

# 指针与数组

胡船长

初航我带你，远航靠自己

# 本期内容

- 1. 第一节：数组
- 2. 第二节：数组-课后实战题
- 3. 第三节：指针
- 4. 第四节：指针-课后实战项目

# 一、数组

- 1. 必须要知道的概念：『地址』
- 2. 数组的基本定义与使用
- 3. 数组的存储方式

## 二、数组-课后实战题

- 1.HZOJ-144: 字符串中 A 的数量
- 2.HZOJ-145: 最长的名字
- 3.HZOJ-146: 字符串
- 4.HZOJ-147: 大数的奇偶性判断
- 5.HZOJ-148: 字符反转

- 6.HZOJ-149: 最后一个单词
- 7.HZOJ-150: 矩阵旋转
- 8.HZOJ-828: 2010年数据结构42题

## 三、指针

- 1. 『指针变量』也是『变量』
- 2. 地址的操作与取值规则
- 3. 重要：指针的几种等价形式
- 4. 数组指针与函数指针
- 5. 常用：内存管理方法
- 6. 指针学习技巧总结

# 四、指针-课后实战项目

1. qsort 函数的使用方法
2. 回调函数的基本概念
3. 快速排序算法讲解
4. 小项目：从0实现 qsort 函数

# 一、数组

1. 必须要知道的概念：『地址』
2. 数组的基本定义与使用
3. 数组的存储方式

# 聊聊：计算机中的数据存储

计算机**存储**数据的基本单位：字节

计算机**表示**数据的基本单位：位

表示数据方式：二进制

整型数据占**4**字节

0x0fd9a0

0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---

0x0fd9a1

0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---

0x0fd9a2

0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---

0x0fd9a3

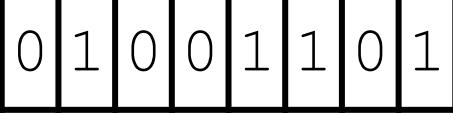
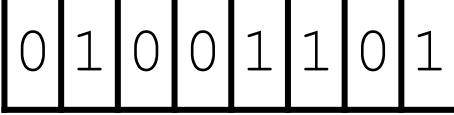
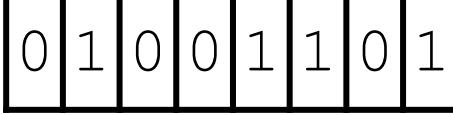
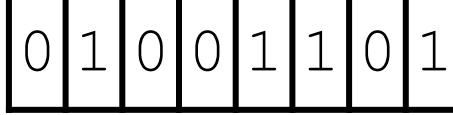
0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---

# 必须要知道的概念：『地址』

```
0x0fd9a0    0x0fd9a1    0x0fd9a2    0x0fd9a3  
int a = [01001101] [01001101] [01001101] [01001101]
```

# 必须要知道的概念：『地址』

**&a = 0x0fd9a0**      0x0fd9a1      0x0fd9a2      0x0fd9a3

**int a =**    

# 聊聊：十六进制数字

# 聊聊：十六进制数字

1. 符号组成: 0~9, A~F
2. 十六进制表示的好处

```
int a = 0x0fd9a0 0x0fd9a1 0x0fd9a2 0x0fd9a3  
      01001101 01001101 01001101 01001101
```

**问题：『地址』是一个几位的二进制数据？**

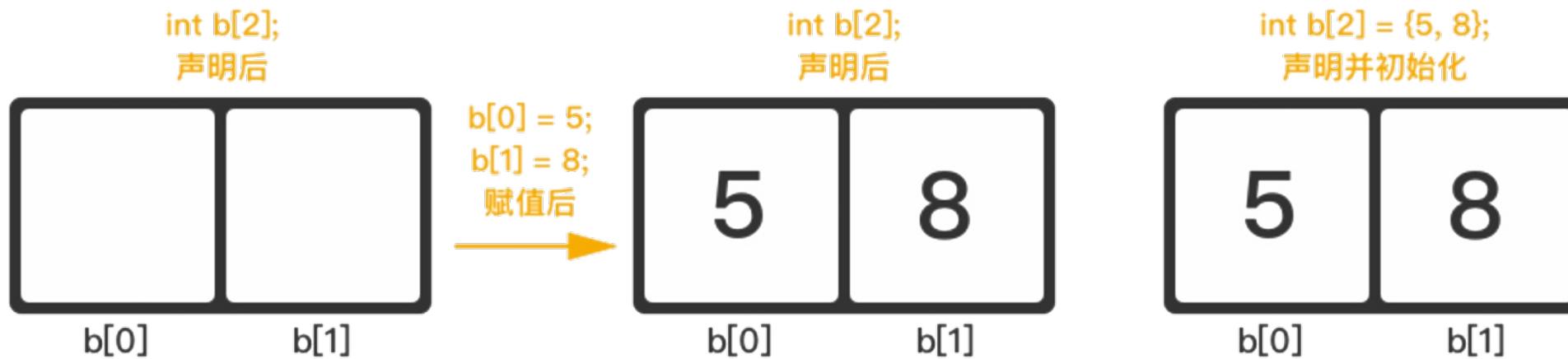
# 聊聊：32位与64位系统

所以：『地址』是一个几位的二进制数据？

# 一、数组

1. 必须要知道的概念：『地址』
2. **数组的基本定义与使用**
3. 数组的存储方式

# 数组声明与初始化



# 素数筛

- 1、标记一个范围内的数字是否是合数，没有被标记的则为素数
- 2、算法的空间复杂度为  $O(N)$ ，时间复杂度为  $O(N * \log\log N)$
- 3、总体思想是用素数去标记掉不是素数的数字，例如我知道了；是素数，那么 $2*i$ 、 $3*i$ 、 $4*i$ ……就都不是素数

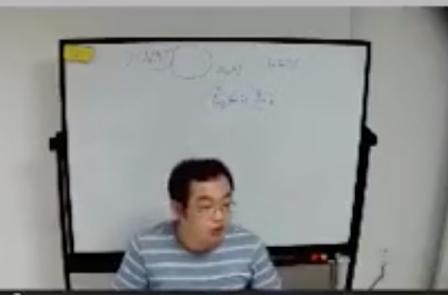
# 素数筛

- 1、用 prime[i] 来标记 i 是否是合数
- 2、标记为1的数字为合数，否则为素数
- 3、第一次知道2是素数，则将2的倍数标记为1
- 4、向后找到第一个没有被标记的数字 i
- 5、将 i 的倍数全部标记为合数
- 6、重复4--6步，直到标记完范围内的所有数字

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

1. vim

```
vim *1 bash *2 bash *3  
39 }  
40  
41 Node *insert_maintain(Node *root) {  
42     if (!hasRedChild(root)) return root;  
43     if (root->lchild->color == RED && root->rchild->color == RED)  
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;  
45         root->color = RED;  
46         root->lchild->color = root->rchild->color = BLACK;  
47         return root;  
48     }  
49     if (root->lchild->color == RED) {  
50         if (!hasRedChild(root->lchild)) return root;  
51  
52     } else {  
53         if (!hasRedChild(root->rchild)) return root;  
54     }  
55  
56 }  
57  
58 [ ]
```



## 素数筛：代码演示

```
59  
60  
61 Node *__insert(Node *root, int key) {  
62     if (root == NIL) return getNode(key);
```

# 二分查找

待查找

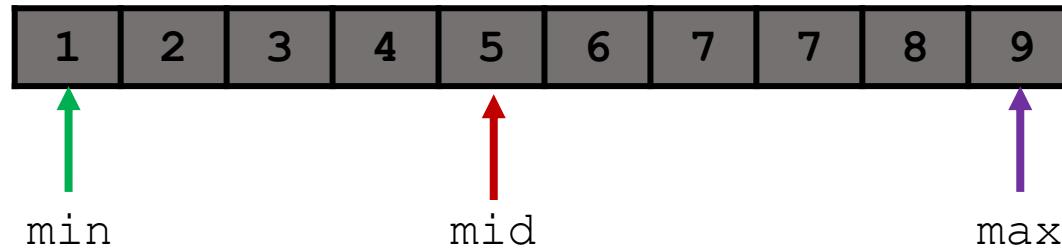
x

1	2	3	4	5	6	7	7	8	9
---	---	---	---	---	---	---	---	---	---

# 二分查找

待查找

x



**min**是头指针; **max**是尾指针; **mid = (min + max) / 2**

调整:

如果  $\text{arr}[\text{mid}] < \text{x}$ ,  $\text{min} = \text{mid} + 1$

如果  $\text{arr}[\text{mid}] > \text{x}$ ,  $\text{max} = \text{mid} - 1$

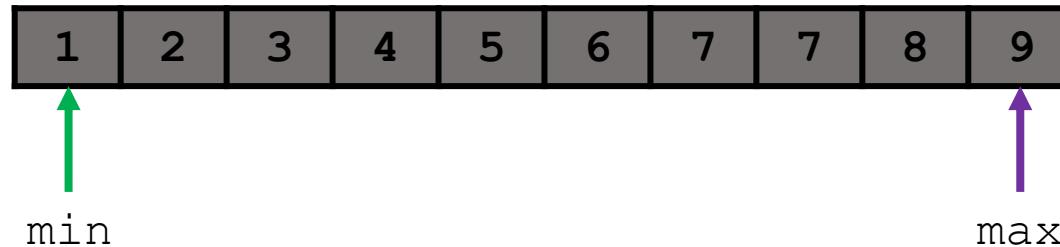
如果  $\text{arr}[\text{mid}] == \text{x}$ , 找到结果

终止条件:  $\text{min} \geq \text{max}$

# 二分查找

待查找

7

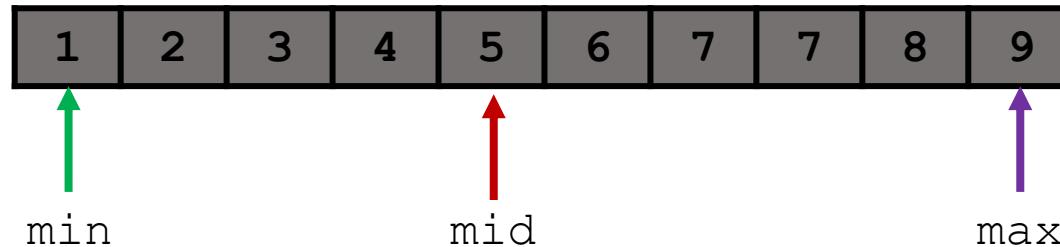


```
arr[mid] < x, min = mid + 1  
重新计算, 得到 mid = 6
```

# 二分查找

待查找

7



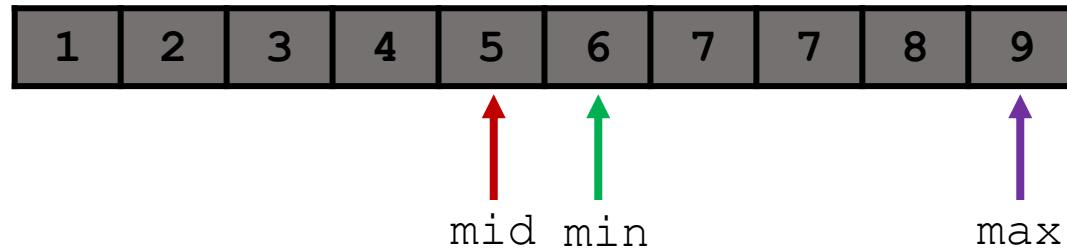
`arr[mid] < x, min = mid + 1`

重新计算，得到  $mid = 6$

# 二分查找

待查找

7

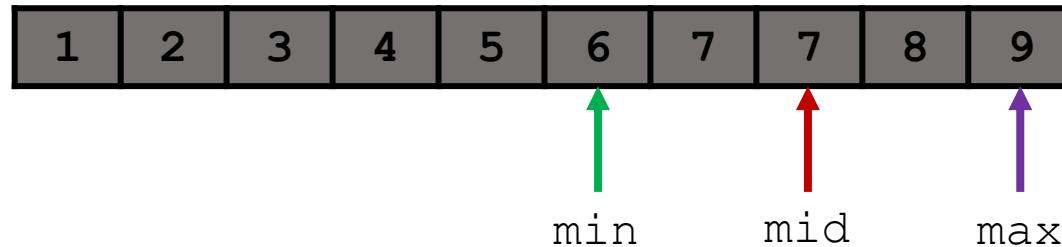


```
arr[mid] < x, min = mid + 1  
重新计算, 得到 mid = 6
```

# 二分查找

待查找

7



`arr[mid] == x`, 找到结果  
总查找次数: 2次

# 二分查找

待查找

7



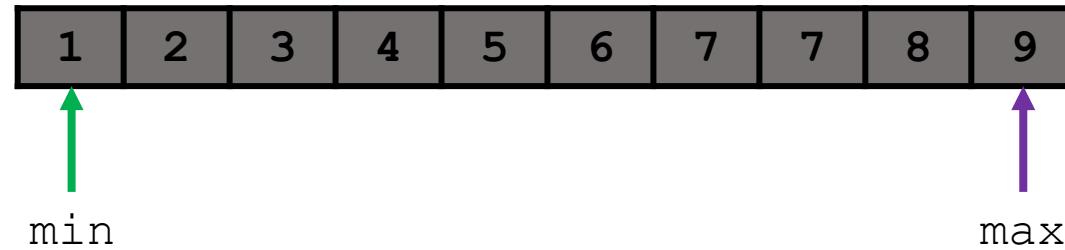
mid  
↑

`arr[mid] == x`, 找到结果  
总查找次数: 2次

# 二分查找

待查找

4

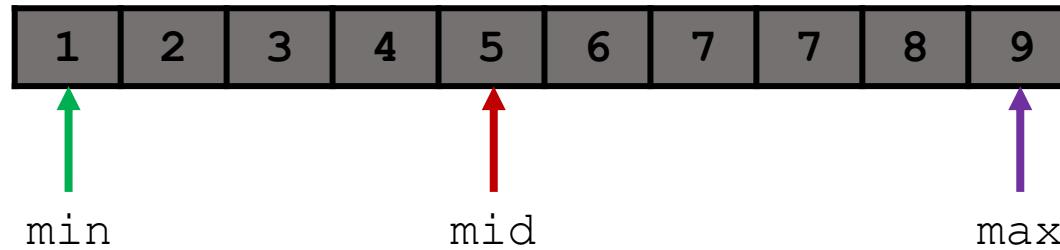


采用二分查找，总查找次数为：?次

# 二分查找

待查找

4

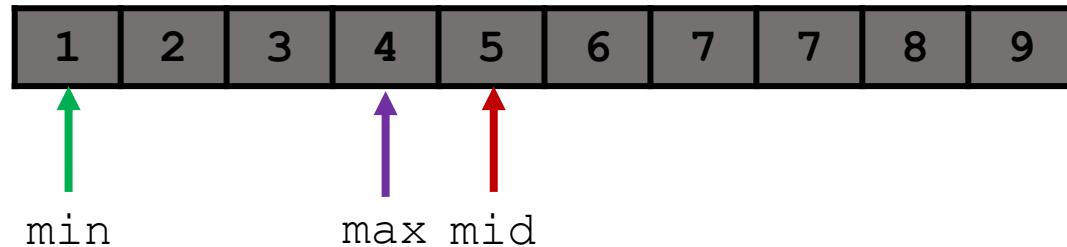


采用二分查找，总查找次数为：?次

# 二分查找

待查找

4

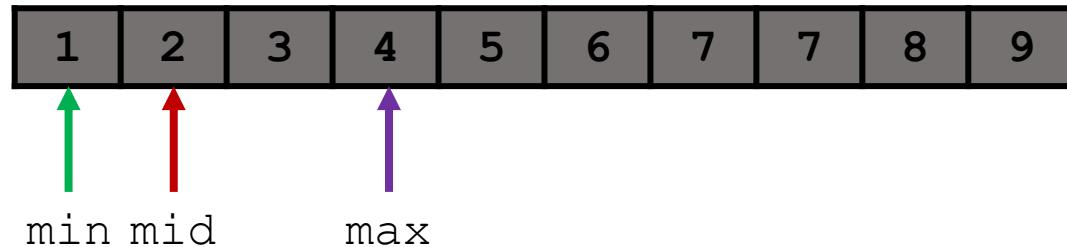


采用二分查找，总查找次数为：?次

# 二分查找

待查找

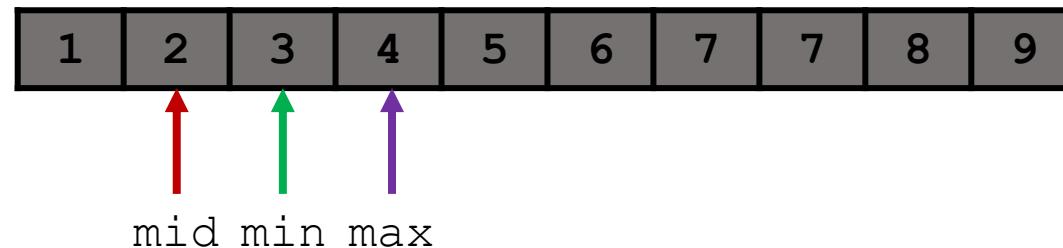
4



# 二分查找

待查找

4



# 二分查找

待查找

4

1	2	3	4	5	6	7	7	8	9
---	---	---	---	---	---	---	---	---	---

min max  
mid

# 二分查找

待查找

4

1	2	3	4	5	6	7	7	8	9
---	---	---	---	---	---	---	---	---	---



max



min

采用二分查找，总查找次数为：4次

# 二分查找

待查找

4

1	2	3	4	5	6	7	7	8	9
---	---	---	---	---	---	---	---	---	---

↑  
max  
↑  
min  
↑  
mid

采用二分查找，总查找次数为：4次

《船说：C 语言全能实战课》  
第5章-指针与数组

# 二分查找

待查找

4

1	2	3	4	5	6	7	7	8	9
---	---	---	---	---	---	---	---	---	---



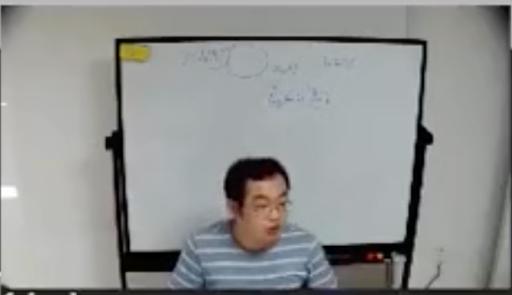
mid

采用二分查找，总查找次数为：4次

《船说：C 语言全能实战课》  
第5章-指针与数组

vim    #1    bash    #2    bash    #3

```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED) {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54     }
55
56 }
57
58 
```



## 二分算法：代码演示

```
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNode(key);
```

# 从一维数组到多维数组

```
int b[3][4];
```

	0	1	2	3
0				
1				
2				

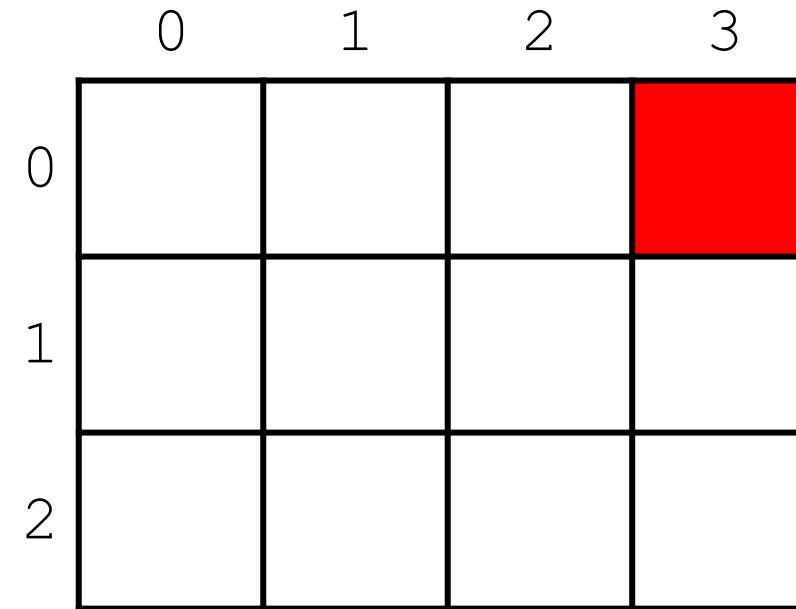
# 从一维数组到多维数组

```
int b[3][4];
```

	0	1	2	3
0				
1				
2				

# 从一维数组到多维数组

```
int b[3][4];
```



# 从一维数组到多维数组

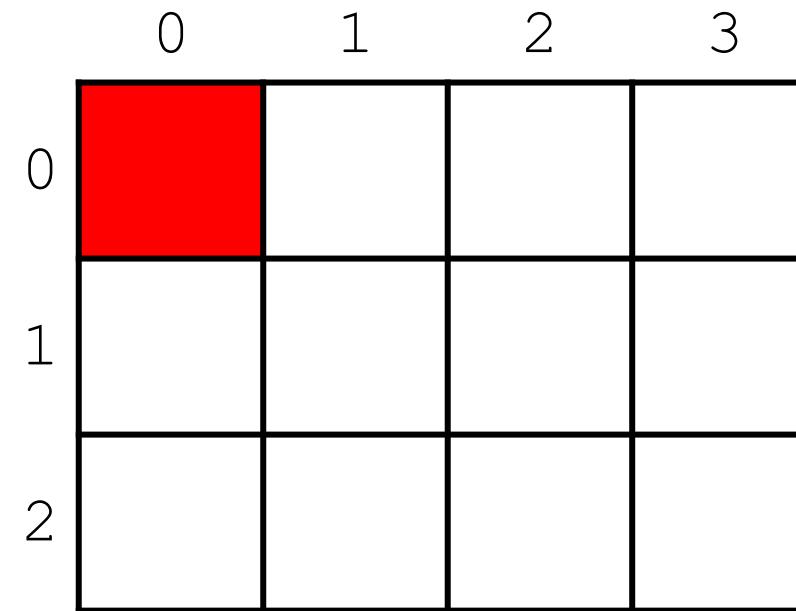
```
int b[3][4];
```

	0	1	2	3
0				
1				
2				

# 从一维数组到多维数组

```
int b[3][4];
```

```
b = &b[0][0]
```



# 聊聊：字符数组及操作

# 聊聊：字符数组及操作

<https://zh.cppreference.com/w/c/string/byte>

## 空终止字节字符串

空终止字节字符串（NTBS）是尾随零值字节（空终止字符）的非零字节序列。字节字符串中的每个字节都是一些字符集的编码。例如，字符数组 `{'\x63', '\x61', '\x74', '\0'}` 是一个以 ASCII 编码表示字符串 "cat" 的 NTBS。

### 函数

#### 字符分类

在标头 `<ctype.h>` 定义

**isalnum** 检查一个字符是否是字母或数字  
(函数)

**isalpha** 检查一个字符是否是英文字母  
(函数)

**islower** 检查一个字符是否是小写字母  
(函数)

# 字符数组

定义字符数组 : `char str[size];`

初始化字符数组 :

`char str[] = "hello world";`

`char str[size] = {'h', 'e', 'l', 'l', 'o'};`

# 字符串相关操作

头文件：string.h

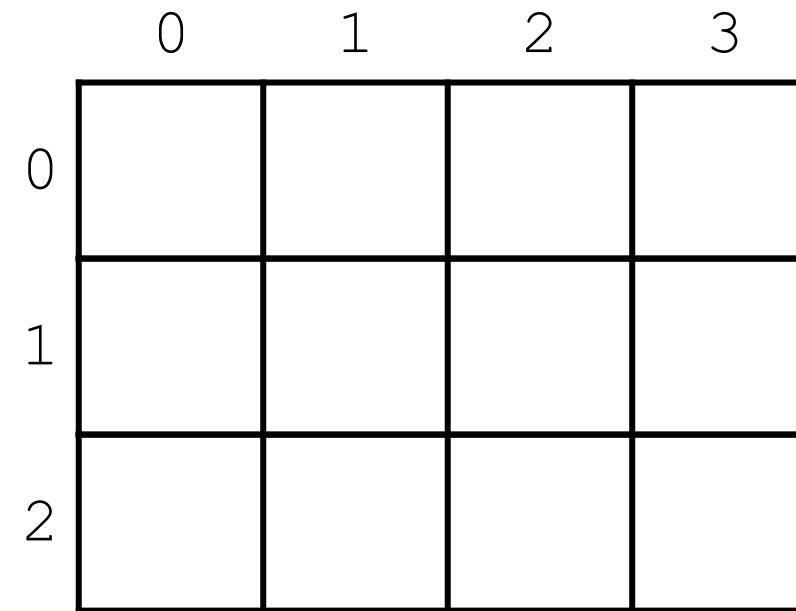
函数	说明
strlen(str)	计算字符串长度，以 \0 作为结束符
strcmp(str1, str2);	字符串比较
strcpy(dest, src);	字符串拷贝
strncmp(str1, str2, n);	安全的字符串比较
strncpy(str1, str2, n);	安全的字符串拷贝
memcpy(str1, str2, n);	内存拷贝
memcmp(str1, str2, n);	内存比较
memset(str1, c, n);	内存设置

# 一、数组

1. 必须要知道的概念：『地址』
2. 数组的基本定义与使用
3. 数组的存储方式

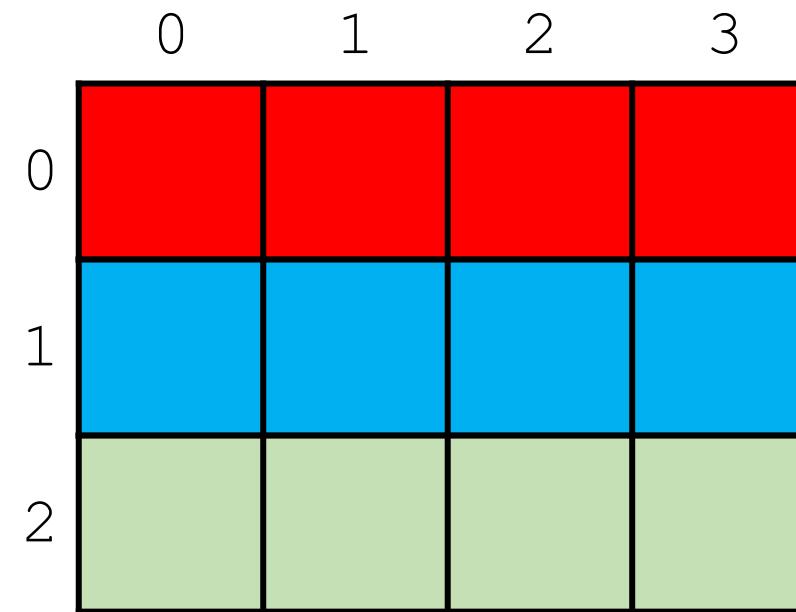
# 数组的存储方式：行序优先

```
int b[3][4];  
b = &b[0][0]
```



# 数组的存储方式：行序优先

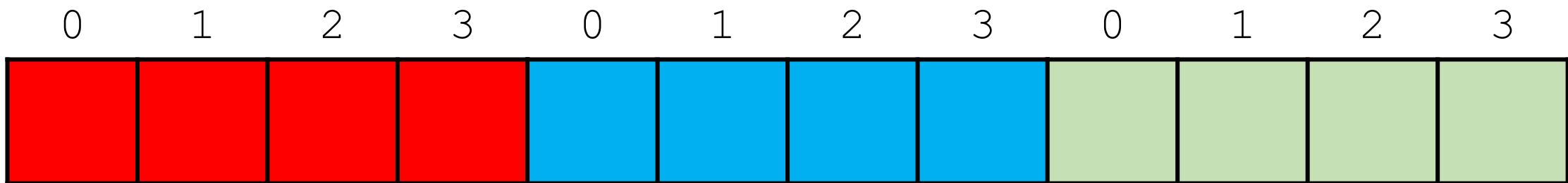
```
int b[3][4];  
b = &b[0][0]
```



# 数组的存储方式：行序优先

```
int b[3][4];
```

```
b = &b[0][0]
```

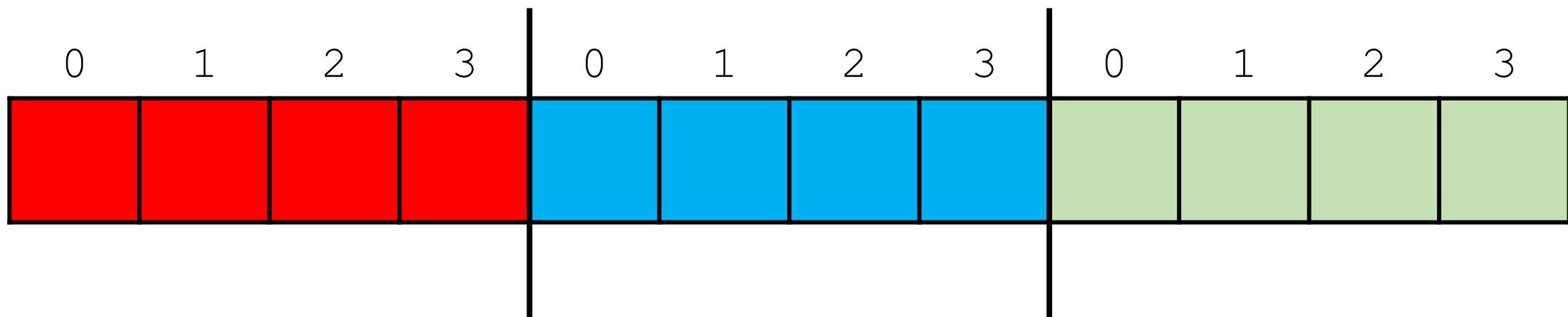


# 数组的存储方式：行序优先

```
int b[3][4];
```

```
b = &b[0][0]
```

问题： $b[1][2]$ 是第几个元素？



# 数组的存储方式：列序优先

```
int b[3][4];  
b = &b[0][0]
```

0	3	6	9
1	4	7	10
2	5	8	11

# 数组的存储方式：列序优先

```
int b[3][4];  
b = &b[0][0]
```

0	1	2
---	---	---

3	6	9
4	7	10
5	8	11

# 数组的存储方式：列序优先

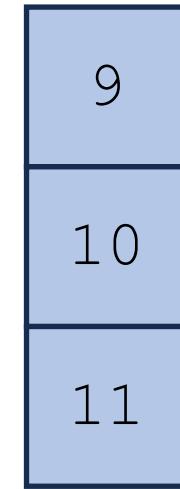
```
int b[3][4];  
b = &b[0][0]
```

0	1	2	3	4	5
---	---	---	---	---	---

6	9
7	10
8	11

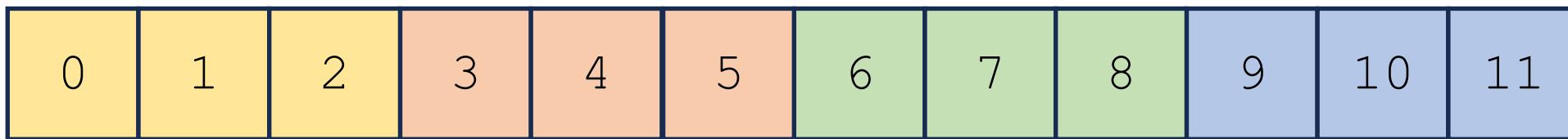
# 数组的存储方式：列序优先

```
int b[3][4];  
b = &b[0][0]
```



# 数组的存储方式：列序优先

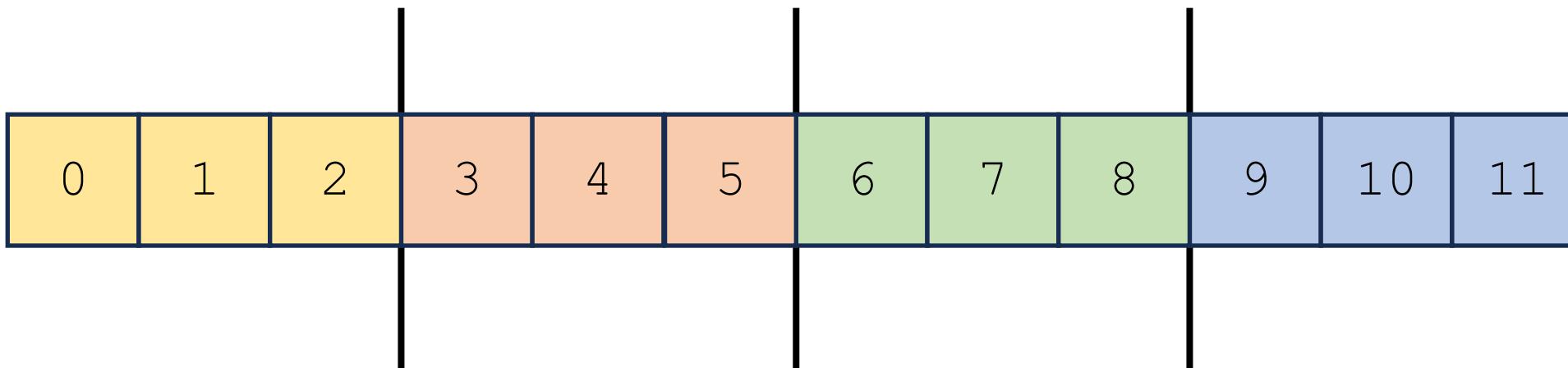
```
int b[3][4];  
b = &b[0][0]
```



# 数组的存储方式：列序优先

```
int b[3][4];  
b = &b[0][0]
```

问题：b[1][2]是第几个元素？



## 二、数组-课后实战题

- 1.HZOJ-144: 字符串中 A 的数量
- 2.HZOJ-145: 最长的名字
- 3.HZOJ-146: 字符串
- 4.HZOJ-147: 大数的奇偶性判断
- 5.HZOJ-148: 字符反转

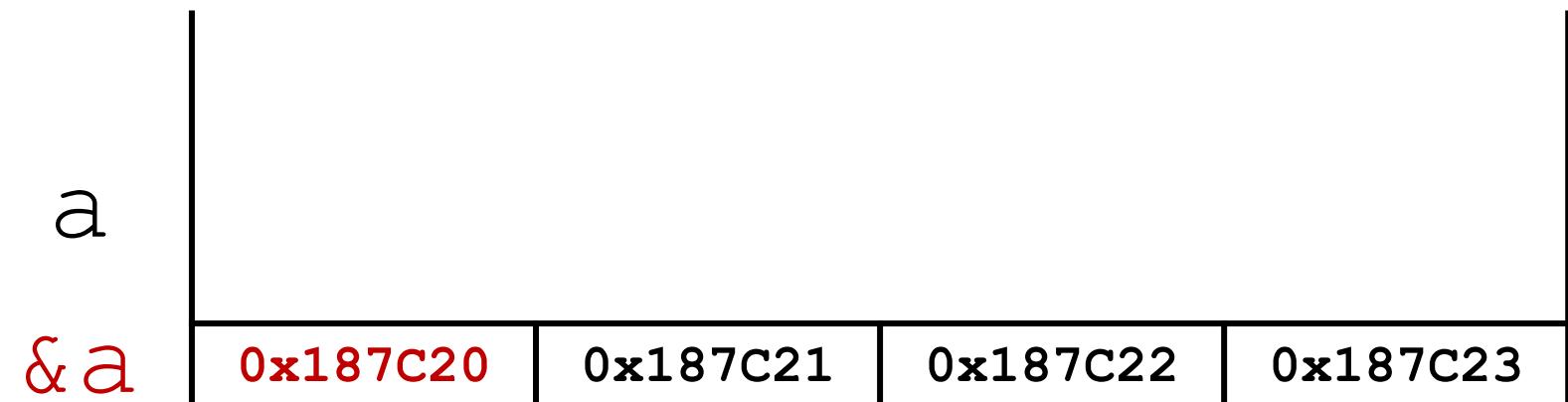
- 6.HZOJ-149: 最后一个单词
- 7.HZOJ-150: 矩阵旋转
- 8.HZOJ-828: 2010年数据结构42题

# 三、指针

1. 『指针变量』也是『变量』
2. 地址的操作与取值规则
3. 重要：指针的几种等价形式
4. 数组指针与函数指针
5. 常用：内存管理方法
6. 指针学习技巧总结

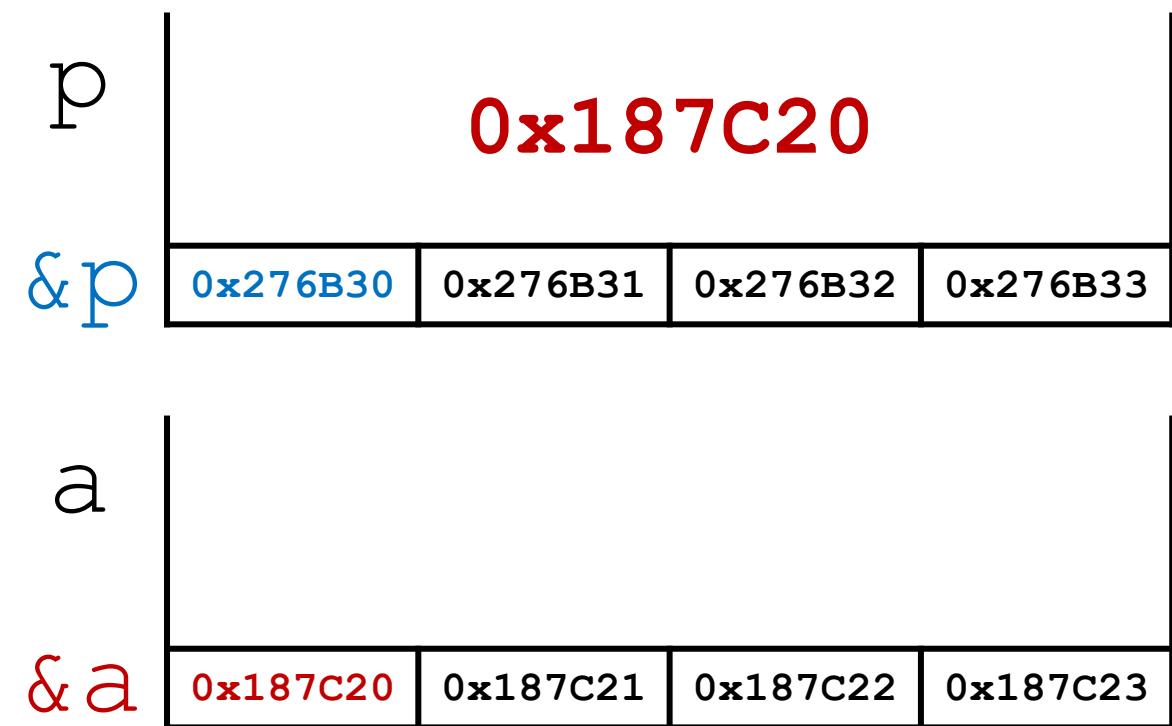
# 『指针变量』也是『变量』

```
int a;
```



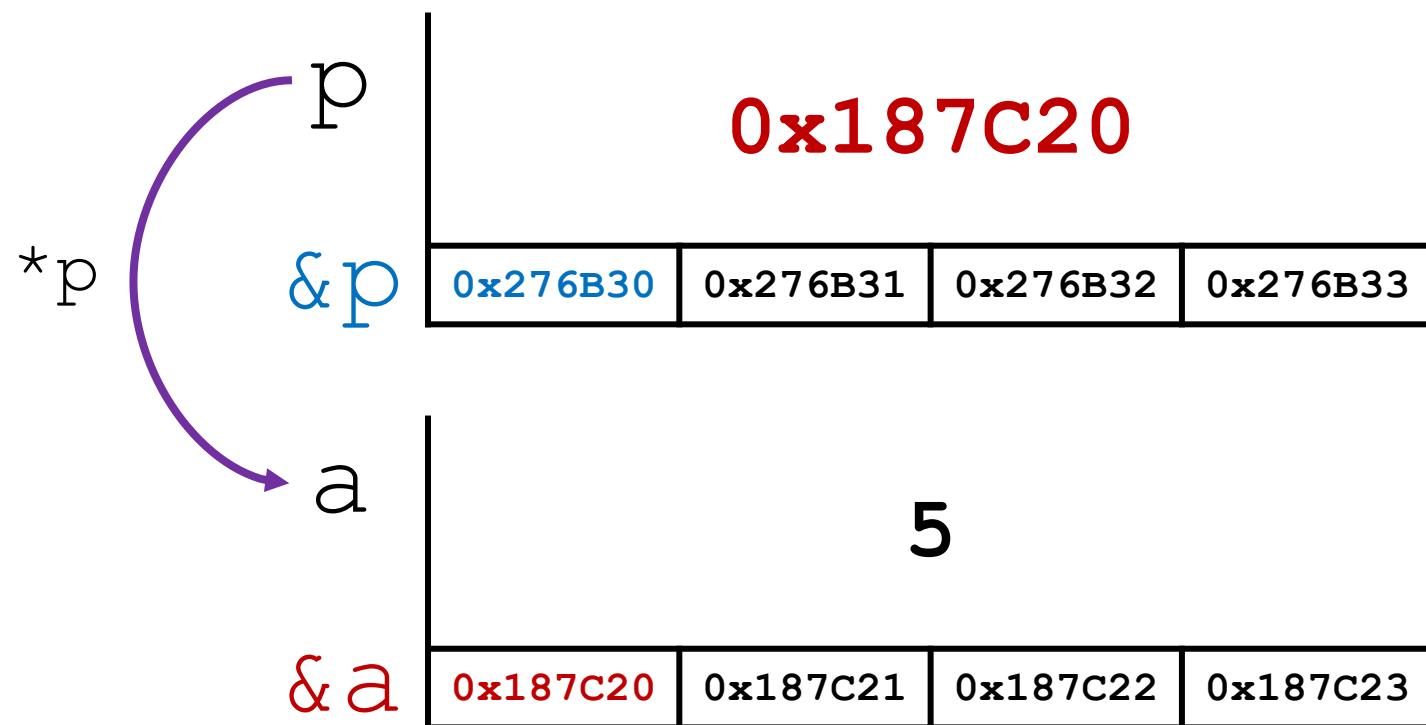
# 『指针变量』也是『变量』

```
int a;  
int *p = &a;
```



# 『指针变量』也是『变量』

```
int a;  
int *p = &a;  
*p = 5;
```



# 核心法门

指针变量，也是变量

所以：『指针变量』 占几个字节？

# 函数传递指针变量的场景和用途

# 输出函数说明

## scanf 函数

头文件 : stdio.h

原型 : int **scanf**(const char \*format, ...);

format : 格式控制字符串

... : 可变参数列表

返回值 : 成功读入的参数个数

例子 : `scanf("%d", &n);`

# 输出函数说明

## scanf 函数

头文件 : stdio.h

原型 : int **scanf**(const char \*format, ...);

format : 格式控制字符串

... : 可变参数列表

返回值 : 成功读入的参数个数

例子 : `scanf("%d", &n);`

交换指针变量：HZ0J-881

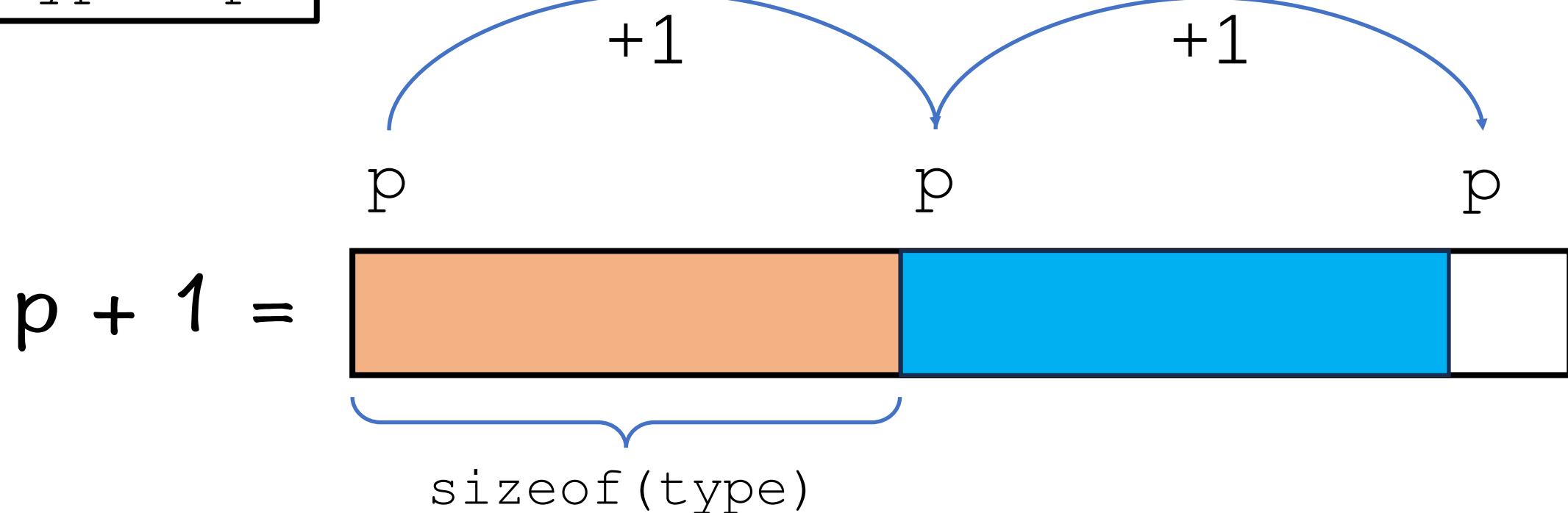
# 三、指针

1. 『指针变量』也是『变量』
2. 地址的操作与取值规则
3. 重要：指针的几种等价形式
4. 数组指针与函数指针
5. 常用：内存管理方法
6. 指针学习技巧总结

# 深入理解『 $p + 1$ 』

# 深入理解 $\lceil p + 1 \rceil$

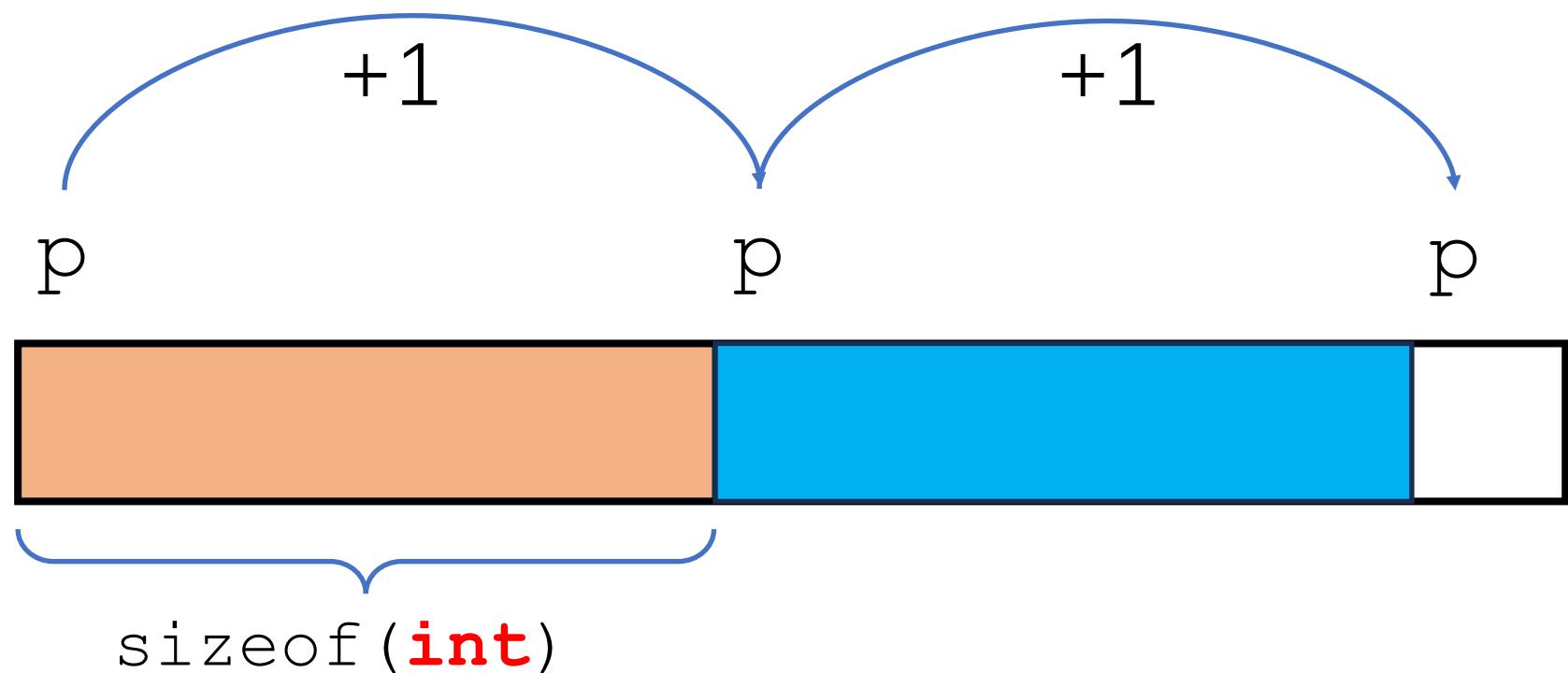
```
type *p;
```



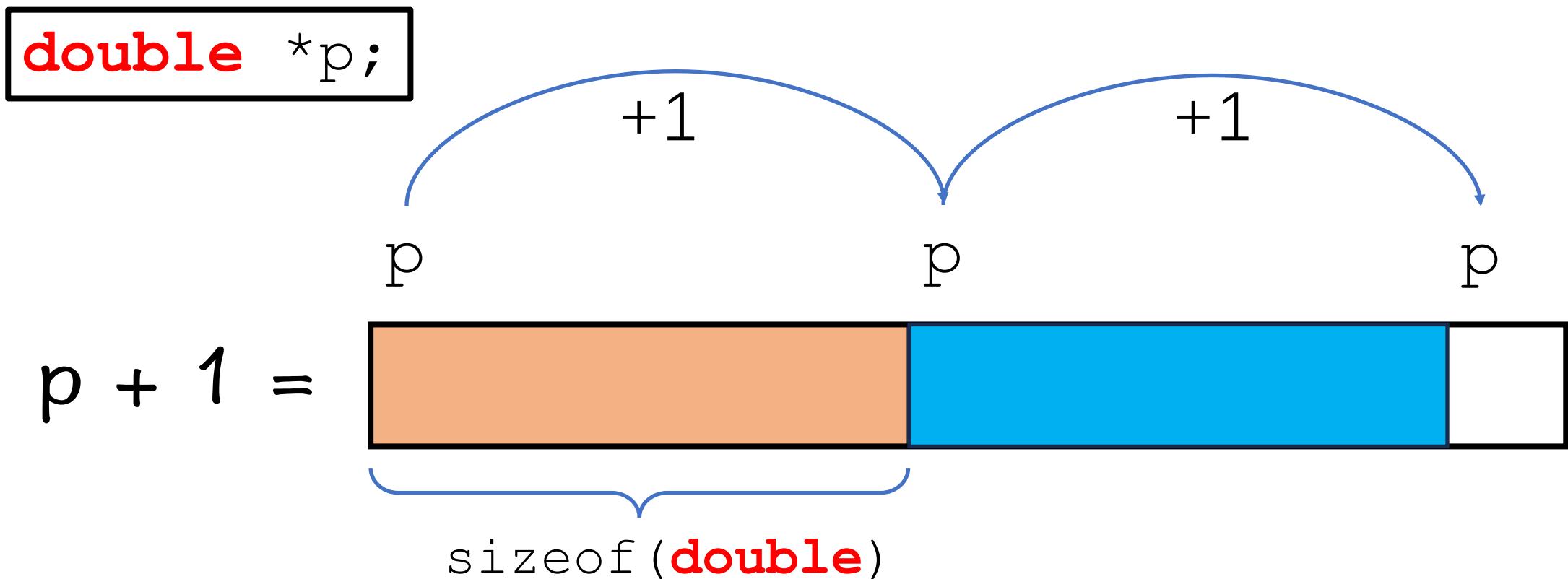
# 深入理解 $\lceil p + 1 \rceil$

**int \*p;**

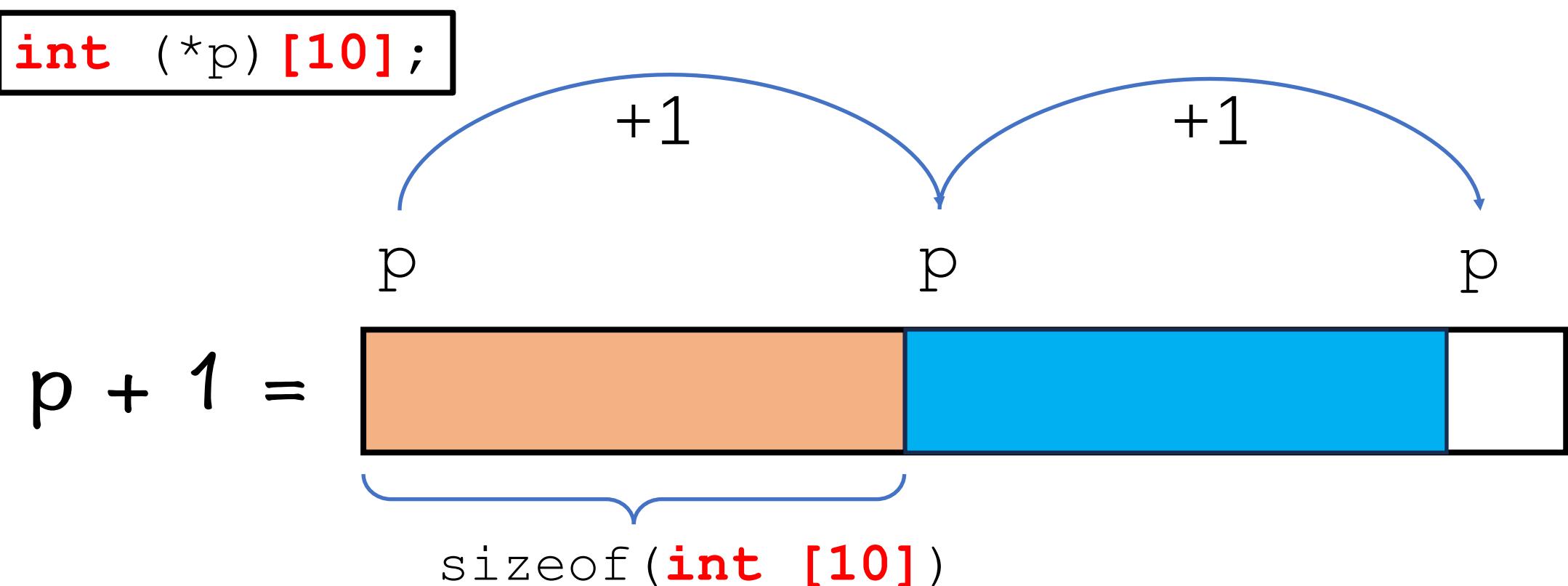
$p + 1 =$



# 深入理解 $\lceil p + 1 \rceil$

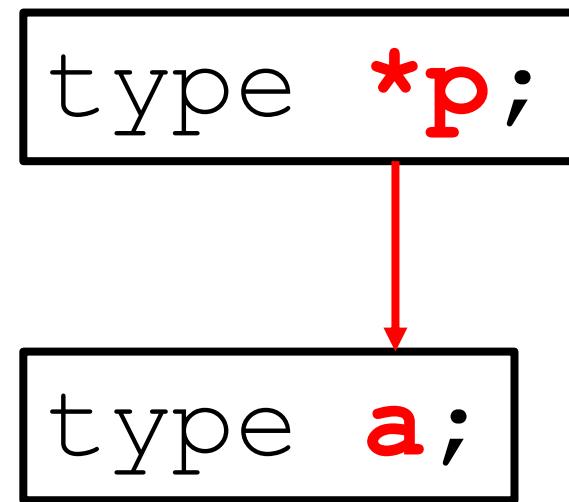


# 深入理解 $\lceil p + 1 \rceil$



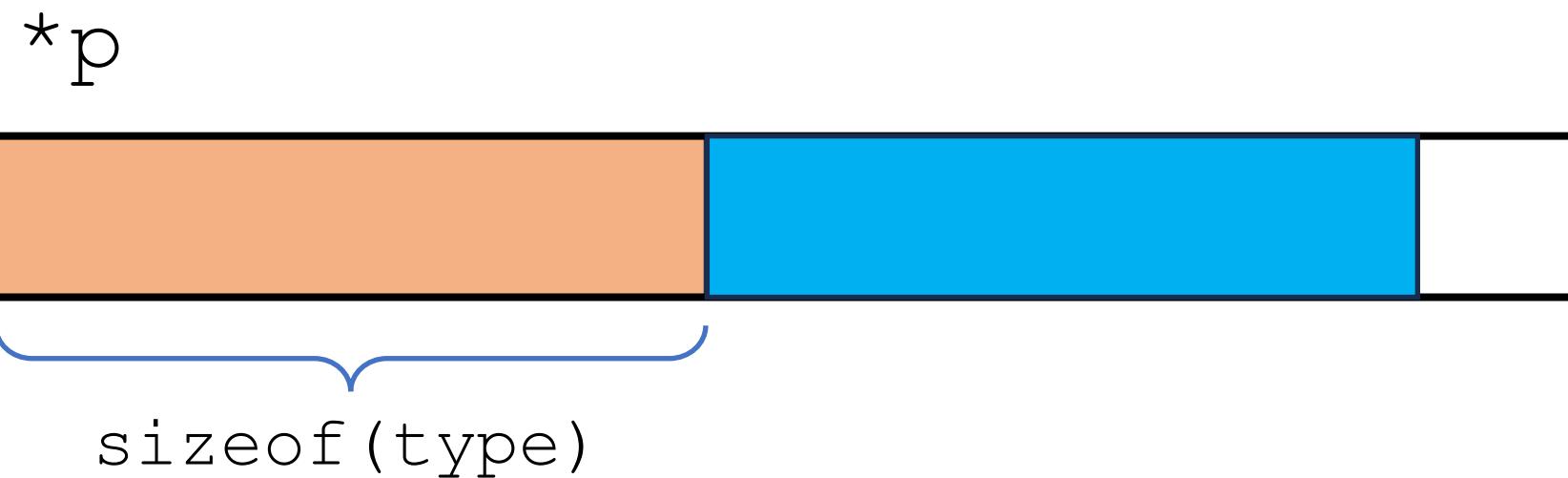
# 深入理解『\*p』

# 深入理解『\*p』



# 深入理解 `『*p』`

```
type *p;
```



# 随堂练习题1

Ip 地址一般是由 4 个不超过 255 的数字组成，例如：192.168.1.1

实现一个程序，读入 ip 地址，转换成一个32位整数

**要求：**不要使用数学运算符

# 三、指针

1. 『指针变量』也是『变量』
2. 地址的操作与取值规则
3. **重要：指针的几种等价形式**
4. 数组指针与函数指针
5. 常用：内存管理方法
6. 指针学习技巧总结

# 看懂一段代码

```
8 #include<stdio.h>
9
10 int main() {
11     int arr[] = {0, 1, 2, 3, 4, 5, 6};
12     int *p = arr;
13     for (int i = 0; i < 3; i++) {
14         printf("%d\n", (i + 5)[&p[1] - 2]);
15     }
16     return 0;
17 }
```

# 看懂一段代码

```
8 #include<stdio.h>
9
10 int main() {
11     int arr[] = {0, 1, 2, 3, 4, 5, 6};
12     int *p = arr;
13     for (int i = 0; i < 3; i++) {
14         printf("%d\n", (i + 5)[&p[1] - 2]);
15     }
16     return 0;
17 }
```

# 看懂一段代码

```
8 #include<stdio.h>
9
10 int main() {
11     int arr[] = {0, 1, 2, 3, 4, 5, 6};
12     int *p = arr;
13     for (int i = 0; i < 3; i++) {
14         printf("%d\n", (i + 5)[&p[1] - 2]);
15     }
16     return 0;
17 }
```

# 等价形式转换

1.  $p \Leftrightarrow \&a$

2.  $p + 1 \Leftrightarrow 1 + p$

3.  $p + 1 \Leftrightarrow \&p[1]$

4.  $*p \Leftrightarrow p[0] \Leftrightarrow a$

5.  $p[n] \Leftrightarrow *(p + n)$

# 等价形式转换

1.  $p \Leftrightarrow \&a$

2.  $p + 1 \Leftrightarrow 1 + p$

3.  $p + 1 \Leftrightarrow \&p[1]$

4.  $*p \Leftrightarrow p[0] \Leftrightarrow a$

5.  $p[n] \Leftrightarrow *(p + n)$

分析： $(i + 5)[\&p[1] - 2]$

# 三、指针

1. 『指针变量』也是『变量』
2. 地址的操作与取值规则
3. 重要：指针的几种等价形式
4. 数组指针与函数指针
5. 常用：内存管理方法
6. 指针学习技巧总结

# 什么是数组指针

# 语法太奇怪了，如何记忆？

1. 变量或函数名前面加个 `*`，就变成了指向变量或函数的『指针』
2. 变量名后面加个方括号，就变成了相关类型的『数组』

# 数组指针

```
int arr1[10];  
int arr2[10][10];  
int *p1 = arr1;  
int (*p2)[10] = arr2;
```

# 什么是函数指针

# 函数指针

```
int (*add) (int, int);
```

# 函数指针数组

```
int (*add[10])(int, int);
```

# 语法太奇怪了，如何记忆？

1. 变量或函数名前面加个 `*`，就变成了指向变量或函数的『指针』
2. 变量名后面加个方括号，就变成了相关类型的『数组』

## 三、指针

1. 『指针变量』也是『变量』
2. 地址的操作与取值规则
3. 重要：指针的几种等价形式
4. 数组指针与函数指针
5. **常用：内存管理方法**
6. 指针学习技巧总结

# 常用：内存管理方法

1. malloc 函数

2. calloc 函数

3. free 函数

4. memset 函数

5. memcpy 函数

6. memmove 函数

# 常用：内存管理方法

1. malloc 函数	动态申请一段内存空间
2. calloc 函数	动态申请一段内存空间，初始化0值
3. free 函数	释放动态申请的空间
4. memset 函数	设置每个字节为一个固定值
5. memcpy 函数	内存数据拷贝
6. memmove 函数	同 memcpy，能处理空间有重叠的情况

## 三、指针

1. 『指针变量』也是『变量』
2. 地址的操作与取值规则
3. 重要：指针的几种等价形式
4. 数组指针与函数指针
5. 常用：内存管理方法
6. 指针学习技巧总结

# 分辨『指针』的真实身份

1. const int \*p;
2. int const \*p;
3. int \*const p;
4. float (\*p[5])[10];

从后往前读，翻译如下

p 就读『p 是一个』

\* 就读『指针，指向』或『指针数组，指向』

const 就读『常量』

类型正常读

# 分辨『指针』的真实身份

- |                       |                          |
|-----------------------|--------------------------|
| 1. const int *p;      | 『p 是一个』『指针, 指向』『整型』『常量』  |
| 2. int const *p;      | 『p 是一个』『指针, 指向』『常量』『整型』  |
| 3. int *const p;      | 『p 是一个』『常量』『指针, 指向』『整型』  |
| 4. float (*p[5])[10]; | 『p 是一个』『指针数组, 指向』『浮点型数组』 |

从后往前读, 翻译如下

p 就读 『p 是一个』

\* 就读 『指针, 指向』或『指针数组, 指向』

const 就读 『常量』

类型正常读

# 语法太奇怪了，如何记忆？

1. 变量或函数名前面加个 \*，就变成了指向变量或函数的『指针』
2. 变量名后面加个方括号，就变成了相关类型的『数组』

```
int (*add)(int, int);           int (*add[10])(int, int);
```

# 等价形式转换

1.  $p \Leftrightarrow \&a$

2.  $p + 1 \Leftrightarrow 1 + p$

3.  $p + 1 \Leftrightarrow \&p[1]$

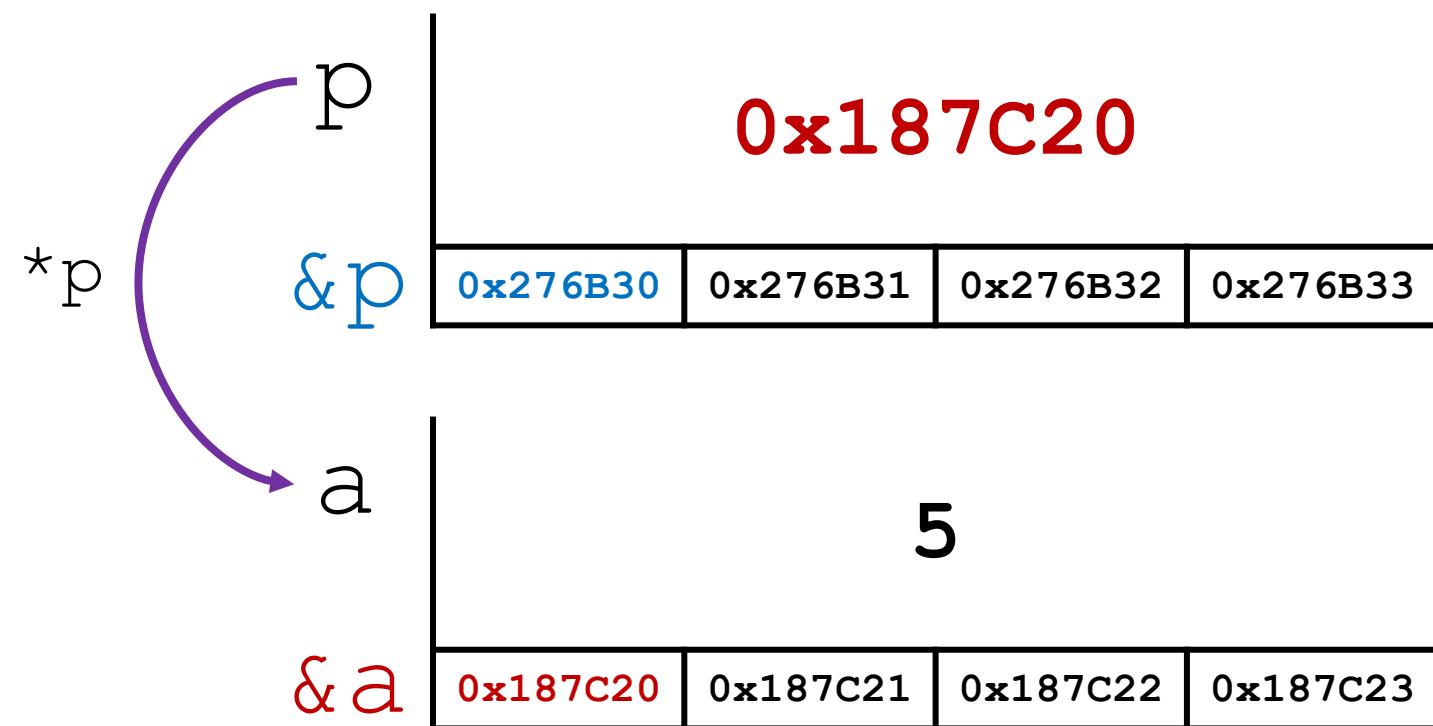
4.  $*p \Leftrightarrow p[0] \Leftrightarrow a$

5.  $p[n] \Leftrightarrow *(p + n)$

分析： $(i + 5)[\&p[1] - 2]$

# 『指针变量』也是『变量』

```
int a;  
int *p = &a;  
*p = 5;
```



# typedef : 将变量名变成类型别名

内建类型的重命名：

```
typedef long long lint;  
typedef char * pchar;
```

结构体类型的重命名：

```
typedef struct __node{  
    int x, y;  
} Node, *PNode;
```

函数指针命名：

```
typedef int (*func) (int);
```

# 四、指针-课后实战项目

1. qsort 函数的使用方法
2. 回调函数的基本概念
3. 快速排序算法讲解
4. 小项目：从0实现 qsort 函数

# 排序函数

## qsort 函数

头文件 : stdlib.h

原型 : void **qsort**(void \*arr, size\_t count, size\_t size,  
                 int (\*comp)(const void \*, const void \*) );

**arr** : 待排序数组的起始位置

**count** : 排序元素数量

**size** : 每个元素的所占字节数

**comp** : 比较规则函数

# 四、指针-课后实战项目

1. qsort 函数的使用方法
2. **回调函数的基本概念**
3. 快速排序算法讲解
4. 小项目：从0实现 qsort 函数

# 什么是回调函数？

『回调函数』就是一个被作为参数传递的函数

# 随堂练习题2

实现对任一定义域在  $[0, 1000]$  内的单调函数求解的方法，测试的时候，  
你的方法可以尝试对以下单调函数进行求解：

1.  $f(x) = x * x$
2.  $f(x) = 3x^2 + 2x - 5$
3.  $f(x) = 1.2^x$

**要求**：实现一个方法，能同时对以上三个函数进行求解

## 随堂练习题3：

个人所得税，是根据收入进行阶梯设置的，我国个人所得税现行标准如下：

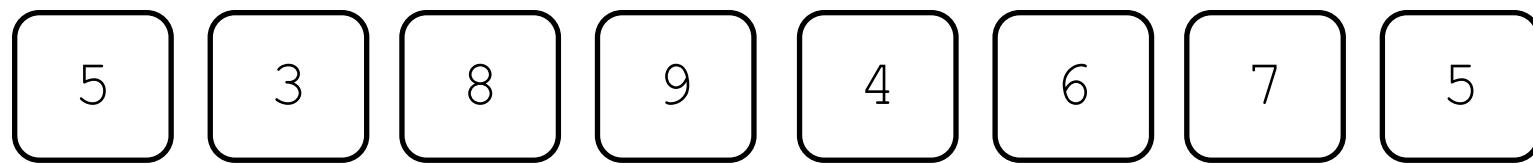
个人所得税税率表三（非居民公司薪金所得等按月换算）			
级数	全年应纳税所得额	税率（%）	速算扣除数
1	不超过3000元的部分	3	0
2	超过3000元至12000元的部分	10	210
3	超过12000元至25000元的部分	20	1410
4	超过25000元至35000元的部分	25	2660
5	超过35000元至55000元的部分	30	4410
6	超过55000元至80000元的部分	35	7160
7	超过80000元的部分	45	15160

**问题：**给出税后收入，求税前收入

# 四、指针-课后实战项目

1. qsort 函数的使用方法
2. 回调函数的基本概念
3. 快速排序算法讲解
4. 小项目：从0实现 qsort 函数

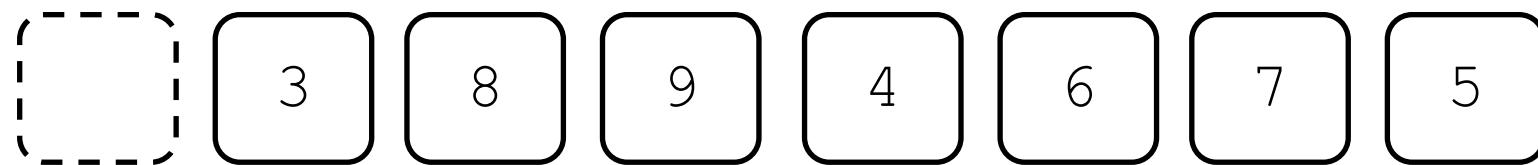
# 快速排序



# 快速排序

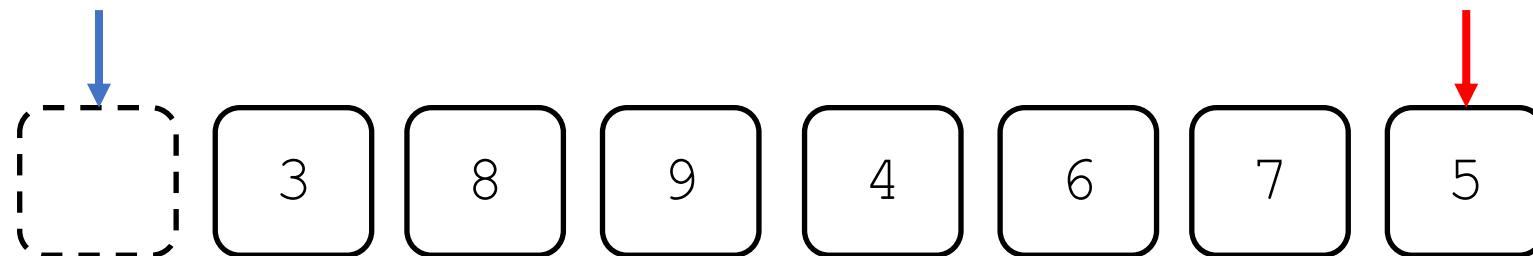
选择基准值

5



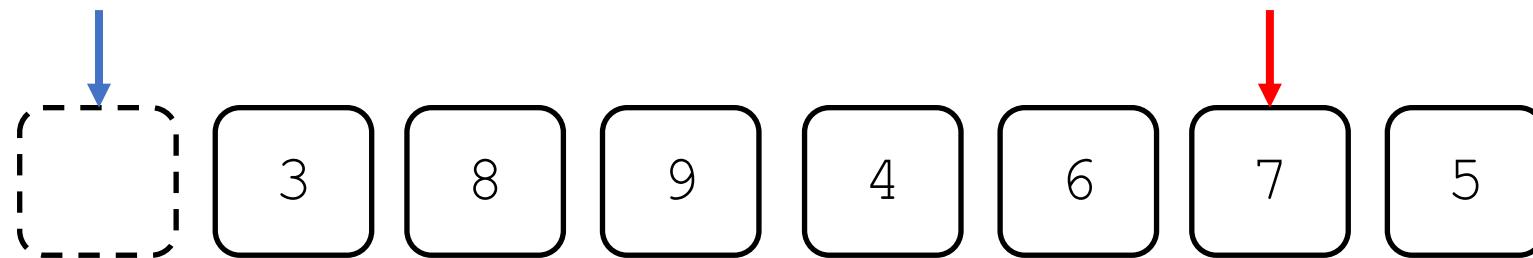
# 快速排序

选择基准值 5



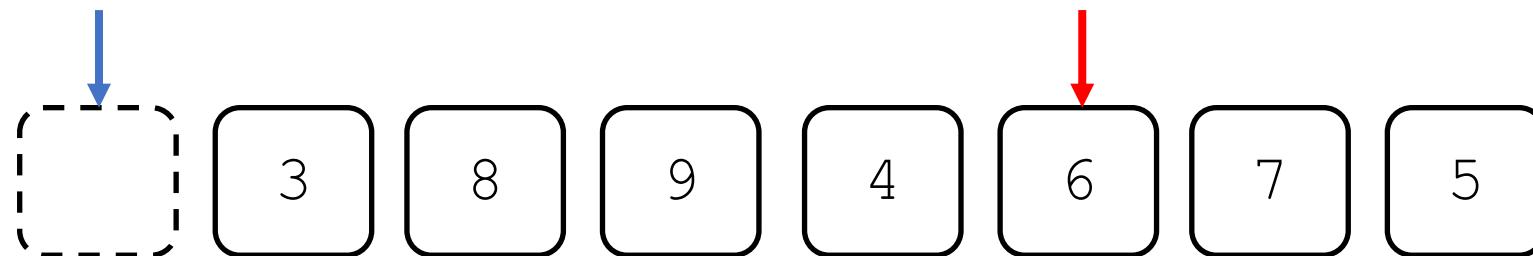
# 快速排序

选择基准值 5



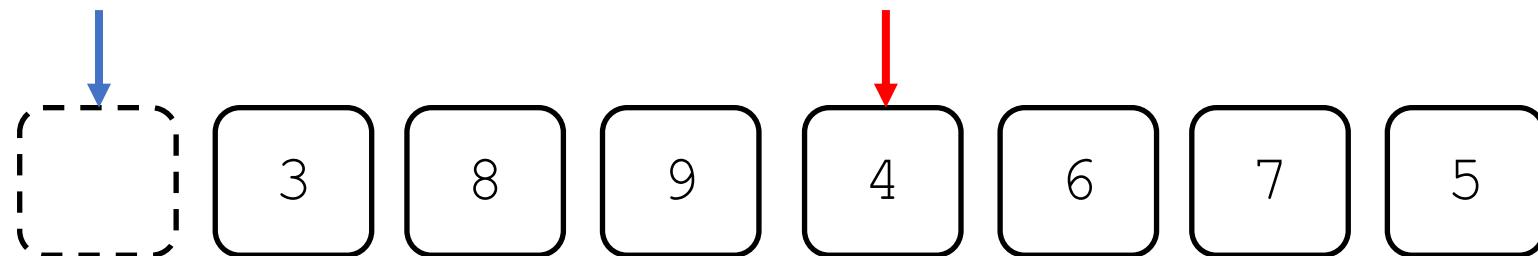
# 快速排序

选择基准值 5

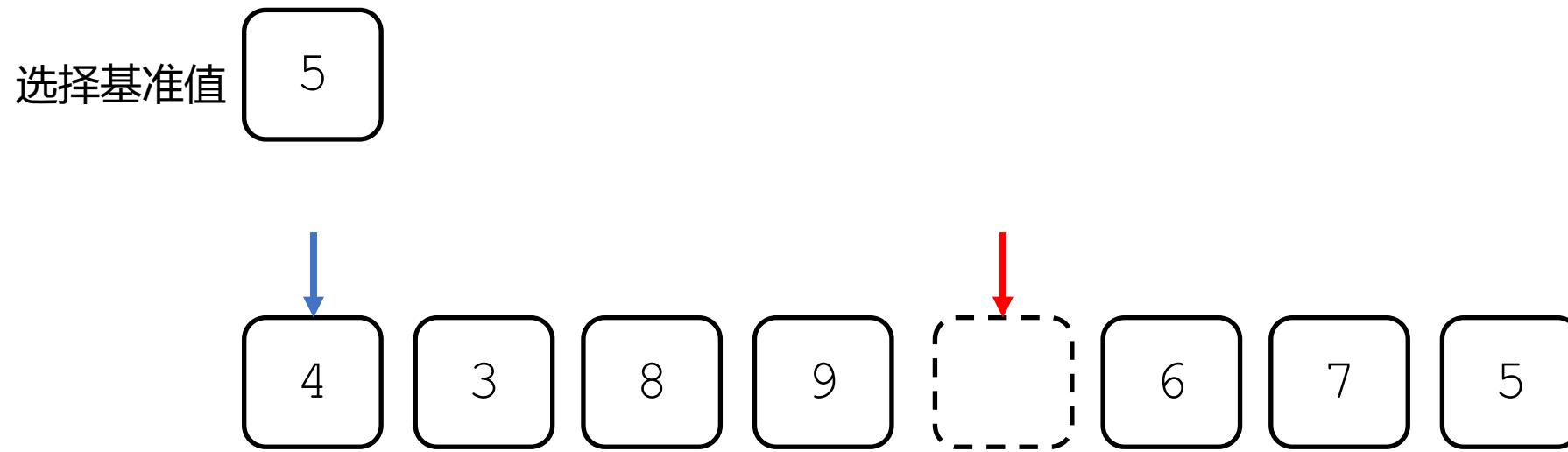


# 快速排序

选择基准值 5

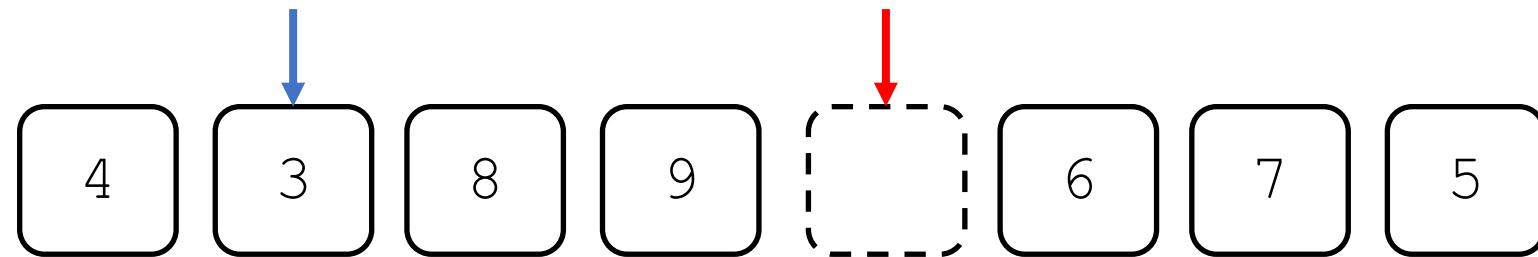


# 快速排序



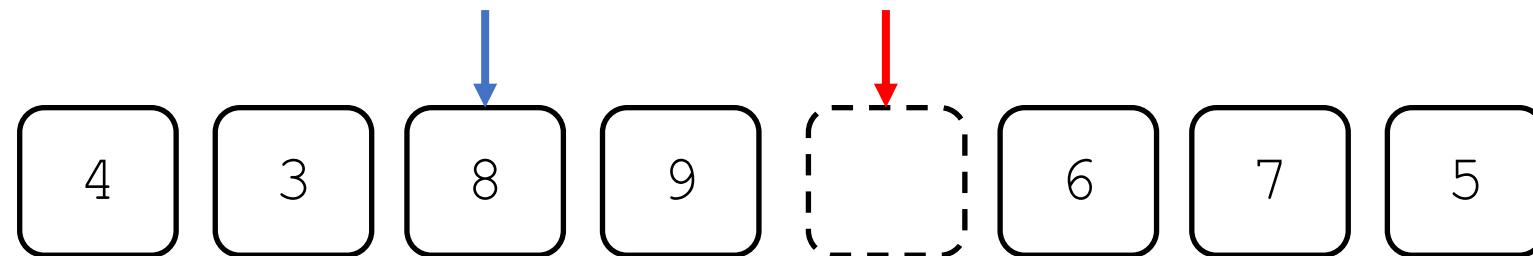
# 快速排序

选择基准值 5



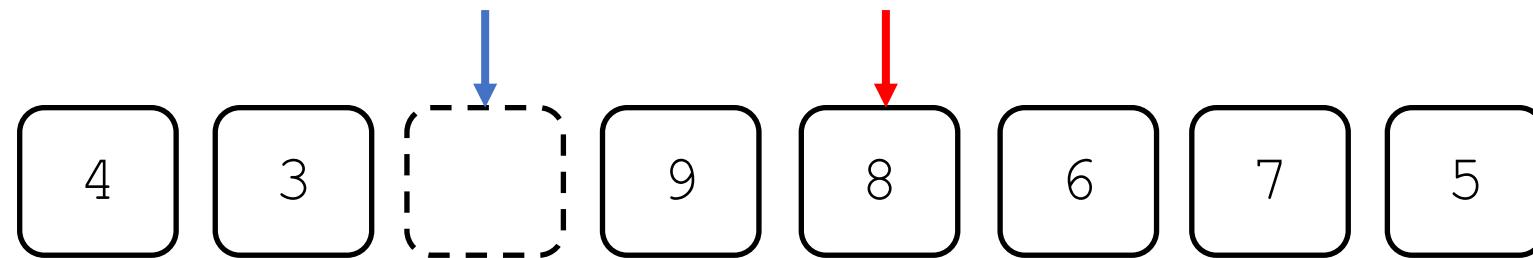
# 快速排序

选择基准值 5



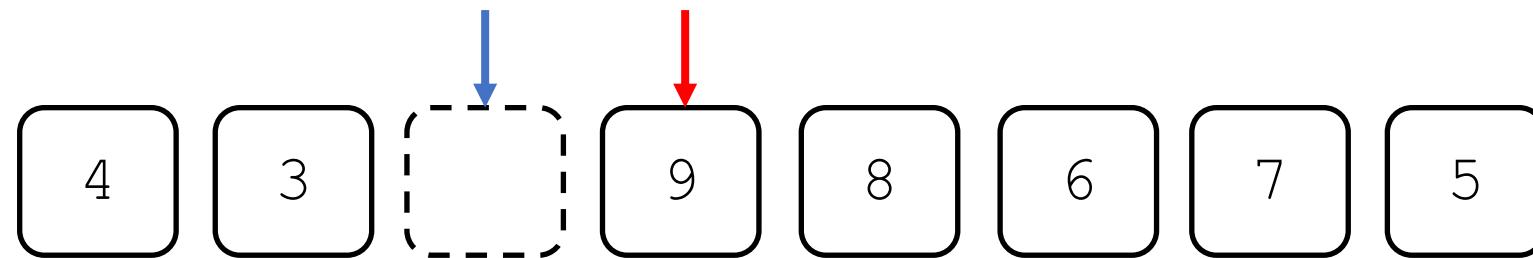
# 快速排序

选择基准值 5

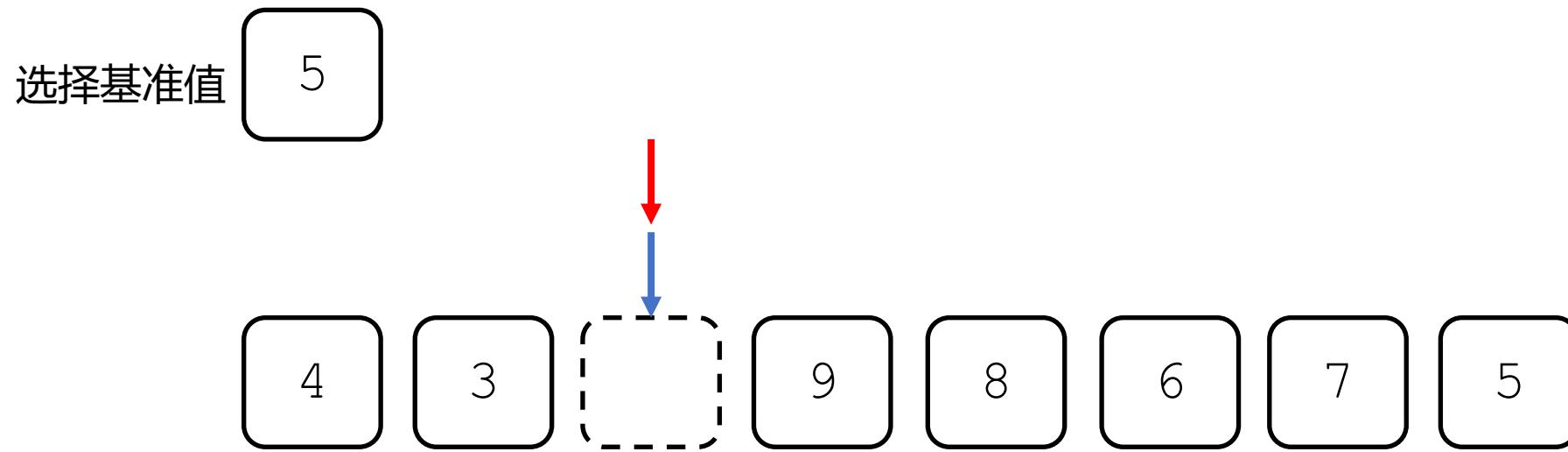


# 快速排序

选择基准值 5

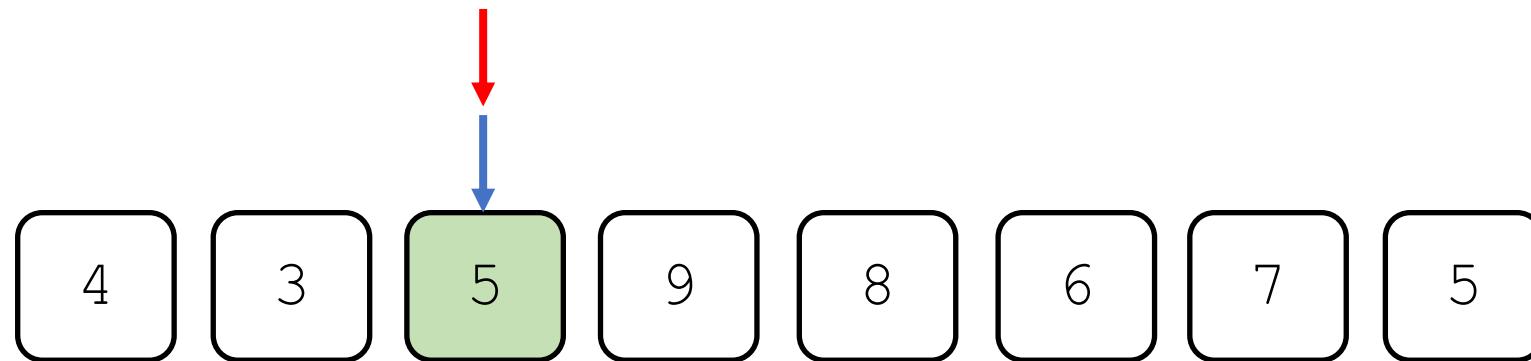


# 快速排序



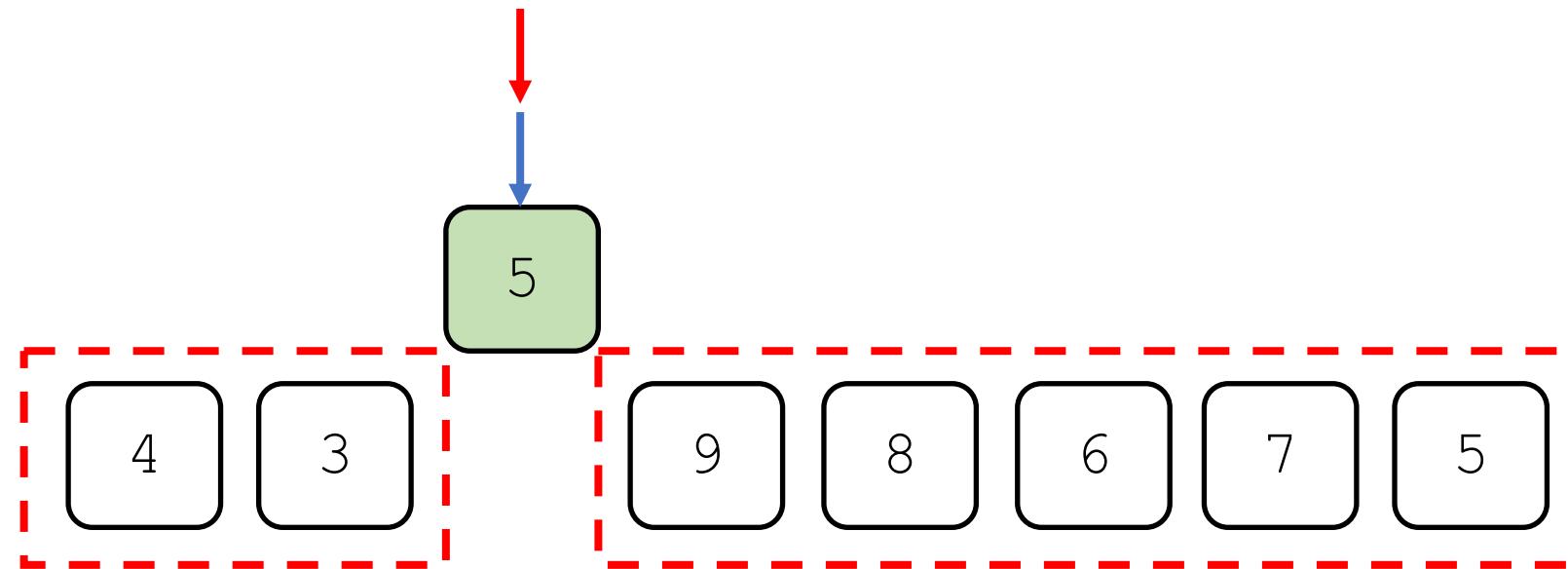
# 快速排序

选择基准值



# 快速排序

选择基准值



# 快速排序

时间复杂度： $O(n \log n) \sim O(n^2)$

X

vim

X

bash

X

bash

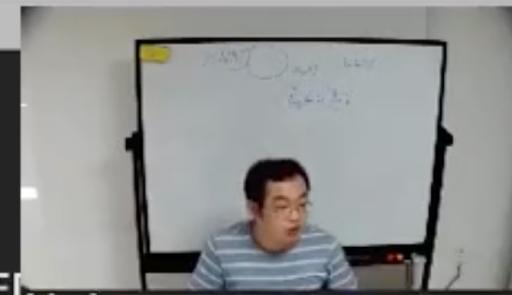
X

bash

X

bash

```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, )
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
```



# 快速排序：代码演示

```
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNode(key);
```

# 四、指针-课后实战项目

1. qsort 函数的使用方法
2. 回调函数的基本概念
3. 快速排序算法讲解
4. 小项目：从0实现 qsort 函数

不要考虑太多，坚持看完，  
你就已经超过了95%的人。

5. 整型数据类型

 | 3.58万次播放

54. 主函数参数

 | 2892次播放