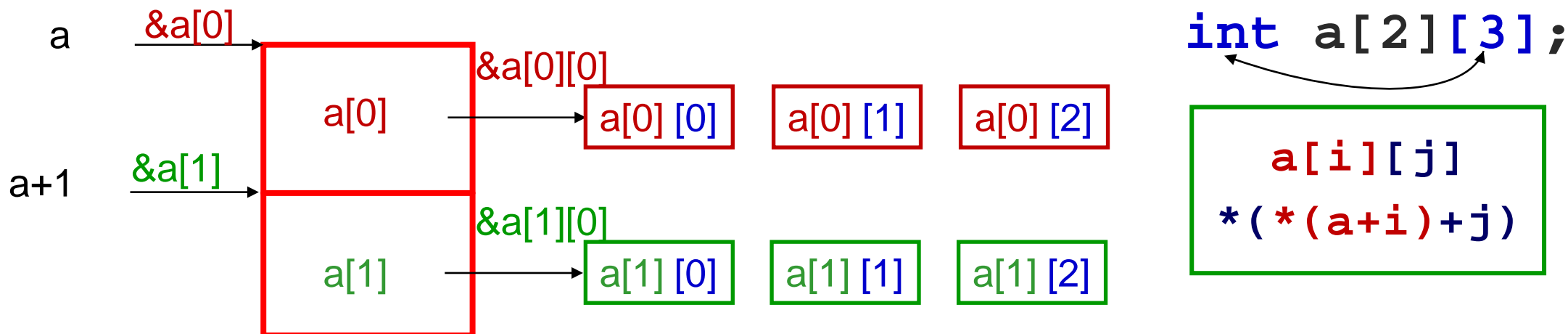


换个角度看二维数组

- 将二维数组a看成一维数组，有2个“`int[3]型`”元素



`a`代表二维数组的首地址，第0行的地址，行地址

`a+i`代表第*i*行的地址

但并非增加*i*个字节！

`a[i] ↔ *(a+i)`

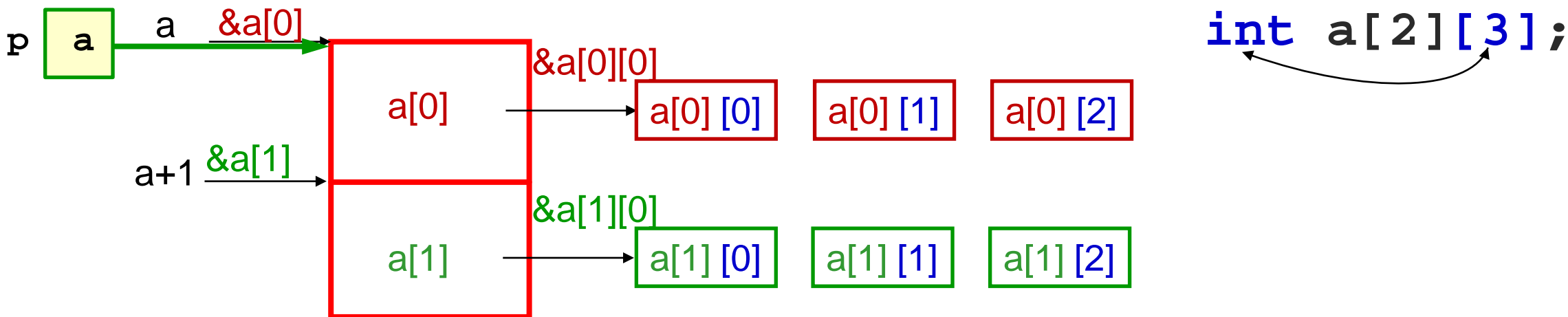
`&a[i] ↔ (a+i)`

1) `a`包含2个元素`a[0]`, `a[1]`

2) `a[0]`, `a[1]`又分别是一个一维数组，包含3个元素

指向二维数组的行指针

- 将二维数组a看成一维数组，有2个“`int[3]型`”元素



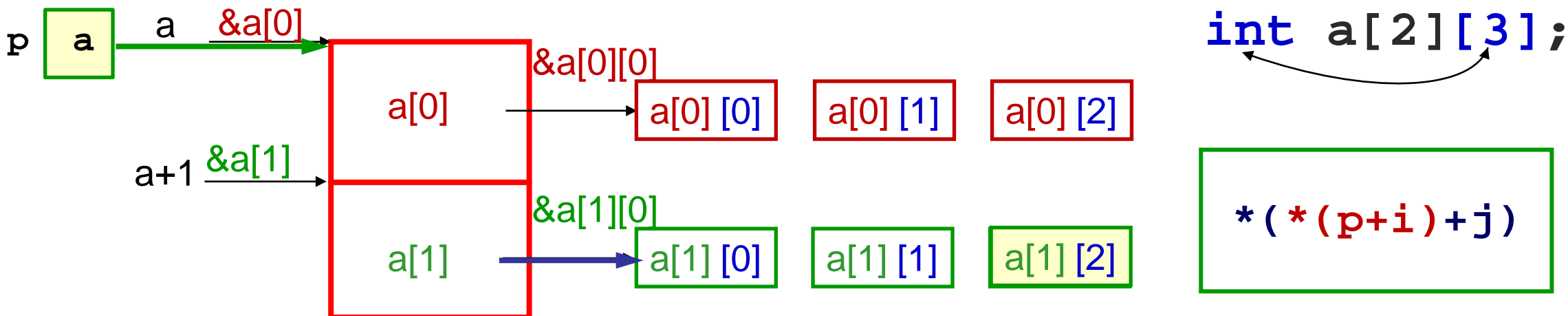
- 若要让一个指针指向它，则应定义为

`int (*p)[3];` //行指针，基类型“`int[3]型`” `int *p[3];` //指针数组

`p = a;` //`&a[0]`指向第0行的“`int[3]型`”元素

指向二维数组的行指针

- 将二维数组a看成一维数组，有2个“`int[3]`型”元素



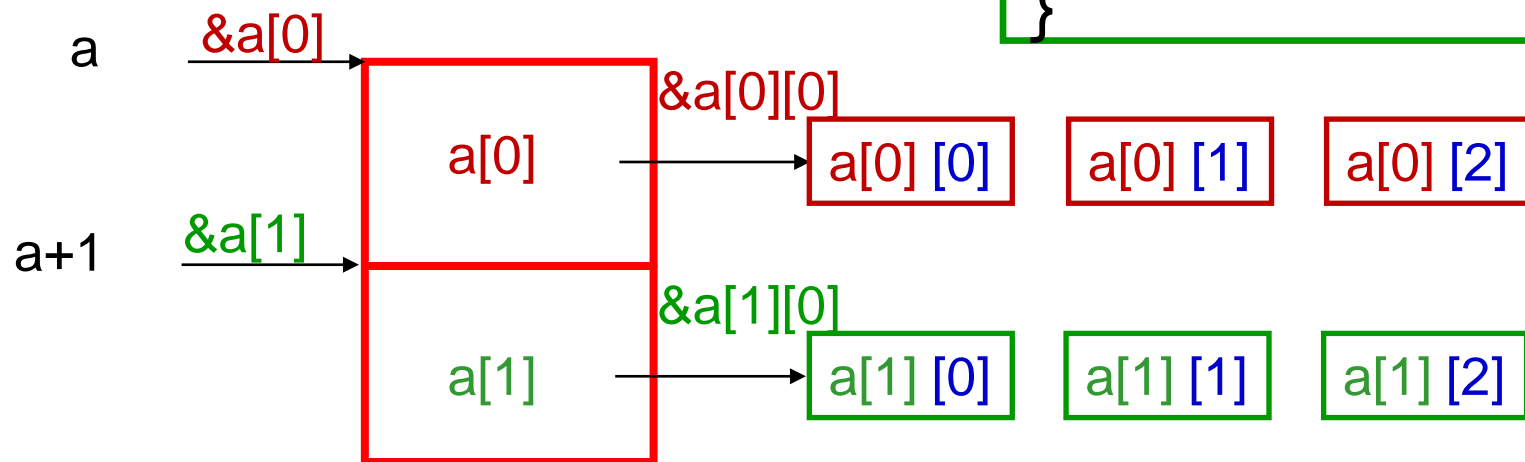
- `p+i` 指向第*i*行的“`int[3]`型”元素，即`&a[i]`
- `*(p+i)`，即`a[i]`，第*i*行的数组名，即指向第*i*行第0列的`int`型元素
- `*(p+i)+j` 指向第*i*行第*j*列的`int`型元素
- `*(* (p+i)+j)` 取出第*i*行第*j*列的内容（`int`型元素的值），即`a[i][j]`

按行指针访问二维数组元素

■ 逐行查找→逐列查找

```
int (*p)[3];  
p = a;
```

```
for (i=0; i<m; i++) //行下标变化  
{  
    for (j=0; j<n; j++) //列下标变化  
    {  
        printf("%d", *(*p+i)+j);  
    }  
}
```

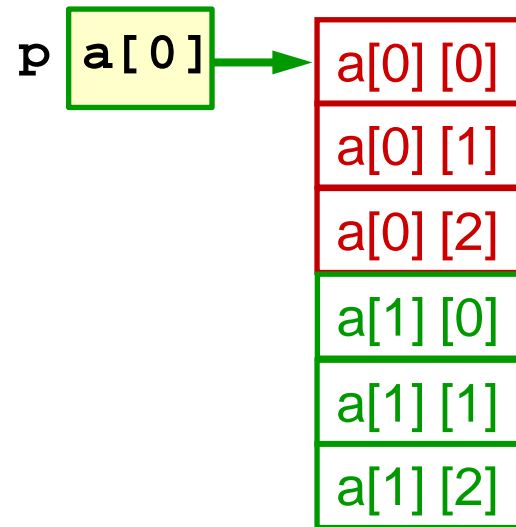
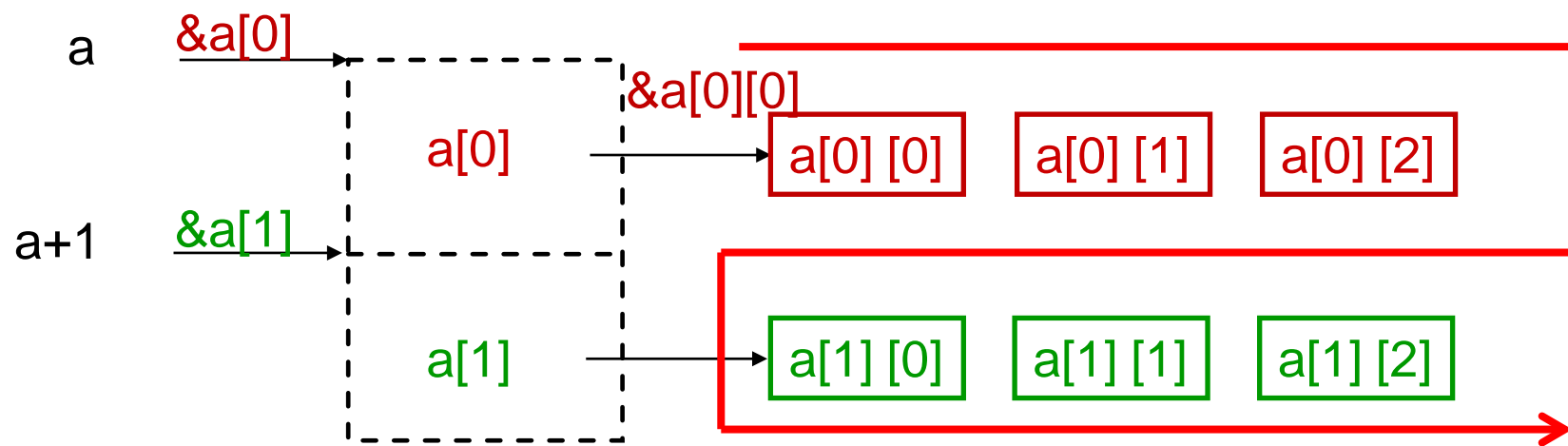


```
int a[2][3];
```

```
*(*p+i)+j
```

再换一个角度看二维数组

- 将二维数组a看成一维数组，有6个int型元素



- 若要让一个指针指向它，则应定义为

`int *p;` //列指针，基类型是int型

`p = a[0];`
`p = *a;`
`p = &a[0][0];`

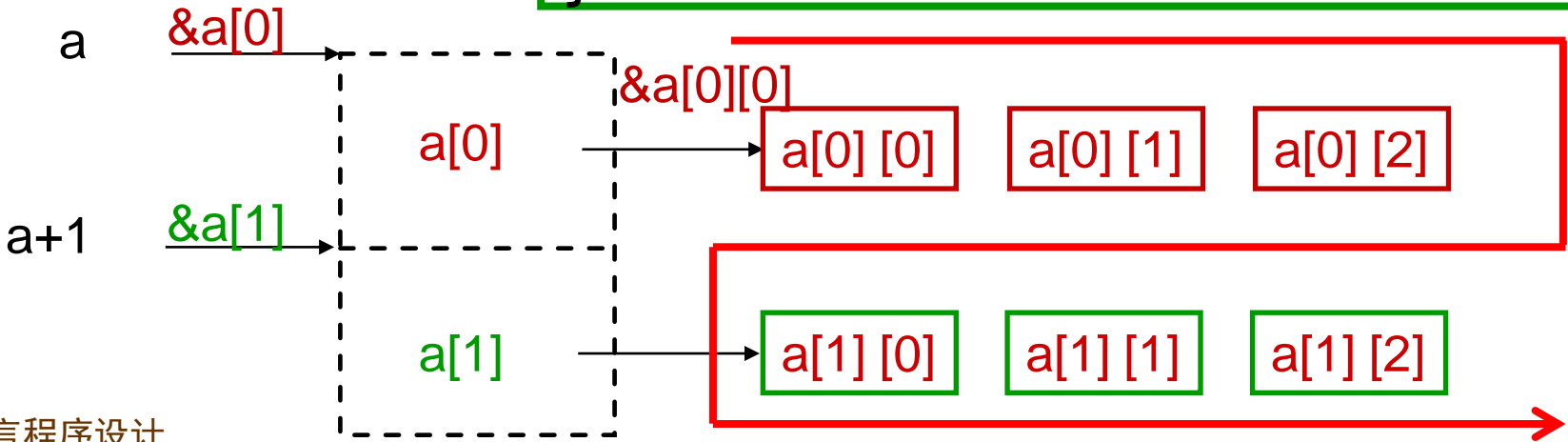
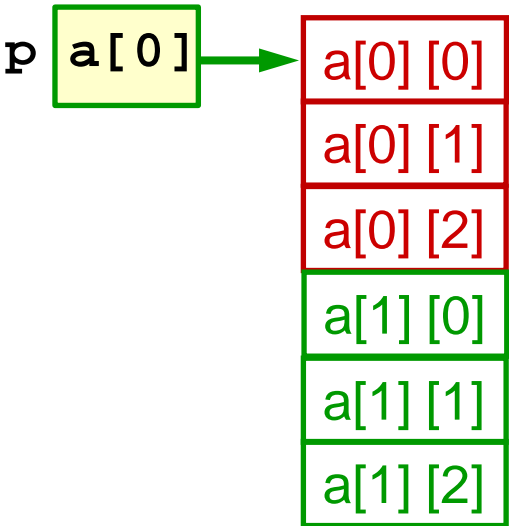
`p = a[0];` //*(a+0)+0即&a[0][0]，指向第0行第0列的int型元素

按列指针访问二维数组元素

- 根据相对偏移量逐列查找

```
int *p;  
p = &a[0][0];
```

```
for (i=0; i<m; i++)  
{  
    for (j=0; j<n; j++)  
    {  
        printf( "%d", *(p+i*n+j) );  
    }  
}
```



```
*(p+i*n+j)  
p[i*n+j]
```

二维数组作函数参数

```
void InputArray(int p[][N], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", &p[i][j]);
        }
    }
}
```

```
InputArray(a, 3, 4);
OutputArray(a, 3, 4);
```

形参声明为二维数组，
列数必须为常量

```
void OutputArray(int p[][N], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            printf("%4d", p[i][j]);
        }
        printf("\n");
    }
}
```

指向二维数组的**行**指针作函数参数

```
void InputArray(int (*p)[N], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", *(p+i)+j);
        }
    }
}
```

```
InputArray(a, 3, 4);
OutputArray(a, 3, 4);
```

形参声明为二维数组的**行**指针，
列数必须为常量

```
void OutputArray(int (*p)[N], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            printf("%4d", (*(p+i)+j));
        }
        printf("\n");
    }
}
```


指向二维数组的列指针作函数参数

```
void InputArray(int *p, int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", &p[i*n+j]);
        }
    }
}
```

行指针：传入的参数为a
列指针：传入的参数为*a

`InputArray(*a, 3, 4);`
`OutputArray(*a, 3, 4);`

形参声明为二维数组的列指针，
列数可为变量

```
void OutputArray(int *p, int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            printf("%4d", p[i*n+j]);
        }
        printf("\n");
    }
}
```

问题：
`p[i*N+j];`
v.s.
`p[i*n+j];`

从for循环的角度去理解
存储形式

用 **const** 保护你传给函数的数据

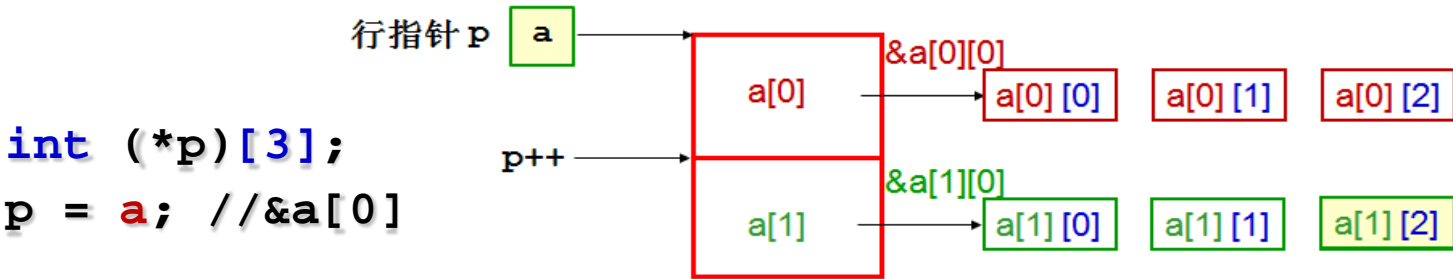
```
void OutputArray(const int p[][N], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            printf("%4d", p[i][j]);
        }
        printf("\n");
    }
}
```

若只向函数传数据，则把形参定义为 **const**，防止它被意外修改

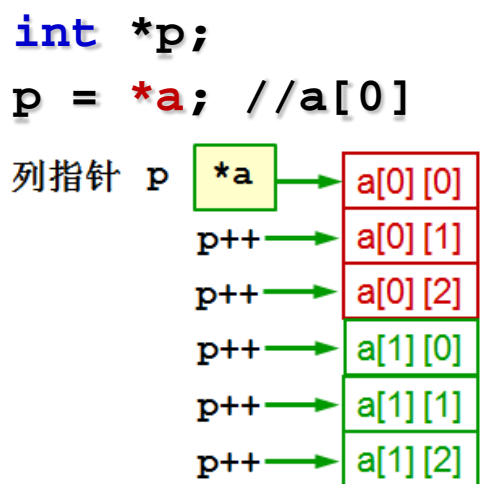
```
void OutputArray(const int *p, int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            printf("%4d", p[i*n+j]);
        }
        printf("\n");
    }
}
```

小结

- 指针与二维数组间的关系的关键
 - 理解二维数组的行指针和列指针
 - 二维数组在内存中按行存储，但可以以两种方式看待它
 - 一个x型的指针指向x型的数据
 - $a[i] \leftrightarrow *(a+i)$



$$a[i][j] \leftrightarrow *(*(p+i)+j)$$



$$a[i][j] \leftrightarrow *(p+i*n+j)$$