        p = &arr[i];
        printf("%d\n", *p);
    }
}
```

Output

```
1
2
3
4
5
```

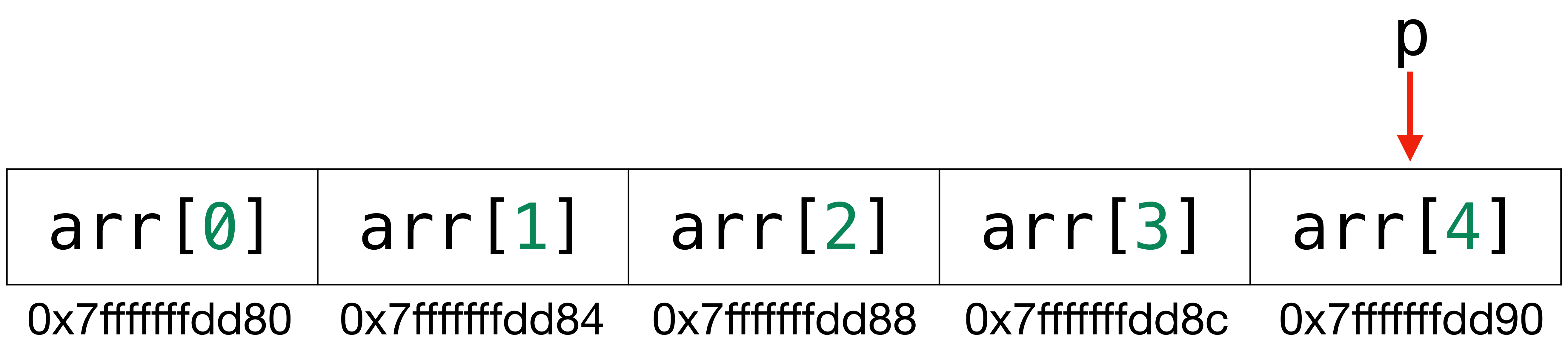
```c
#include <stdio.h>
int main(void){
    int arr[5] = {1, 2, 3, 4, 5};
    int *p = &arr[0];
    for (int i = 0; i < 5; i++){
        printf("%d\n", *p);
        p++;
    }
}
```

Set the pointer from the beginning of an array

[Illustration] Subtracting an integer from a pointer

```
int arr[5], *p, *q;
```

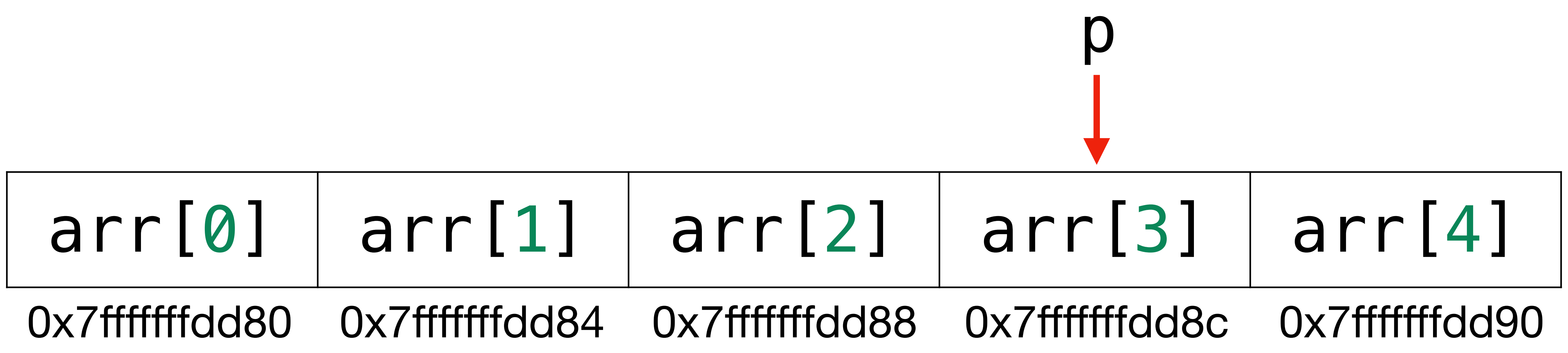
```
p = &arr[4];
```



p

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

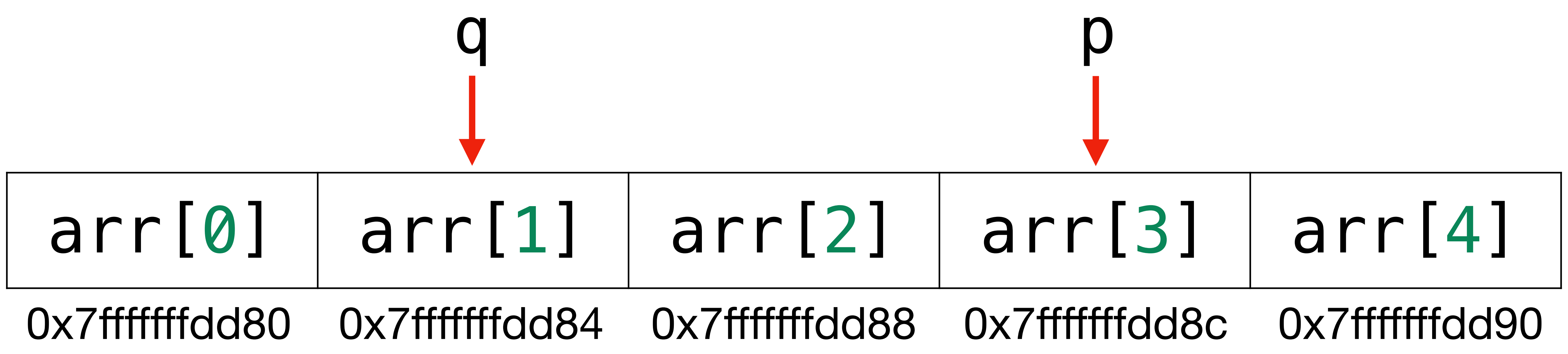
```
p = p - 1;
```



p

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

```
q = p - 2;
```



q

p

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

[Illustration] Internal Procedures of Pointer Arithmetics

- Take “Subtracting an integer to a pointer” as an example:

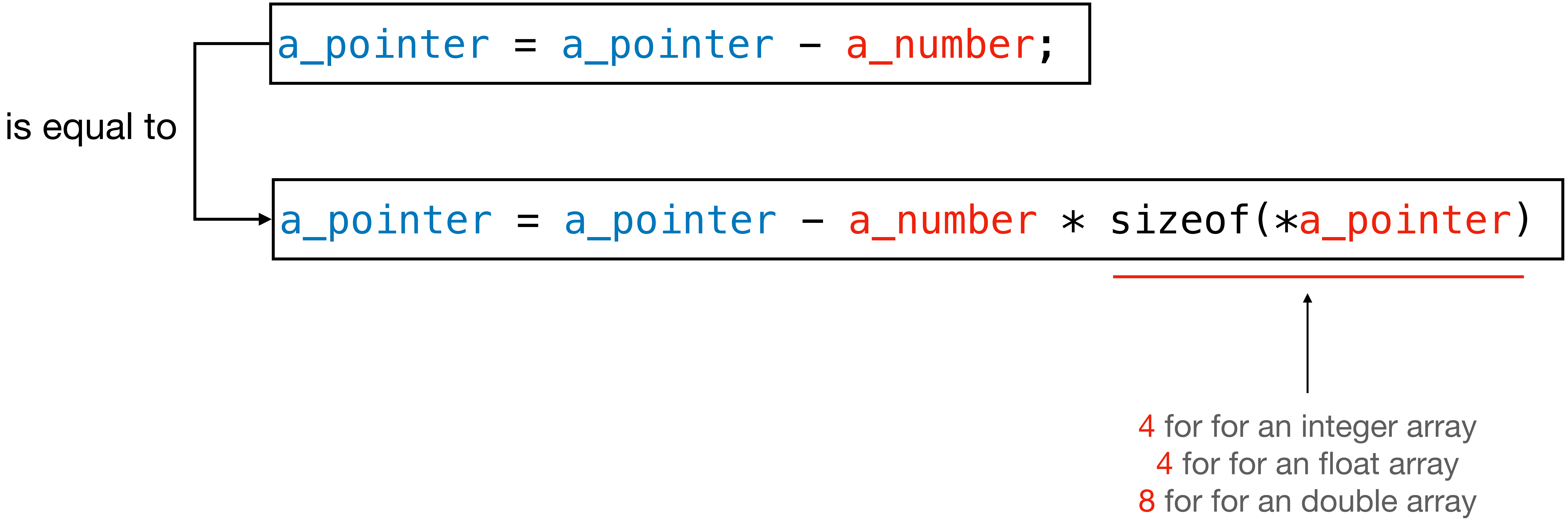
```
a_pointer = a_pointer - a_number;
```

is equal to

```
a_pointer = a_pointer - a_number * sizeof(*a_pointer)
```

[Illustration] Internal Procedures of Pointer Arithmetics

- Take “Subtracting an integer to a pointer” as an example:



```
a_pointer = a_pointer - a_number;
```

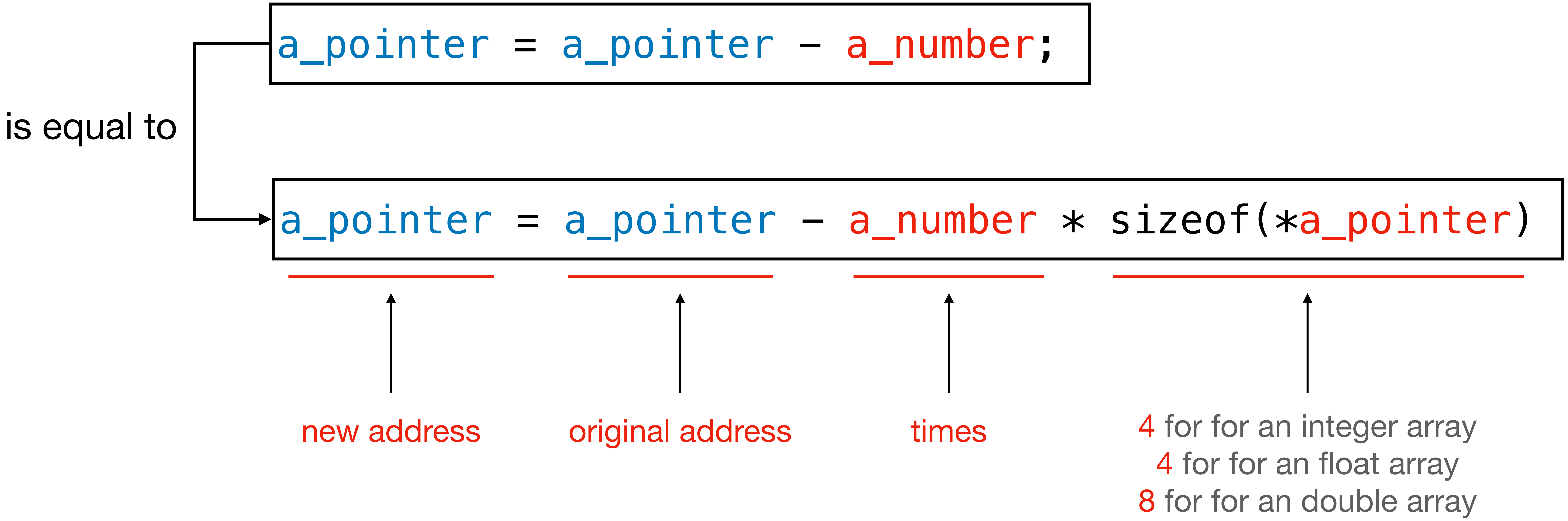
is equal to

```
a_pointer = a_pointer - a_number * sizeof(*a_pointer)
```

4 for for an integer array
4 for for an float array
8 for for an double array

[Illustration] Internal Procedures of Pointer Arithmetics

- Take “Subtracting an integer to a pointer” as an example:



```
a_pointer = a_pointer - a_number;
```

is equal to

```
a_pointer = a_pointer - a_number * sizeof(*a_pointer)
```

new address

original address

times

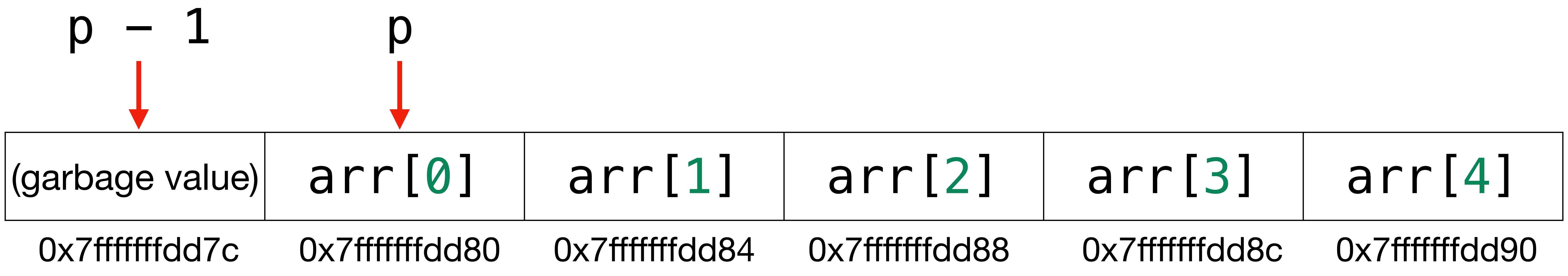
4 for for an integer array
4 for for an float array
8 for for an double array

Address can be negative

C-course-materials/06-Pointers-arrays/arr_subtraction_neg.c

```c
#include <stdio.h>
int main(void){
    int arr[5] = {2, 3, 5, 9, 10};
    int *p = &arr[0];
    printf("%p\n", p);
    printf("%p\n", p - 1);
    printf("%d\n", *(p - 1));
}
```

*(p - 1) is different from *p - 1



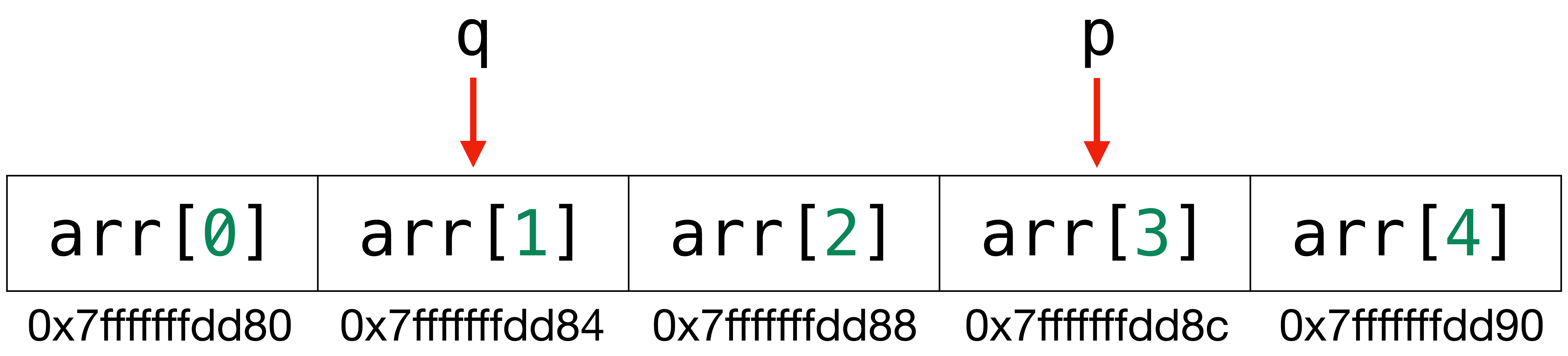
p – 1
p

(garbage value)	arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
0x7fffffffdd7c	0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd8c	0x7fffffffdd90

[Illustration] Subtracting one pointer from another

```
int arr[5], *p, *q;
```

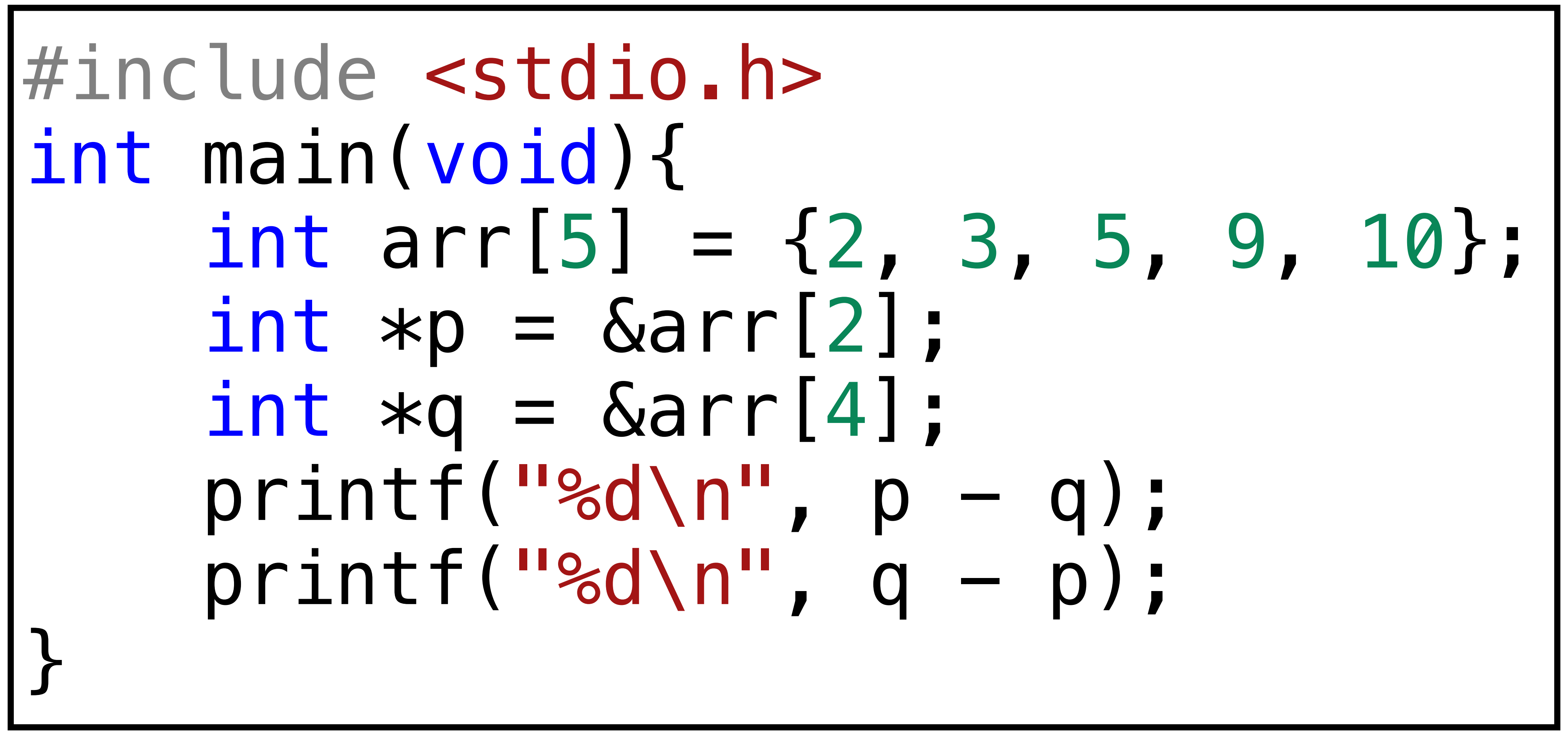
```
p = &arr[3];
q = &arr[1];
```



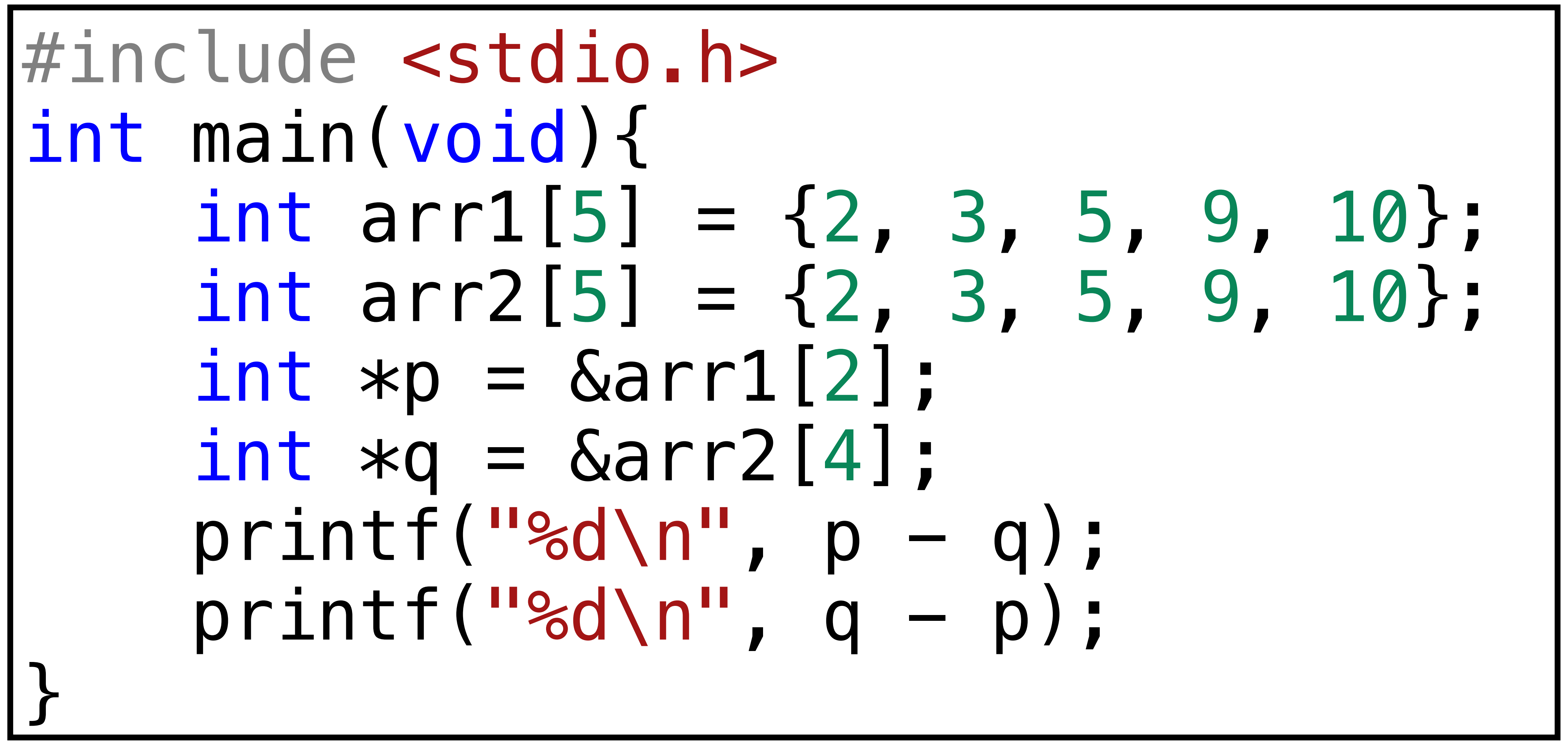
- Different from adding or subtracting an integer to a pointer, the result of subtracting one pointer from another is **distance**.
- In this case:
  - the result of p - q is 2.
  - the result of q - p is -2.

Subtracting one pointer from another

C-course-materials/06-Pointers-arrays/arr_pointer_difference.c
C-course-materials/06-Pointers-arrays/arr_pointer_difference_wrong.c



```c
#include <stdio.h>
int main(void){
    int arr[5] = {2, 3, 5, 9, 10};
    int *p = &arr[2];
    int *q = &arr[4];
    printf("%d\n", p - q);
    printf("%d\n", q - p);
}
```



```c
#include <stdio.h>
int main(void){
    int arr1[5] = {2, 3, 5, 9, 10};
    int arr2[5] = {2, 3, 5, 9, 10};
    int *p = &arr1[2];
    int *q = &arr2[4];
    printf("%d\n", p - q);
    printf("%d\n", q - p);
}
```

- If two pointers from different arrays, the result of subtraction will be wrong.

Address Arithmetic for int variables

C-course-materials/06-Pointers-arrays/int_addr_arithmetic.c

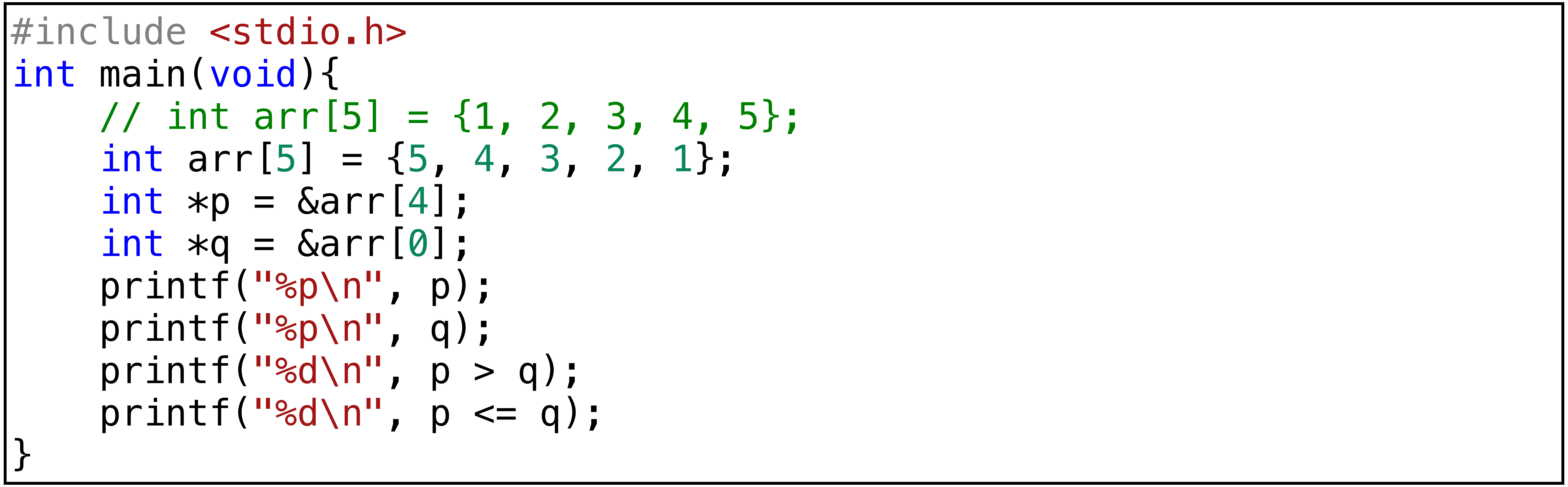
```c
#include <stdio.h>
int main(void){
    int int_a;
    printf("%p\n", &int_a);
    printf("%p\n", &int_a+1);
    printf("%p\n", &int_a+2);
}
```

Output

```
0x7fffffffdd64
0x7fffffffdd68
0x7fffffffdd6c
```

Comparing Pointers

C-course-materials/06-Pointers-arrays/comparing_pointers.c



```c
#include <stdio.h>
int main(void){
    // int arr[5] = {1, 2, 3, 4, 5};
    int arr[5] = {5, 4, 3, 2, 1};
    int *p = &arr[4];
    int *q = &arr[0];
    printf("%p\n", p);
    printf("%p\n", q);
    printf("%d\n", p > q);
    printf("%d\n", p <= q);
}
```

[Important Notes] Pointer Arithmetics

- C does not support “adding one pointer from another”.
- Pointer arithmetics are not strictly limited to arrays, but only safe and well-defined within the boundaries of a contiguous memory block such as arrays.

Input pointers of arrays to functions

[Declaration] Input Array to a Function

prototype

```
return_type func_name(type arr[], type2 param2, ...)
```

Not required

main function

```
int main(void){
 array_declaration;
 ...
}
```

custom function

```
return_type func_name(type arr[], type2 param2, ...){
   body;
   return value;
}
```

Not required

[Declaration] Input a Pointer to a Function

- A function prototype is:

```
return_type func_name(type1 *, type2 *, ...);
```

- Purposes:

1. **Type Checking:** Help the compiler **check the correctness of data types** when you use a function in the main function.
2. **Function Declaration**: Allows function calls before the function is defined.

- You can also write a function prototype as the following to increase readability:

```
return_type func_name(type1 *param1, type2 *param2, ...);
```

[Declaration] Input a Pointer of an Array to a Function

- A function prototype is:

```
return_type func_name(type1 *, type2 *, ...);
```

- Purposes:

1. **Type Checking:** Help the compiler **check the correctness of data types** when you use a function in the main function.
2. **Function Declaration**: Allows function calls before the function is defined.

- You can also write a function prototype as the following to increase readability:

```
return_type func_name(type1 *arr1, int size1, ...);
```

W9 Quiz: Sorting Values Using Pointers

Please write a program for using pointers to sort four integer variables a, b, c, and d in ascending order.

Initial Values: Given the following values for four integer variables:

```
int a = 99, b = 35, c = 34, d = 97;
```

Requirements

1. Implement a function called sorting to take pointers as arguments and sort the values in ascending order. You can swap two pointers at each time. You can also sort all four values in one function call. The former is preferred and easier.
2. Value modifications must be done using pointers.

[Recap] Swap two variables inside a function

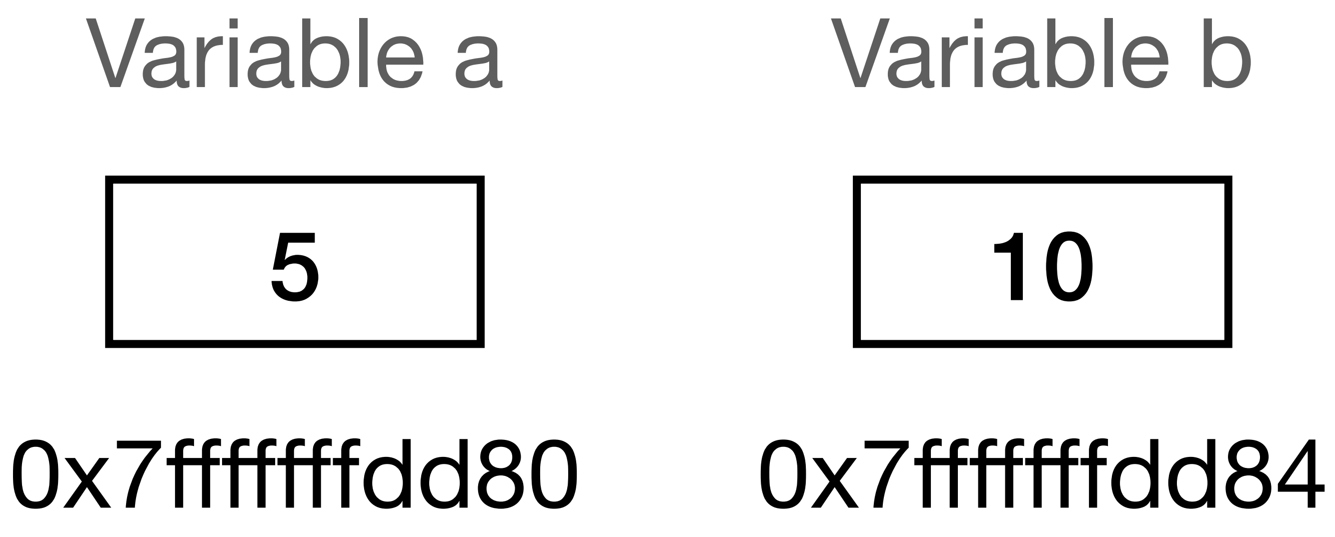
C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

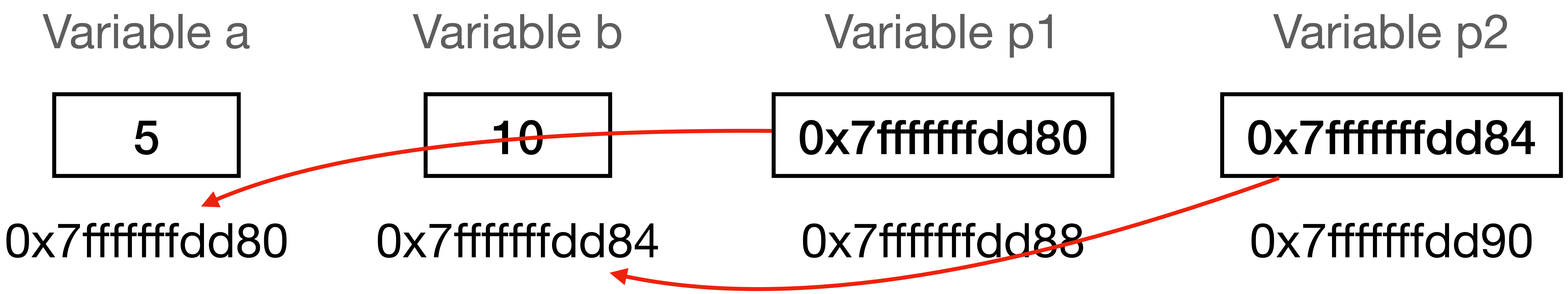
```
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

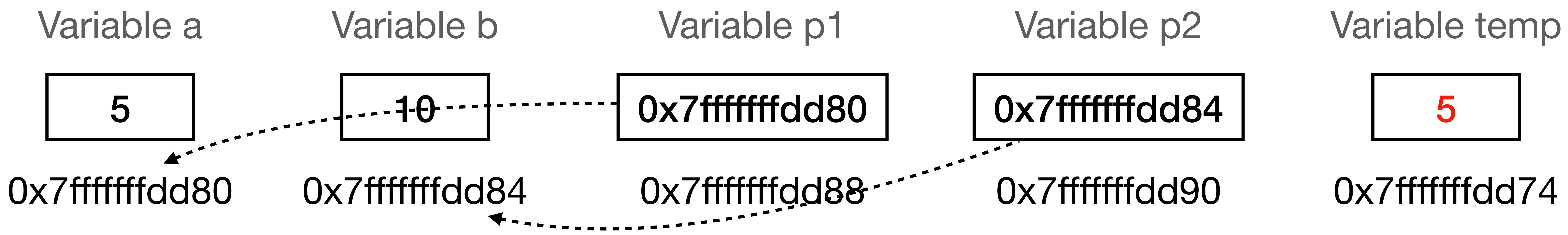


Variable a	Variable b	Variable p1	Variable p2
5	10	0x7ffffffdd80	0x7ffffffdd84
0x7ffffffdd80	0x7ffffffdd84	0x7ffffffdd88	0x7ffffffdd90

[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

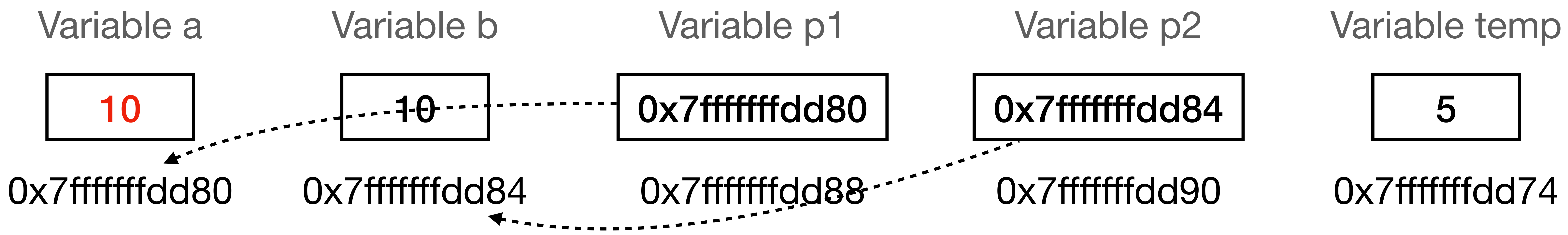


Variable a	Variable b	Variable p1	Variable p2	Variable temp
5	10	0x7fffffffdd80	0x7fffffffdd84	5
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd90	0x7fffffffdd74

[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

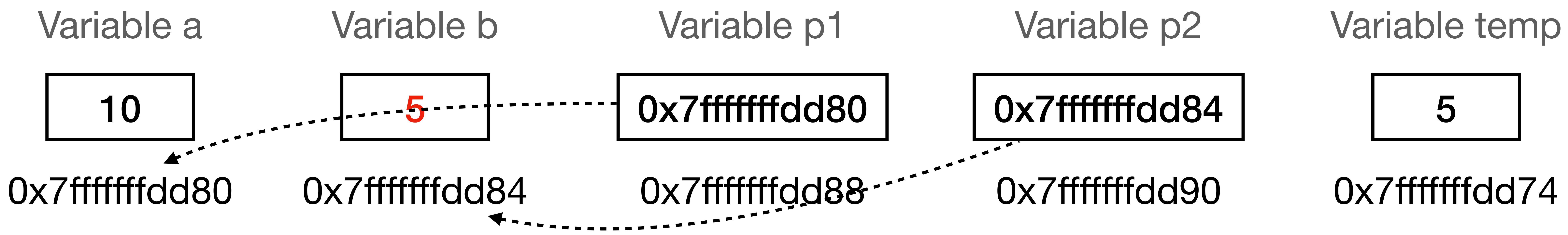


Variable a	Variable b	Variable p1	Variable p2	Variable temp
10	10	0x7fffffffdd80	0x7fffffffdd84	5
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd90	0x7fffffffdd74

[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

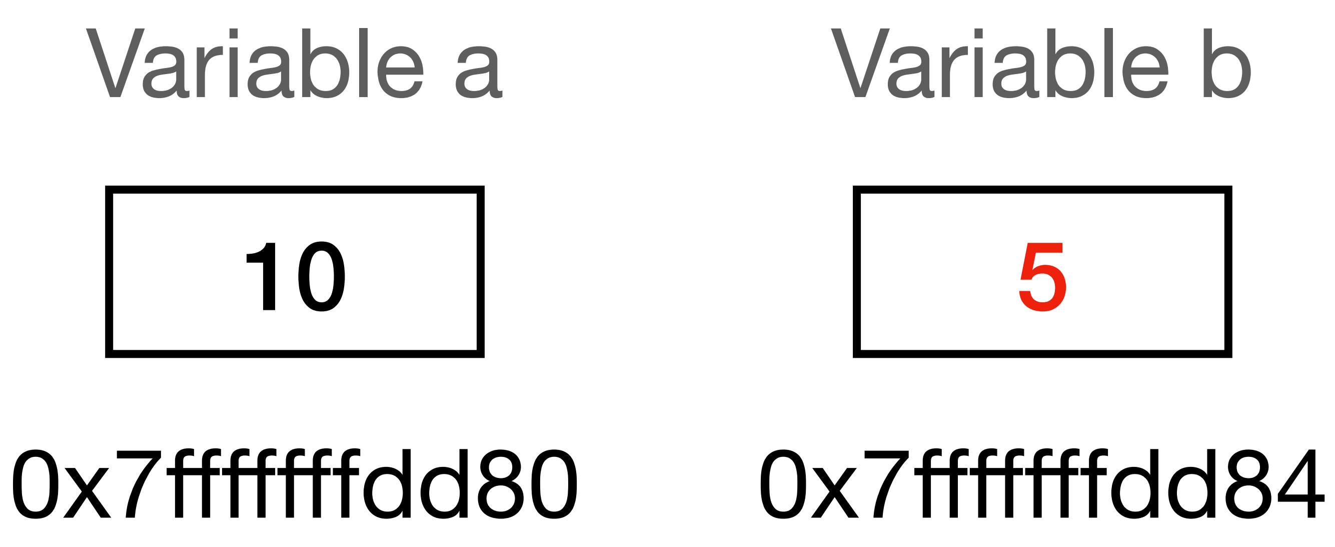


Variable a	Variable b	Variable p1	Variable p2	Variable temp
10	5	0x7fffffffdd80	0x7fffffffdd84	5
0x7fffffffdd80	0x7fffffffdd84	0x7fffffffdd88	0x7fffffffdd90	0x7fffffffdd74

[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



Variable a

10

0x7fffffffdd80

Variable b

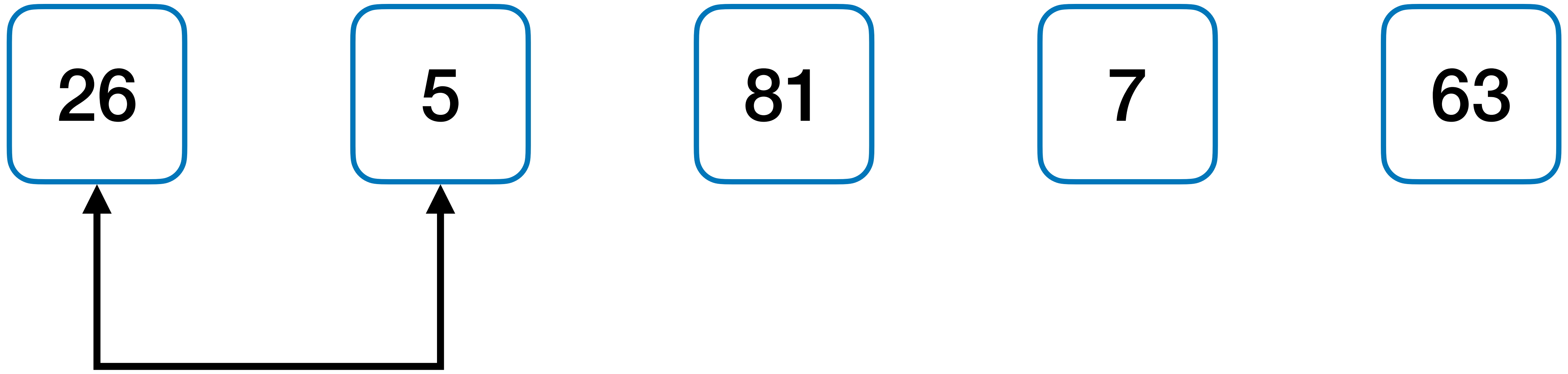
5

0x7fffffffdd84

[Illustration] Bubble Sort

C-course-materials/slides/arrays.pdf (p25)

- We need to sort the sequence for an increasing order. (small to big)

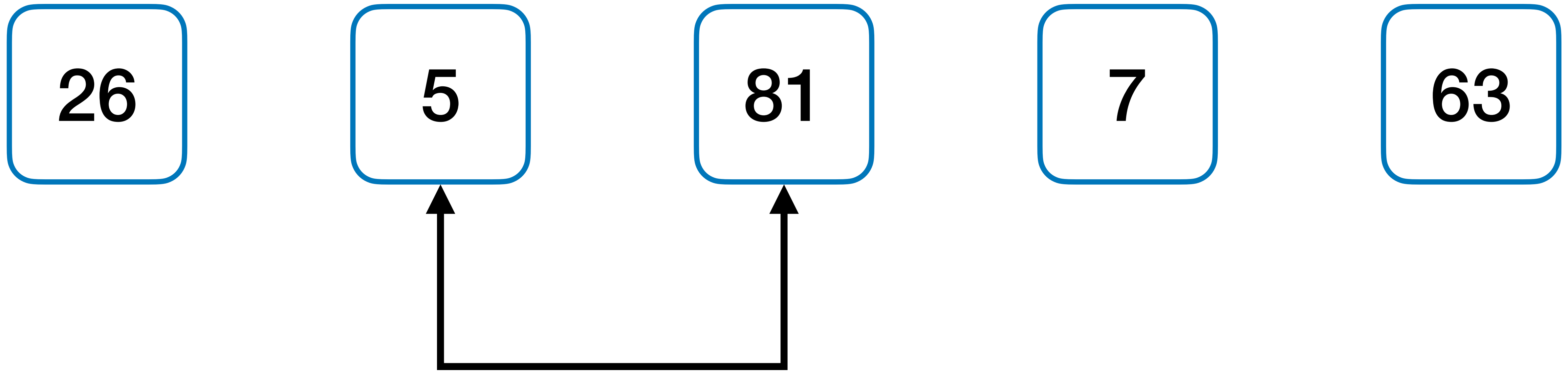


26 5 81 7 63

[Illustration] Bubble Sort

C-course-materials/slides/arrays.pdf (p25)

- We need to sort the sequence for an increasing order. (small to big)

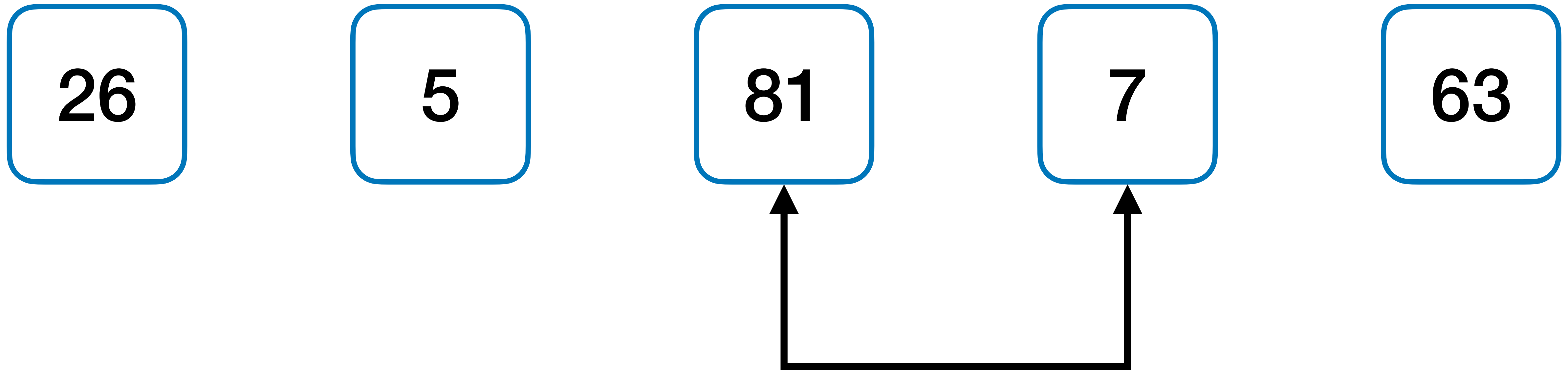


26 5 81 7 63

[Illustration] Bubble Sort

C-course-materials/slides/arrays.pdf (p25)

- We need to sort the sequence for an increasing order. (small to big)

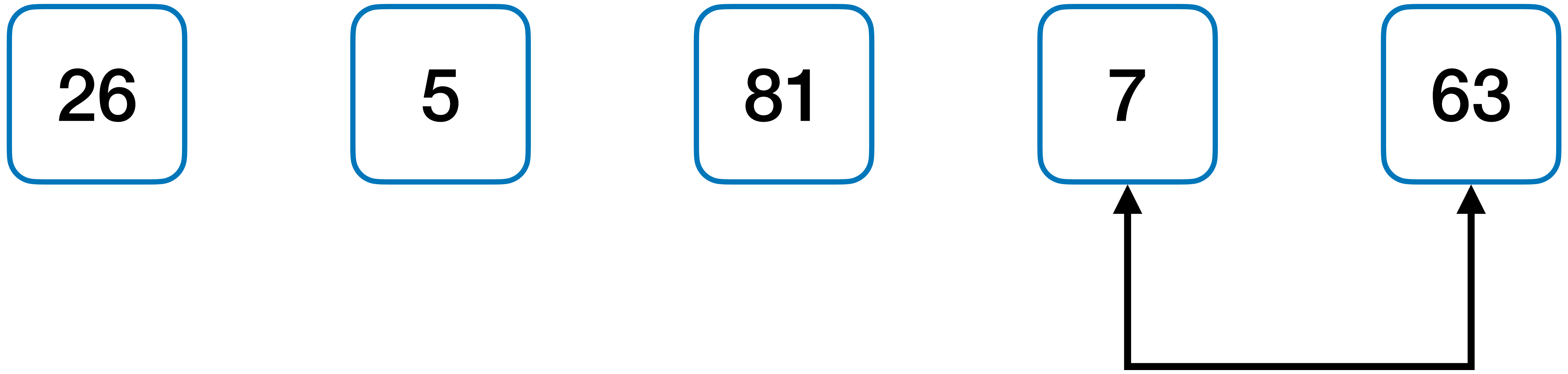


26 5 81 7 63

[Illustration] Bubble Sort

C-course-materials/slides/arrays.pdf (p25)

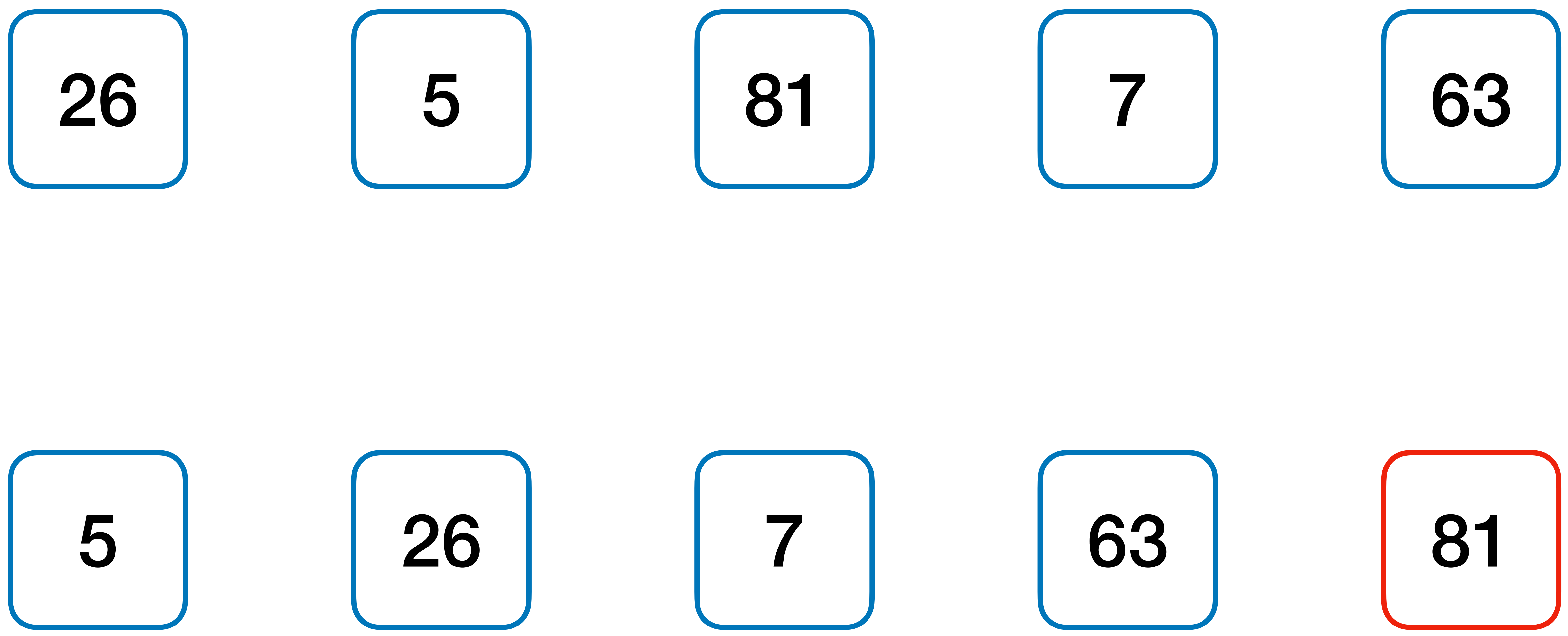
- We need to sort the sequence for an increasing order. (small to big)



[Illustration] Bubble Sort

C-course-materials/slides/arrays.pdf (p25)

- We need to sort the sequence for an increasing order. (small to big)



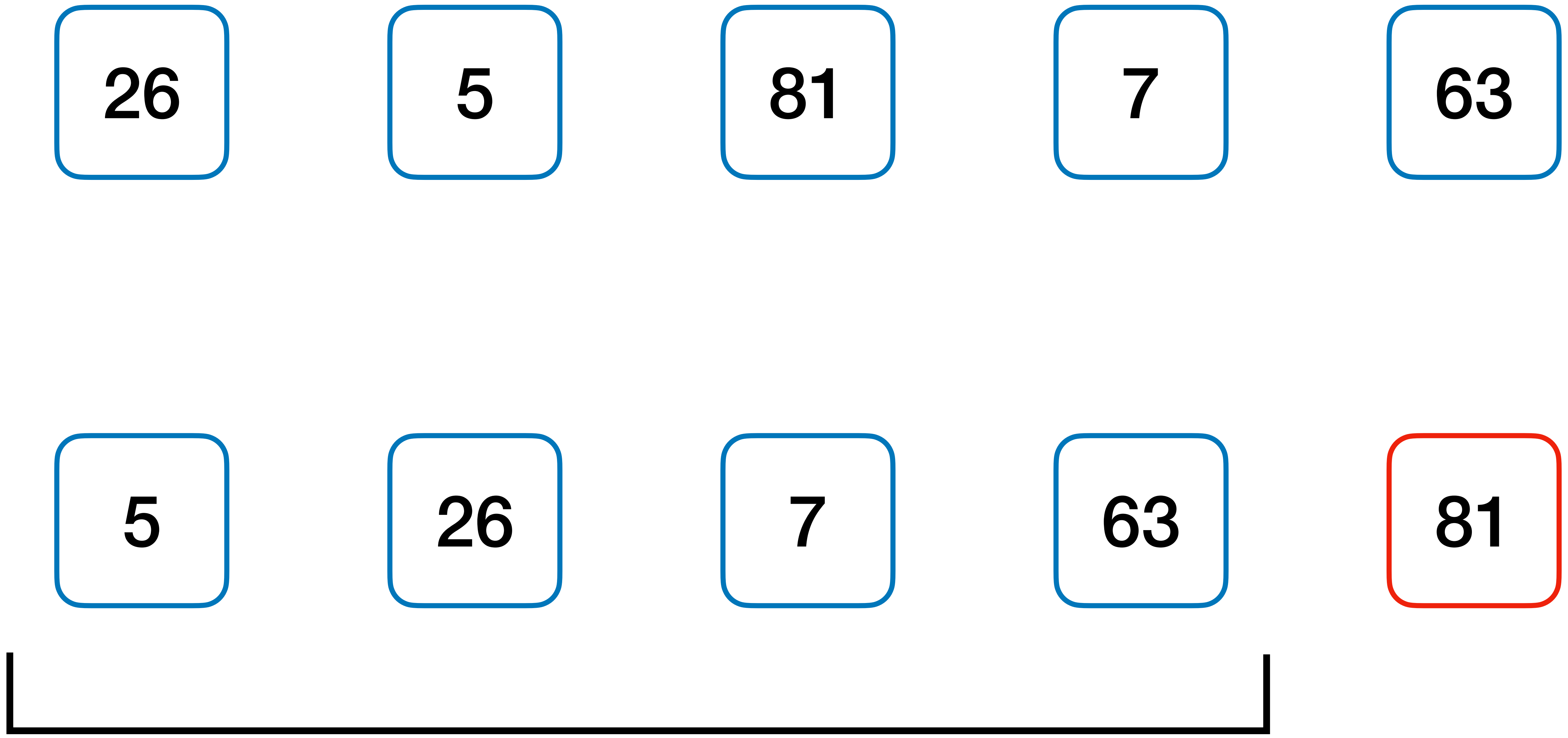
26 5 81 7 63

5 26 7 63 81

[Illustration] Bubble Sort

C-course-materials/slides/arrays.pdf (p25)

- We need to sort the sequence for an increasing order. (small to big)



26 5 81 7 63

5 26 7 63 81

Next: Only compare the items except the most right one

Manual Bubble Sort

```c
int swap_all(int *a, int *b, int *c, int *d) {
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*c > *d) {
        swap(c, d);
    }
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*a > *b) {
        swap(a, b);
    }
}
```

Manual Bubble Sort

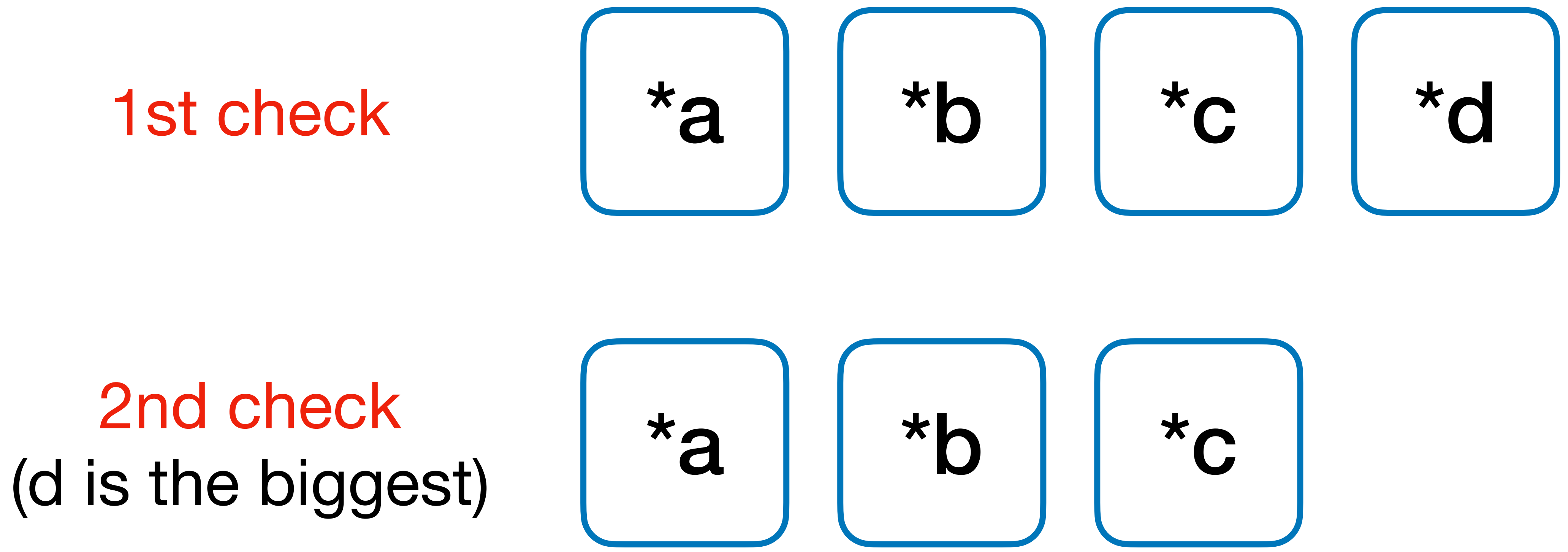
```
int swap_all(int *a, int *b, int *c, int *d) {
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*c > *d) {
        swap(c, d);
    }
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*a > *b) {
        swap(a, b);
    }
}
```

1st check

*a *b *c *d

Manual Bubble Sort

```c
int swap_all(int *a, int *b, int *c, int *d) {
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*c > *d) {
        swap(c, d);
    }
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*a > *b) {
        swap(a, b);
    }
}
```



1st check

*a *b *c *d

2nd check
(d is the biggest)

*a *b *c

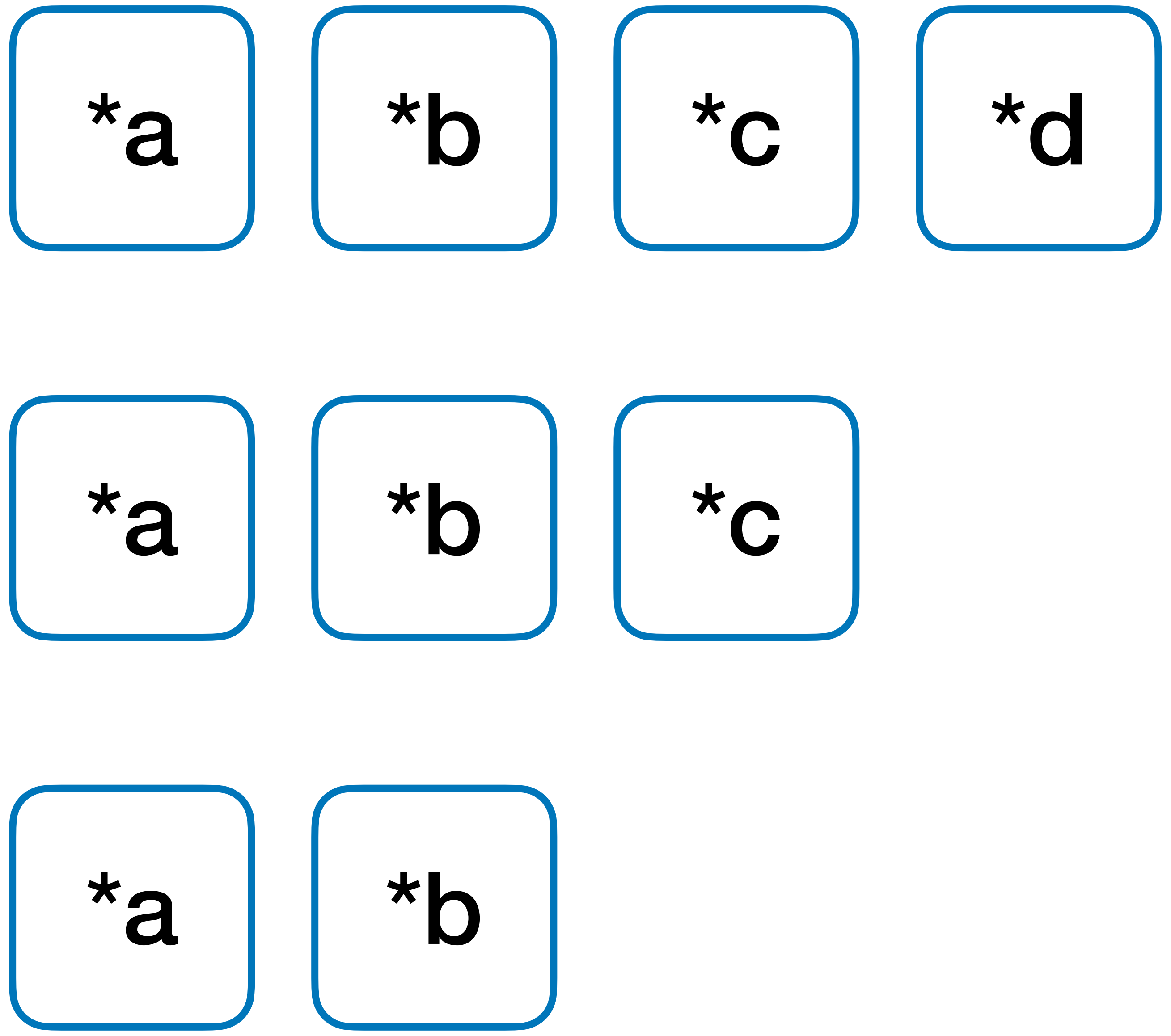
Manual Bubble Sort

```c
int swap_all(int *a, int *b, int *c, int *d) {
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*c > *d) {
        swap(c, d);
    }
    if (*a > *b) {
        swap(a, b);
    }
    if (*b > *c) {
        swap(b, c);
    }
    if (*a > *b) {
        swap(a, b);
    }
}
```

1st check

2nd check
(d is the biggest)

3rd check
(c, d is the biggest)



Bubble Sort with Pointers

C-course-materials/06-Pointers-arrays/w9_quiz.c
C-course-materials/04-Arrays/bubble_sort.c

with pointer

```c
void bubbleSort(int *arr, int size) {
    for(int i = 1; i < size; i++) {
        for(int j = 0; j < size - i; j++) {
            if(*(arr + j) > *(arr + j + 1)) {
                swap((arr + j), (arr + j + 1));
            }
        }
    }
}
```

without pointer

```c
for (int i = 1; i < 5; i++){
    for (int j = 0; j < 5 - i; j++){
        if (unsorted[j] > unsorted[j+1]){
            // swap
            temp = unsorted[j];
            unsorted[j] = unsorted[j+1];
            unsorted[j+1] = temp;
        }
    }
}
```

Bubble Sort with Pointers

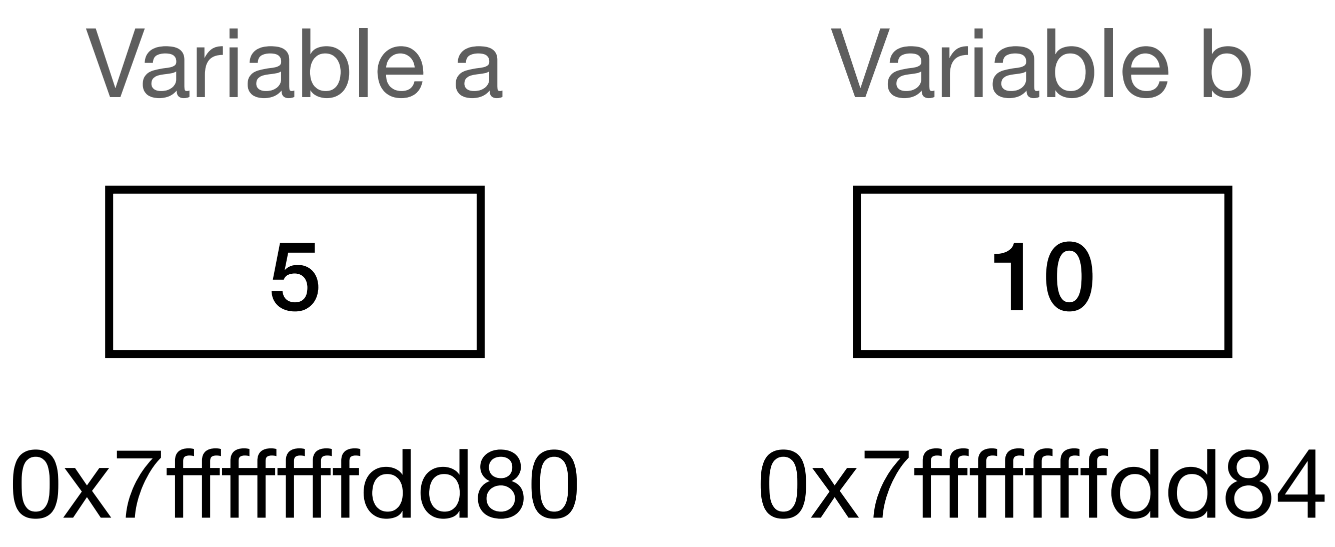
C-course-materials/06-Pointers-arrays/w9_quiz.c

```c
int main(void) {
    int arr[4] = {99, 35, 34, 97};
    int size = 4;

    printf("排序前: ");
    printArray(arr, size);

    bubbleSort(arr, size);

    printf("排序後: ");
    printArray(arr, size);

    return 0;
}
```

[Recap] Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



Variable a

5

0x7ffffffdd80

Variable b

10

0x7ffffffdd84