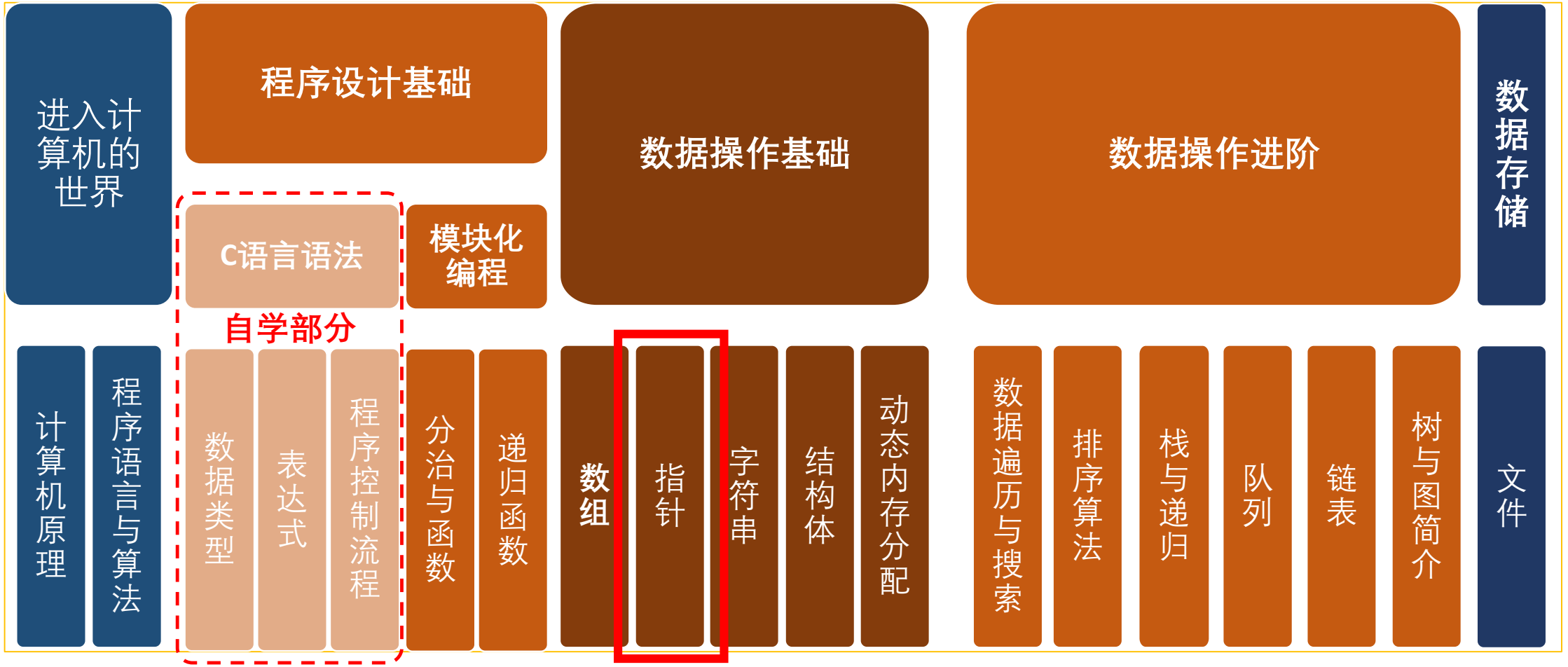
计算思维与实践

哈尔滨工业大学（深圳）
计算机科学与技术学院
大数据技术中心
张保权

课程内容安排



程序设计思想与数据操作方法融会贯通，内容由浅入深

第八讲 指针——学习内容

8.1 指针与二维数组
8.2 指针数组与指向数组的指针区别
8.3 函数与指针

第八讲 指针——学习内容

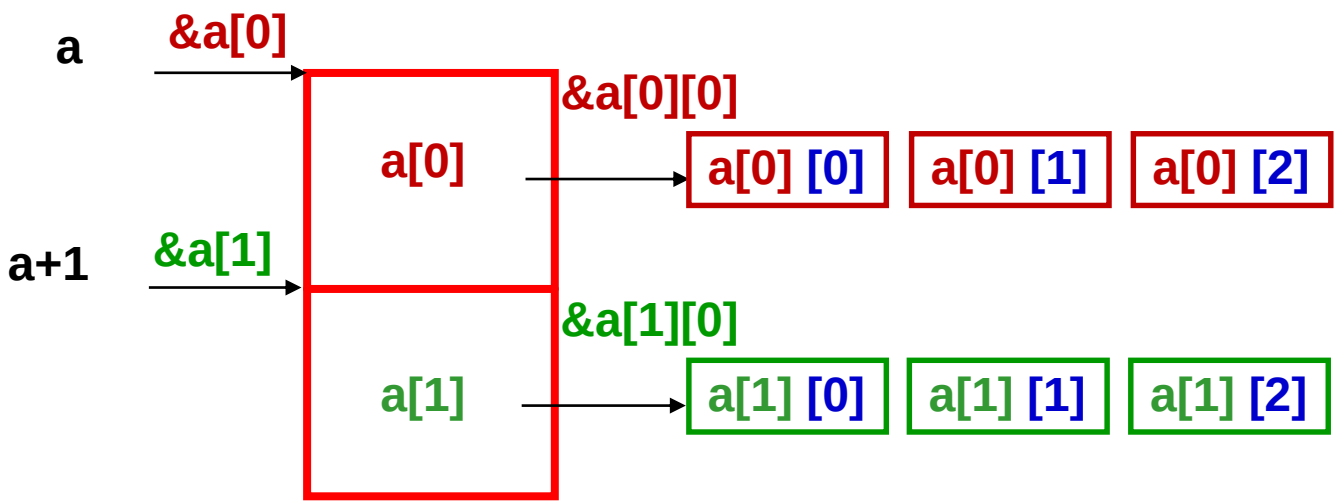
8.1 指针与二维数组

8.2 指针数组与指向数组的指针

8.3 函数与指针

8.1 指针与二维数组

- 将二维数组a看成一维数组，有2个“int[3]型”元素




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

a[i][j]

((a+i)+j)

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！

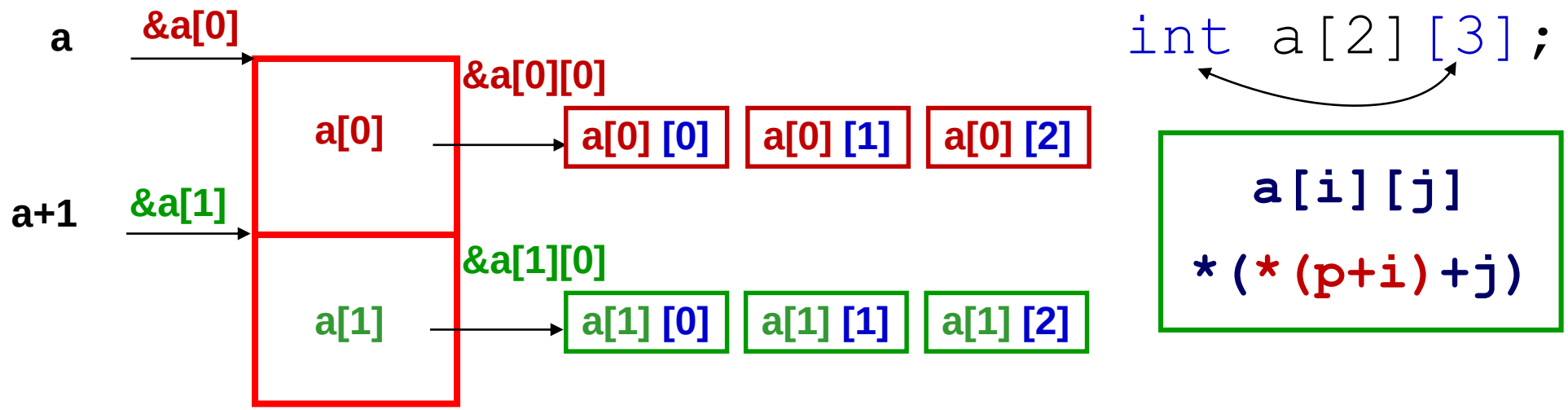
a[i] ↔ *(a+i)

&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

8.1 指针与二维数组



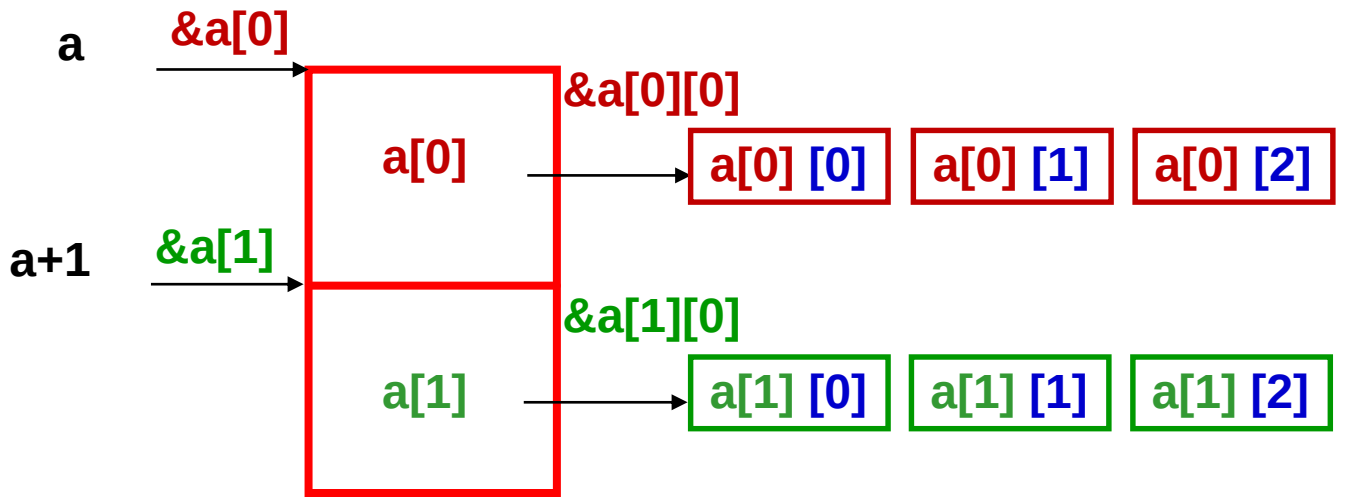
- 若要让一个指针指向它，则应定义为
  - `int (*p)[3];` //**行指针**，基类型是`int[3]`
  - `p = a;`
  - `p = &a[0];` //指向**第0行**的“`int[3]`型”元素

8.1 指针与二维数组

- 逐行查找→
- 逐列查找

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

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



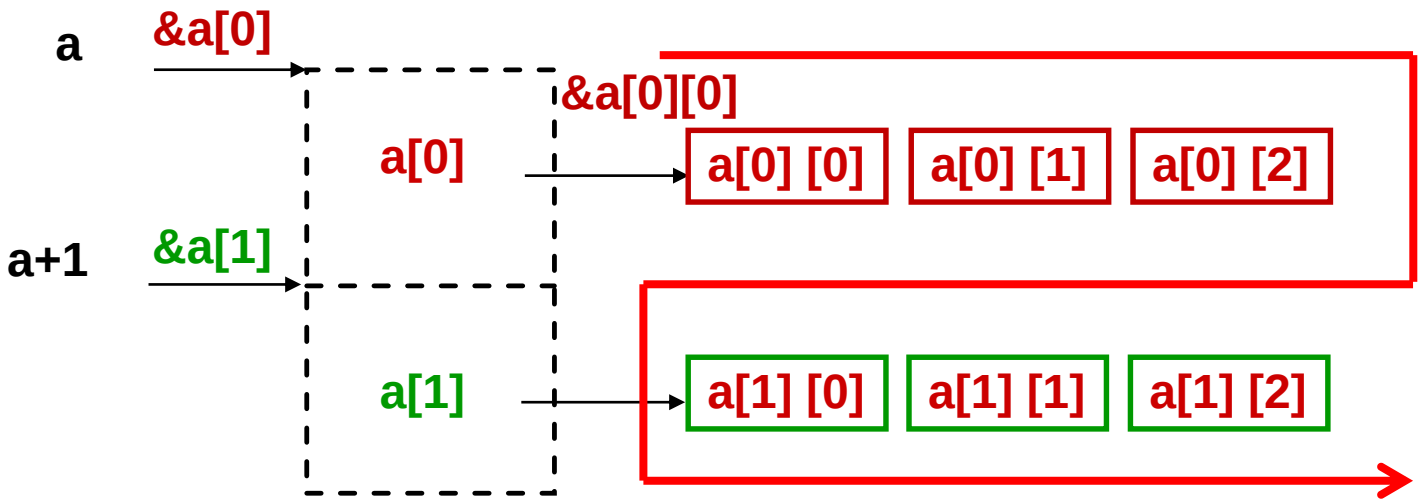

```
int a[2][3];
```

a[i][j]

((p+i)+j)

8.1 指针与二维数组

- 将二维数组a看成一维数组，有6个int型元素

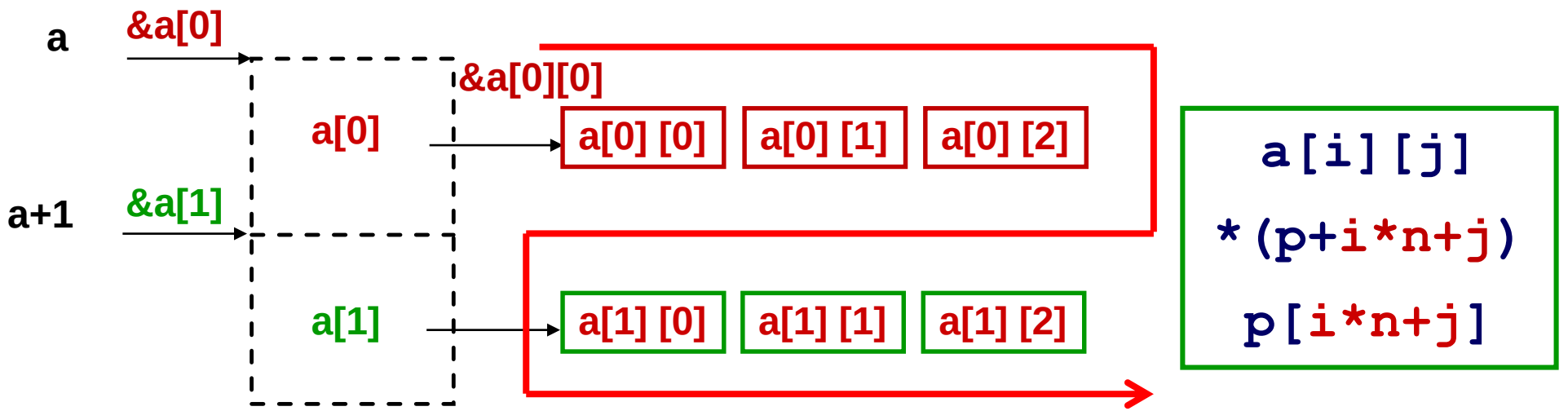


*(a + i)即 a[i]代表第i行第0列的地址，列地址

*(a+i)+j 即 a[i]+j
代表第i行第j列的地址&a[i][j]

((a+i)+j) 即 a[i][j]
代表第i行第j列的内容

8.1 指针与二维数组



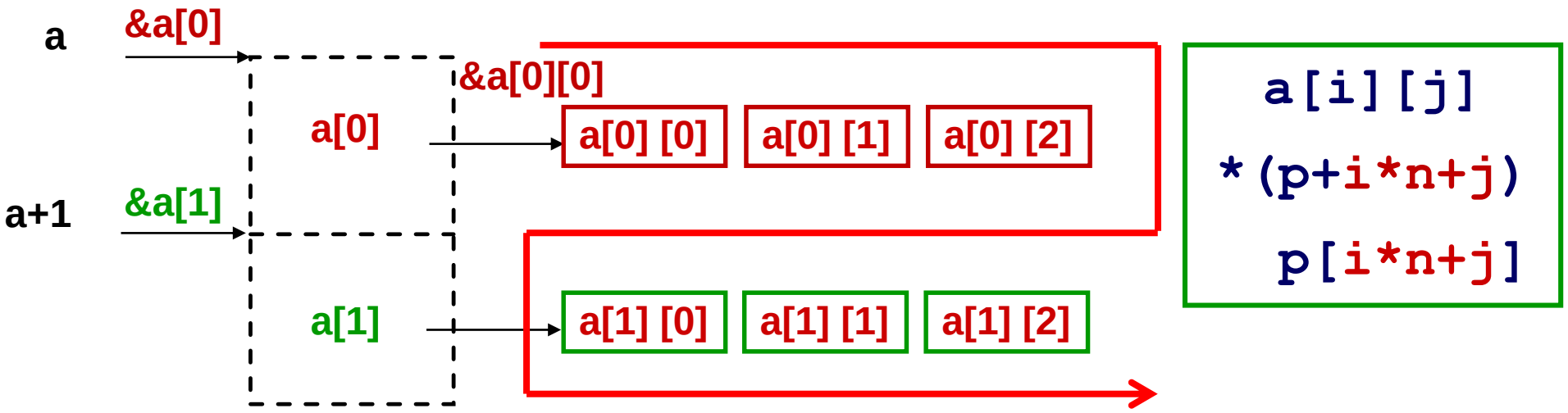
- 若要让一个指针指向它，则应定义为
- `int *p;` //**列指针**，基类型是`int`
- `p = a[0];`
- `p = &a[0][0];` //指向**第0行第0列**的`int`型元素

8.1 指针与二维数组

- 逐列查找
- 根据相对偏移量

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

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



8.1 指针与二维数组

【例】输入一个3行4列的二维数组，然后输出这个二维数组的元素值

```
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",  ? ? ?  ;
        }
    }
}
```

形参声明为二维数组，列数须为常量

```
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",  ? ? ?  );
        }
        printf("\n");
    }
}
```

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

8.1 指针与二维数组

【例】输入一个3行4列的二维数组，然后输出这个二维数组的元素值

```
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]);
        }
    }
}
```

形参声明为二维数组，列数须为常量

```
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");
    }
}
```

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

8.1 指针与二维数组

【例】输入一个3行4列的二维数组，然后输出这个二维数组的元素值

```
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", ? ? ? );
        }
    }
}
```

形参声明为二维数组的**行指针**，列数须为常量

```
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", ? ? ? );
        }
        printf("\n");
    }
}
```

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

8.1 指针与二维数组

【例】输入一个3行4列的二维数组，然后输出这个二维数组的元素值

```
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);
        }
    }
}
```

形参声明为二维数组的**行指针**，列数须为常量

```
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");
    }
}
```

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

8.1 指针与二维数组

【例】输入一个3行4列的二维数组，然后输出这个二维数组的元素值

```
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", ? ? ? );
        }
    }
}
```

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", ? ? ? );
        }
        printf("\n");
    }
}
```

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

8.1 指针与二维数组

【例】输入一个3行4列的二维数组，然后输出这个二维数组的元素值

```
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]);
        }
    }
}
```

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");
    }
}
```

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

第八讲 指针——学习内容

8.1 指针与二维数组

8.2 指针数组与指向数组的指针区别

8.3 函数与指针

8.2 指针数组和指向数组的指针区别

- 任何类型都可以作为指针和数组的基类型（决定字节个数）
  - 当数组作为指针的基类型时，这个指针就叫做指向数组的指针；

    例如 `int(*p)[4];` `//P是指向一维数组的指针变量`

    语法理解逻辑：（*p）意味着这是指针变量 -> 类型为int [4]
  - 当指针作为数组的基类型时，这个数组就叫做指针数组；

    数组元素均为指针类型数据，即指针数组中的每一个元素都存放地址，相当于一个指针变量

    定义一维指针数组的一般形式为：

    类型名 * 数组名[数组长度];

    下面定义一个指针数组： `int* p[4];`

    由于 [ ] 比 * 优先级高，因此 p 先与 [4] 结合，形成 p[4] 形式，这显然是数组形式，表示 p 数组有 4 个元素。然后再与 p 前面的 “ * ” 结合，“ * ” 表示此数组是指针类型的，每个数组元素（相当于一个指针变量）都可指向一个整型变量。

第八讲 指针——学习内容

8.1 指针与二维数组

8.2 指针数组与指向数组的指针区别

8.3 函数与指针

8.3 指针与函数-函数指针及其应用

- 函数指针(Function Pointer)就是指向函数的指针变量

数据类型 （*指针变量名）(形参列表)；

例：

```
int  (*f)(int a, int b);
```

- 函数指针**f**指向的函数原型为：

```
int 函数名(int a, int b);
```

- 令**f = Fun**，就是让**f**指向函数**fun()**
- 编译器将不带()的函数名解释为该函数的入口地址
- 函数指针变量存储的是函数在内存中的入口地址

8.3 指针与函数-函数指针及其应用


```
int (*f)(int a, int b);
```

- 常见错误：
  - 忘了写前一个()——`int *f(int a, int b);`
    - 声明了一个函数名为f、返回值是整型指针类型的函数
  - 忘了写后一个()——`int (*f);`
    - 定义了一个整型指针变量
  - 定义时的参数类型与指向的函数参数类型不匹配
    `int (*f)(float a, float b);`
  - 不建议写成——`int (*f)();`

8.3 指针与函数-函数指针及其应用

例:用函数指针变量作函数参数，求最大值、最小值和两数之和

```
void Fun(int x, int y, int (*f)(int, int));
int main()
{    int a, b;
     scanf("%d,%d", &a, &b);
     Fun(a, b, Max);
     Fun(a, b, Min);
     Fun(a, b, Add);
     return 0;
}
void Fun(int x, int y, int (*f)(int, int))
{    int result;
     result = (*f)(x, y) ;
     printf("%d\n", result);
}
```

```
int Max(int x, int y);
int Min(int x, int y);
int Add(int x, int y);
int Max(int x, int y)
{    printf("max=");
     return  x>y? x : y;
}
int Min(int x, int y)
{    printf("min=");
     return  x<y? x : y;
}
int Add(int x, int y)
{    printf("sum=");
     return  x+y;
}
```

本讲小结

- 指针与二维数组
- 指针与结构体
- 指针数组与指向数组的指针区别
- 函数与指针
  - 按值调用与按地址调用
  - 函数指针