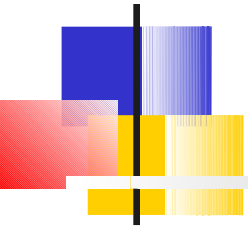
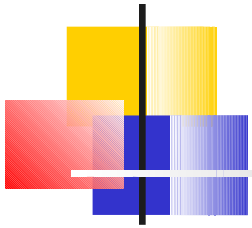
Welcome



# C 指针基础

北京亚嵌教育研究中心

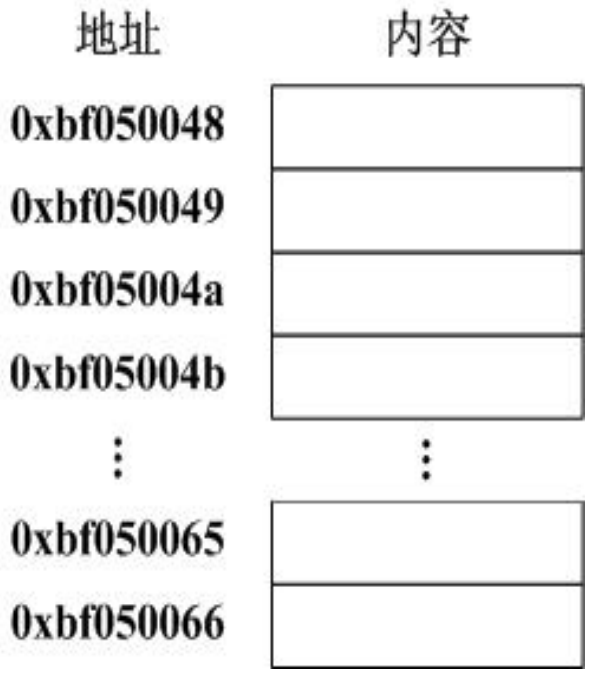


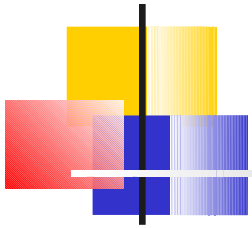
# 本次课程内容大纲

- 指针的基本概念
- 指针类型的参数和返回值
- 指针与数组
- 指针与结构体
- 指向指针的指针与指针数组
- 指向数组的指针与多维数组

# 指针的概念

- 指针 --- 地址
  - 内存是按字节统一编址
    - 顺序编址
    - 每一个字节单元有一个编号
    - 每个字节编号的长度一致



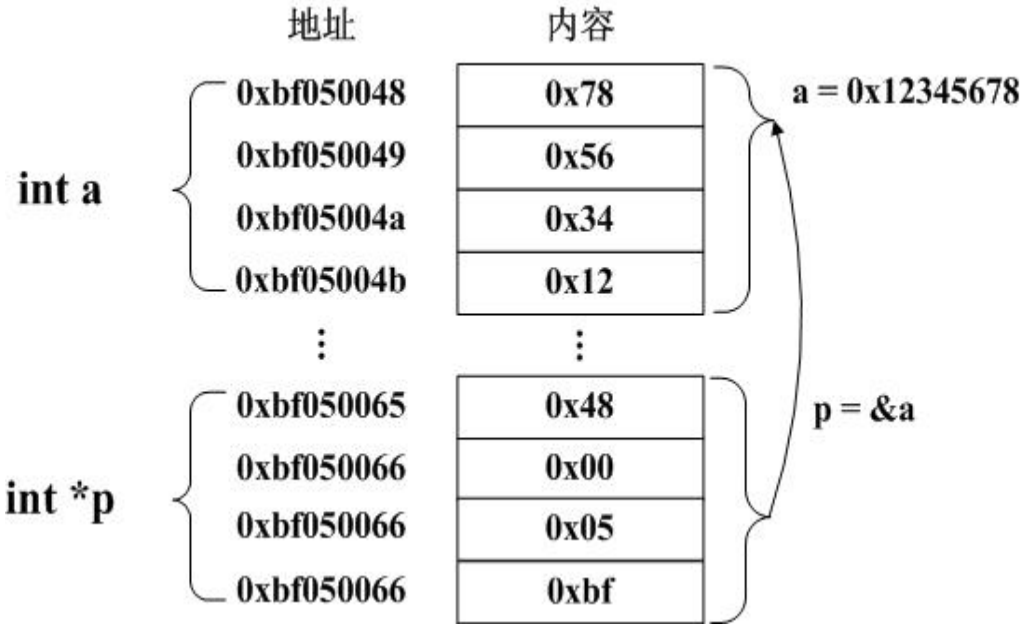


# 指针的概念

- 指针变量 -- 存放地址值的变量
  - 某个内存单元可以存放一个地址值
- 存放地址值的单元应为 4 字节
（ 32 位系统中指针变量的大小）

# 指针的概念

- 指针变量定义
  - 类型名 * 变量名：类型名和 * 一起构成类型指针类型
- int a， *p;
- a = 0x12345678;
- p= &a;
  - 注意区分
- typedef int *p_t;
- #define p char *



# 指针的概念

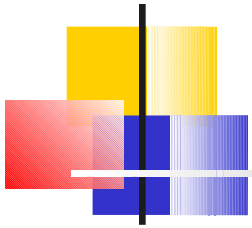
- 指针变量定义
- 指针变量的类型很重要
  - 能够决定存放的指针指向的数据类型
  - 决定指针变量进行加减常数时实际改变的字节数

int *p （只能存放整形数据的地址）

p + 1: 实际改变 4 字节

char *q( 只能存放字符型数据的地址 )

q + 1: 实际改变 1 字节



# 指针的概念

下列程序正确吗？

取变量 a 和 b 的地址

```
int a, b;
int *p1, p2;
p1 = &a;
p2 = &b;
```

# 指针的概念

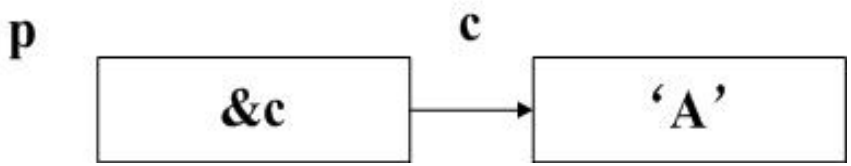
- 指针变量的运算
  - 赋值
- 指针变量定义之后，应指向可操作的空间
  - int *p, a;  // 局部变量
  - p = &a;  // 若该步省略，则 p 为野指针
  - *p = 10;
- 相同类型的指针变量可进行赋值操作
  - int *p, *q, a;
  - p = &a;
  - q = p;

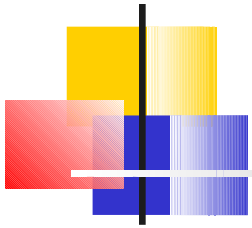
指针变量本身总是可读写的

# 指针的概念

- 指针变量的运算
  - 间接引用：访问指针指向的空间内容

```
char *p, c
p = &c;
*p = 'A';
```





# 指针的概念

- 泛型指针 void *
- void *p: sizeof(p) = 4
- 泛型指针可以和任意类型的指针完成隐式转换

# 指针的概念

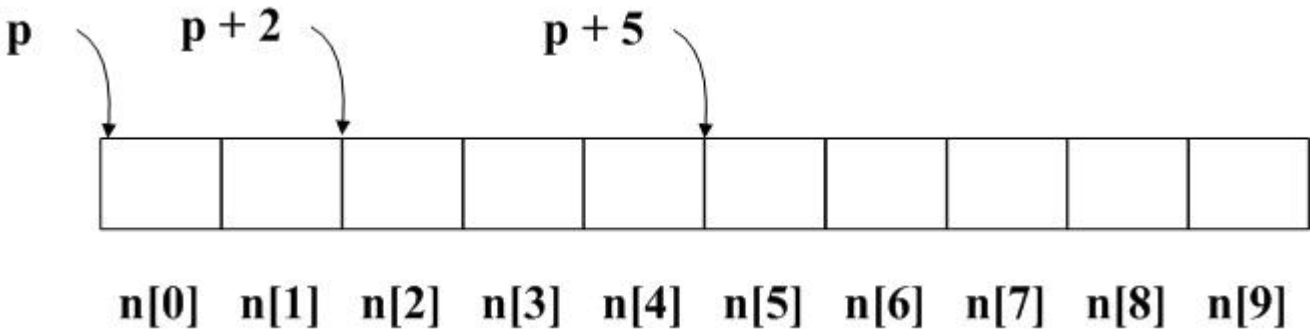
- 注意：可以定义 void * 类型的变量，不能定义 void 类型的变量，因为系统不知道应该给 void 类型变量分配多少字节空间，同样不可以直接对 void * 型的指针进行间接引用操作。
- void *p = malloc(10);
- char *q = p;
- *q = ‘A’;

# 指针的概念

- 练习
  - 猜数：首先生成一个介于 1-100 之间的整数，从键盘不断输入数值直到该数值与生成的那个数相符为止，并打印出猜数的次数

# 指针与数组

- 指针存放数组元素的地址
  - int n[10], *p;
  - p = &n[0];



# 指针与数组

- 指针运算
  - 指针加减整常数

```
int n[10], *p, *q;
p = n;
p = p + 2;
q = n + 2;
```

  - 同类型指针变量相减、比较

```
int n[10], *p, *q;
p = n;
q = n + 2;
printf("%d\n", (int)(q - p));
```

# 指针与数组

- 数组的取下标操作与指针的间接应用操作等价

int n[10], *p;
p = n;
n[0]  等价于  *p;
n[1]  等价于  *(p + 1)  等价于 1[p]

区分:
*p++, *(p++), (*p)++
*++p, *(++p)

# 指针与数组

- 数组作为函数参数
  - 数组作为函数参数时，该参数实际是指针

```
void func(int num[])
```

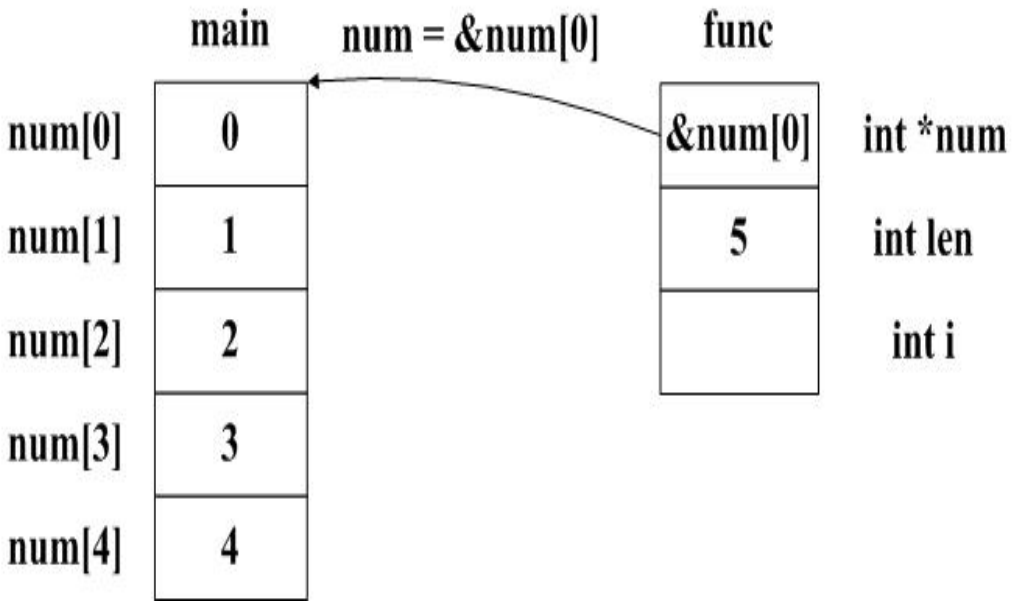
等价于

```
void func(int *num)
```

# 指针与数组

- 数组作为函数参数

```
void func(int num[], int len)
{
    int i;
    for(i = 0; i < len; i++)
    num[i] += i;
}
int main(void)
{
    int num[5] = {0};
    func(num, 5);
    return 0;
}
```

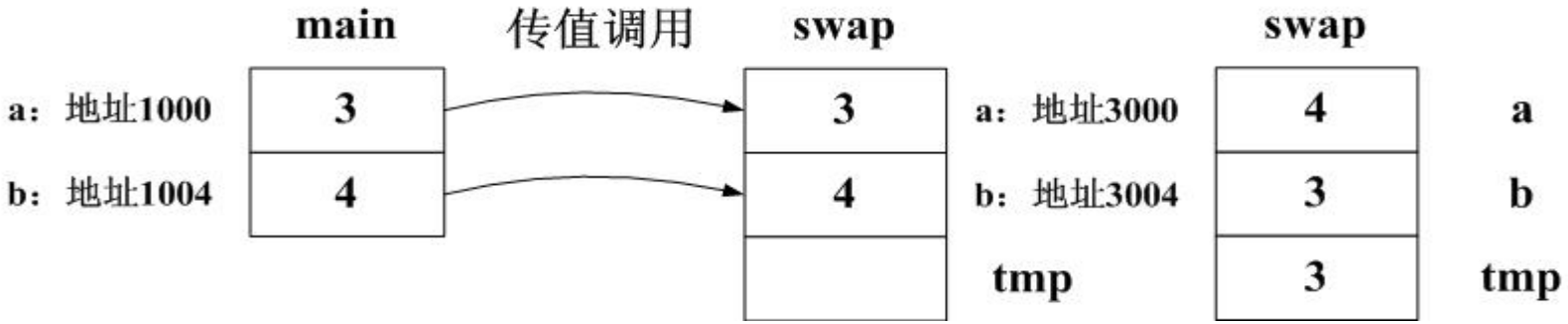


# 指针与数组

- 数组作为函数参数：数据交换

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```
int main(void)
{
     int a = 3, b = 4;
    printf("a = %d, b = %d\n",
    a, b);
    swap(a, b);
    printf("a = %d, b = %d\n",
    a, b);
```

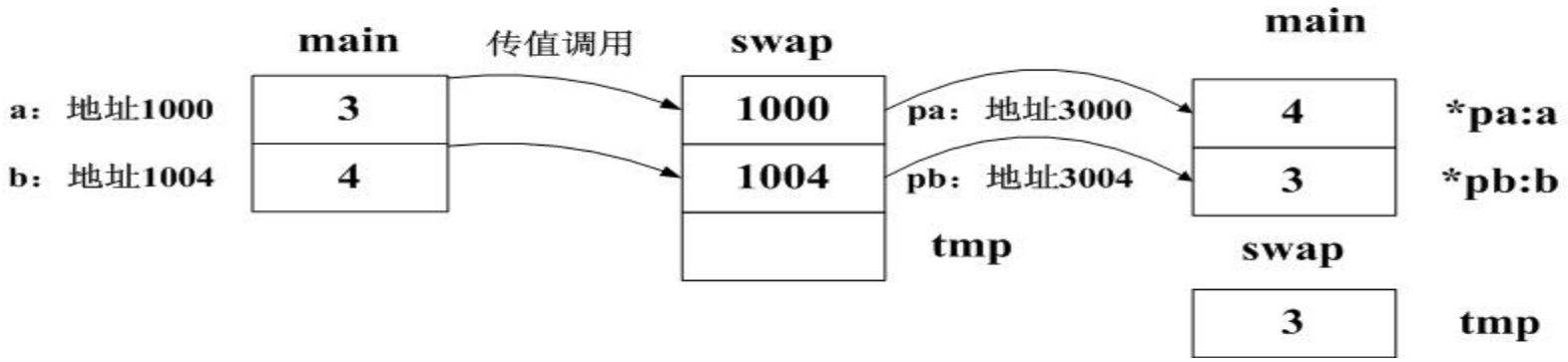


# 指针与数组

- 数组作为函数参数：数据交换

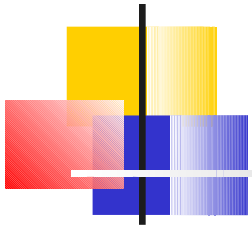
```
void swap(int *pa, int *pb)
{
    int tmp = *pa;
    *pa = *pb;
    *pb = tmp;
}
```

```
int main(void)
{
     int a = 3, b = 4;
    printf("a = %d, b = %d\n",
    a, b);
    swap(&a, &b);
    printf("a = %d, b = %d\n",
    a, b);
```



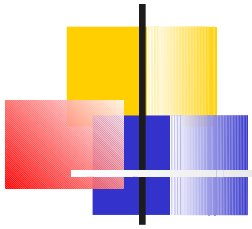
# 指针与数组

- 指针作为参数的这种特性的实际意义：
  - 最重要的一个就是可以实现多个返回值。（从函数带回多个值）
- 函数不能返回数组
- 函数返回指针值：不能返回局部变量的地址



# 指针与数组

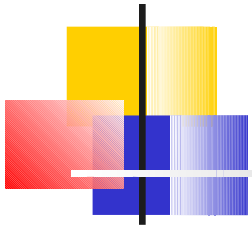
- 练习：
  - 生成无重复的 26 个英文字母
  - 生成 26 个排列好的英文字母



# 指针与结构体

```
struct st{
    int n;
    int num[5];
} s, *p;
p = &s;
```

s.n 等价于 p->n



# 指针与结构体

- 练习

定义结构体： struct student{

```
    int id;
    char *pname;
    int score;
} *pstd;
```

输入 5 条学生信息，根据 score 进行升序排序

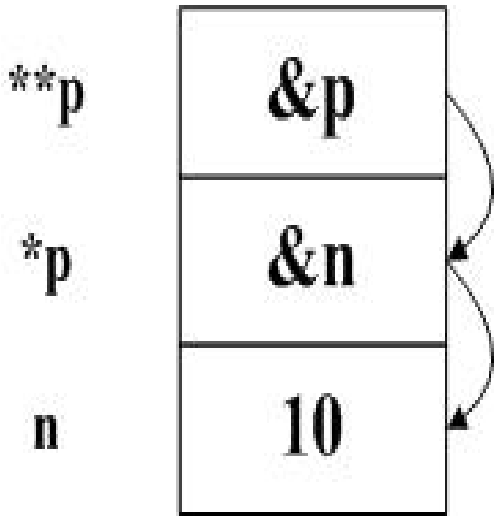
使用 malloc 函数或结构体数组

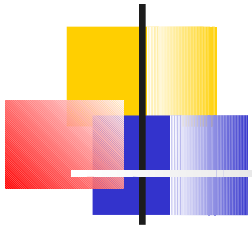
# 指向指针的指针与指针数组

- 指向指针的指针

```
int **pp, *p, n;
pp = &p;
P = &n;
```

- 指针变量一定要先赋予合法地址





# 指向指针的指针与指针数组

- 指针数组：若干个指针构成的集合

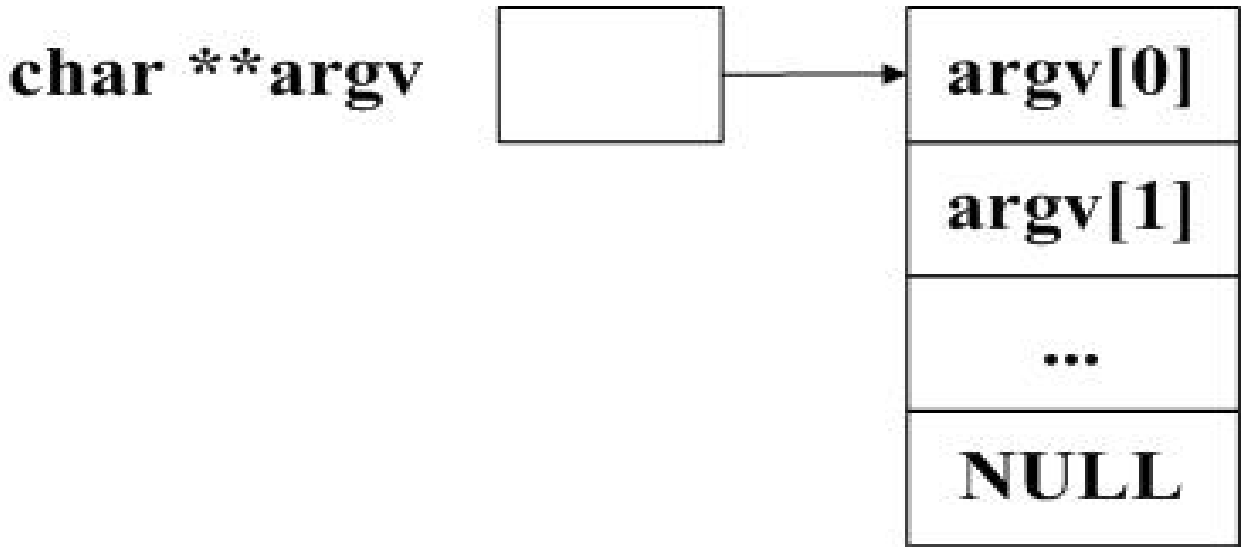
  char *pstr[5];

  数组元素的用法和单个指针变量的用法相同

# 指向指针的指针与指针数组

- 命令行参数：

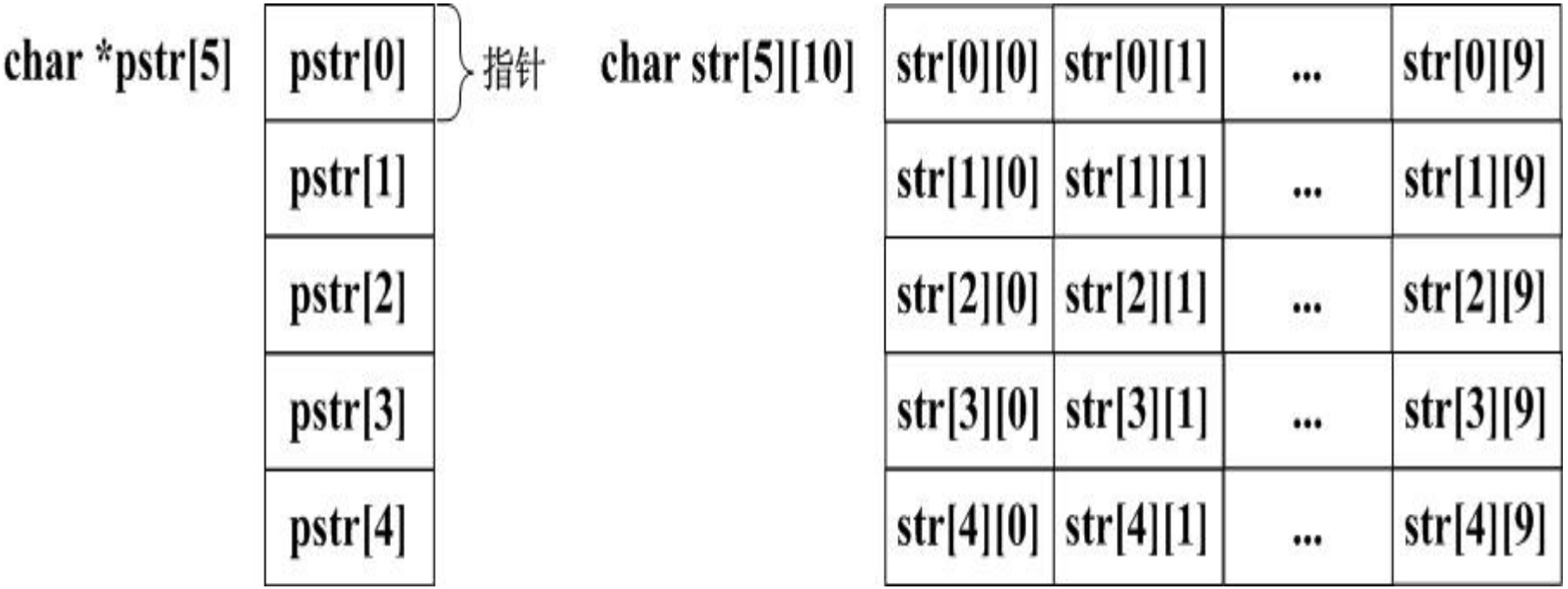
int main(int argc, char *argv[])



# 指向指针的指针与指针数组

- 指针数组与二维数组

char *pstr[5]， str[5][10];



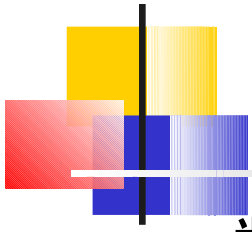
# 指向指针的指针与指针数组

- 练习

  定义一个字符指针数组如：

  char *str[] = {"hello world“, "hello hell“, "hello aka“, "hello hello hoho"};

  从键盘输入字符串，从字符指针数组对应的字符串中查找输入的字符串是否存在，若存在返回该字符串存在于指针数组的行列位置。若输入”exit”（不区分大小写）结束字符串输入



# 指向数组的指针与多维数组

- 数组的指针

```
char  (*p)[16], str[16];
```

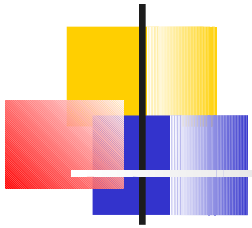
p = &str;

str 代表整个数组空间

&str 表示数组的指针

p + 1 相当于： p + 1 * sizeof(str)

p + 1 会跨过 16 个字节



# 指向数组的指针与多维数组

- 多维数组

  char str[5][10];

  str[0]　　第一行的数据

  str[1]　　第二行的数据

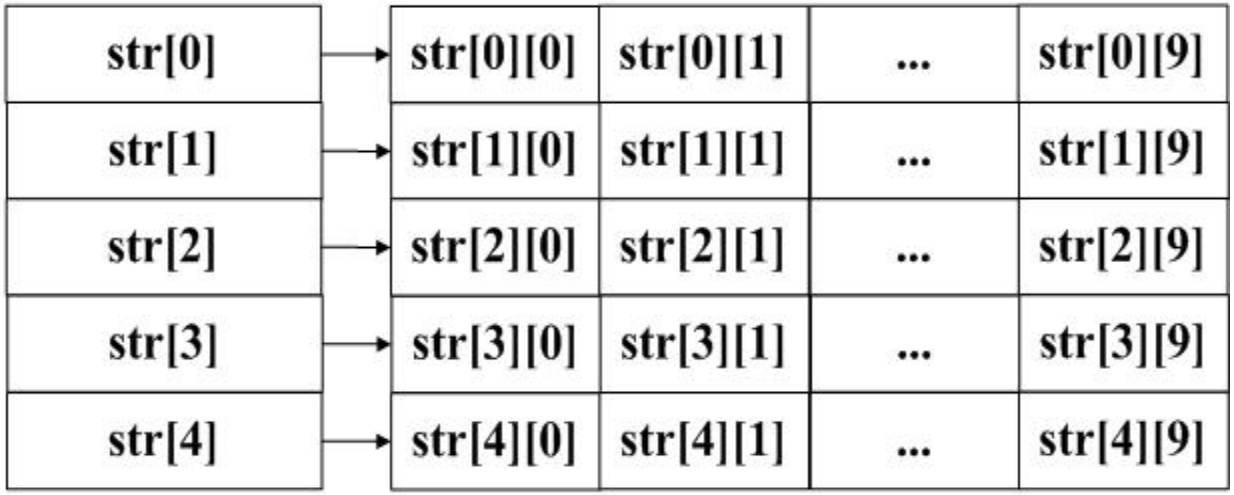
  str[i]　　相当于一个一维数组名

# 指向数组的指针与多维数组

- 多维数组

str:char (*)[10]　　str[i]:char [10]　　str[i][j]:char

char str[5][10]

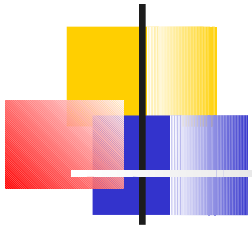



str与&str[0]同类型、数值同

str[0]与&str[0][0]同类型、数值同

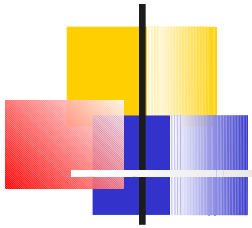
# 指向数组的指针与多维数组

- 练习
  - 对 int num[3][5] 的二维数组进行排序
  - 输入年份以及天数打印这一天是该年的几月几日



# 总结

- 掌握 C 指针的基本用法
  - 定义、运算
  - 指针与数组
  - 指针数组
  - 数组的指针



# Let's DO it!

# Thanks for listening!

☺