

# 指针数组 & 数组指针 & 二级指针

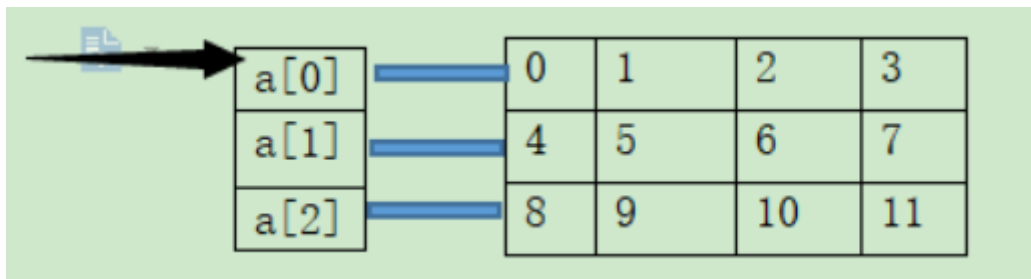
## 二维数组

### 概念

二维数组在概念上是二维的，有行和列，但在内存中所有的数组元素都是连续排列的，它们之间没有“缝隙”。如：

```
int a[3][4] = { {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };
```

从概念上理解，a 的分布像一个矩阵：



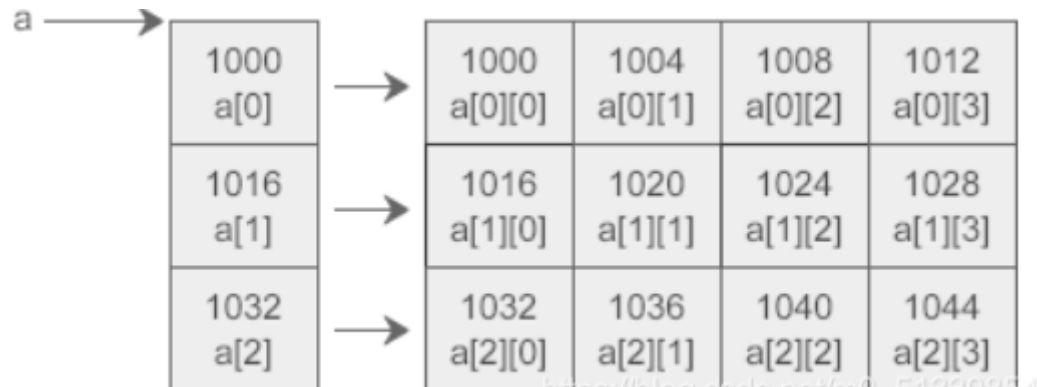
但在内存中，a 的分布是一维线性的，整个数组占用一块**连续的内存**：

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

C语言中的二维数组是**按行排列**的，也就是先存放 a[0] 行，再存放 a[1] 行，最后存放 a[2] 行；每行中的 4 个元素也是依次存放。数组 a 为 int 类型，每个元素占用 4 个字节，整个数组共占用  $4 \times (3 \times 4) = 48$  个字节。

C语言允许把一个二维数组分解成多个一维数组来处理。对于数组 a，它可以分解成三个一维数组，即 a[0]、a[1]、a[2]，而且 C 语言规定，a[0]、a[1]、a[2] 分别是这三个一维数组的数组名。每一个一维数组又包含了 4 个元素，例如 a[0] 包含 a[0][0]、a[0][1]、a[0][2]、a[0][3]。

假设数组 a 中第 0 个元素的地址为 1000，那么每个一维数组的首地址如下图所示：



# 首地址和数组名

在一维数组中，数组名表示的是数组第一个元素的地址，那么二维数组呢？二维数组就是一维数组，二维数组 `a[3][4]` 就是有三个元素 `a[0]`、`a[1]`、`a[2]` 的一维数组，所以数组 `a` 的第一个元素不是 `a[0][0]`，而是 `a[0]`，所以数组名 `a` 表示的不是元素 `a[0][0]` 的地址，而是 `a[0]` 的地址，即

```
a == &a[0]
而 a[0] 又是 a[0][0] 的地址，即：
a[0] == &a[0][0]
所以二维数组名 a 和元素 a[0][0] 的关系是：
a == &(a[0][0])
```

由此可知，二维数组名 `a` 是地址的地址，必须两次取值才可以取出数组中存储的数据。

## 数组(的)指针（行指针）

```
int (*p)[n];
```

`n`代表`p`指向的数组的最大长度

( )优先级高，首先说明`p`是一个指针（二级），只指向一个整型的一维数组首地址，这个一维数组的长度是`n`。也就是说执行`p++`时，`p`要跨过`n`个整型数据的长度。

如要将二维数组赋给一指针，应这样赋值：

```
int a[3][4];
int (*p)[4]; //该语句是定义一个数组指针，指向含4个元素的一维数组。
p=&a[0]; 或者 p=a //将该二维数组的首地址赋给p，也就是a[0]或&a[0][0]
p++;      //该语句执行过后，也就是p=p+1;p跨过行a[0][]指向了行a[1][]
```

## 指针(的)数组

```
int *(p[n])
```

`n`代表一共有`n`个指针类型的元素

[ ]优先级高，先与`p`结合成为一个数组，再由`int*`说明这是一个整型指针数组，它有`n`个指针类型的数组元素。这里执行`p++`是错误的，这样赋值也是错误的：`p=a`；因为`p`是个不可知的表示，只存在`p[0]`、`p[1]`、`p[2]...p[n-1]`，而且它们分别是指针变量可以用来存放变量地址。但可以这样 `p=a`；这里`p`表示指针数组第一个元素的值，`a`的首地址的值。

如要将二维数组赋给一指针数组：

```
int *p[3];
int a[3][4];
for(i=0;i<3;i++)
p[i]=a[i];
```

这里int \*p[3] 表示一个一维数组内存放着三个指针变量，分别是p[0]、p[1]、p[2] 所以要分别赋值。

## 数组指针和指针数组在二维数组上的应用

```
int nums[2][2] = {
    {1, 2},
    {2, 3}
};
//1.数组指针
int (*p)[2] =nums;
//2.或者此时 nums[0]、和 nums[1]各为一个指针数组
int *p[2] = {nums[0], nums[1]};
```

第一种是数组指针：int (\*p)[2]=nums ;语句是定义一个指针变量，指向含2个元素的第一个一维数组。所以p并不是指的是某一个元素，而是指的是一个数组，所以我们应该把第一个数组的地址赋给p所以在赋初值时int (\*p)[2] =&nums[0] ;或者是int (\*p)[2] =nums ;同时\*(p+1) 指向了第二个数组，也就是nums[1];此外，nums[1][0]=\*(\*(p+1)+0)=\*(\*(p+1)=\*\*\*(p+1) ,所以(a+i)+j == &a[i][j]，即\*(\*(a+i)+j) == a[i][j]。

第二种是指针数组，int \*p[2] = {nums[0], nums[1]} ;p是包含两个指针元素的数组，指针指向的是int 型，同时两个指针分别指向nums[0][0]和nums[1][0]的地址，所以在赋初值时，int \*p[2] = {nums[0], nums[1]} ; 或者int \*p[2] = {nums[0], nums[1]} ; int \*p[2] = {&nums[0][0], &nums[1][0]} ;

# 代码分析

```
#include<stdio.h>
int main()
{
    int nums[2][2] = {
        {1, 2},
        {2, 3}
    };

    //此时 nums[0]、和 nums[1]各为一个数组
    int *p[2] = {nums[0], nums[1]};

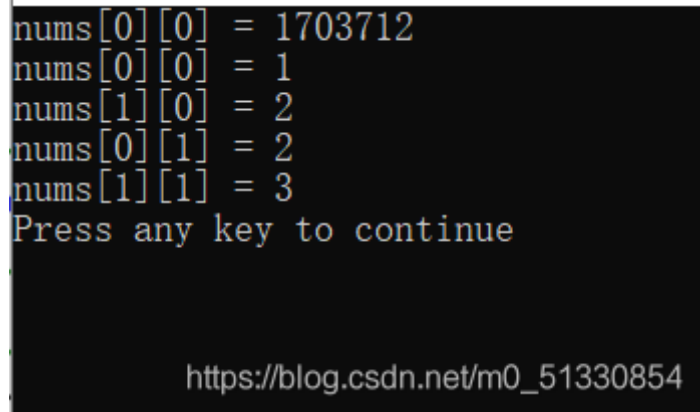
    //我们可以用指针数组 p 操作一个二维数组

    printf("nums[0][0] = %d\n", *p);
    printf("nums[0][0] = %d\n", **p);

    //指针 + 整数形式, p+1 移动到 nums 的地址, *(p + 1) = nums[1], 则**(p + 1) = nums[1][0]
    printf("nums[1][0] = %d\n", *(p + 1));

    //先*p = nums[0], 再*p + 1 = &nums[0][1], 最后获取内容*(*p + 1)即为 nums[0][1]
    printf("nums[0][1] = %d\n", *(*p + 1));
    printf("nums[1][1] = %d\n", *(*p + 1)+1));
    return 0;
}
```

输出结果：



```
nums[0][0] = 1703712
nums[0][0] = 1
nums[1][0] = 2
nums[0][1] = 2
nums[1][1] = 3
Press any key to continue

https://blog.csdn.net/m0\_51330854
```