第 12 章

指针和数组

背景简介

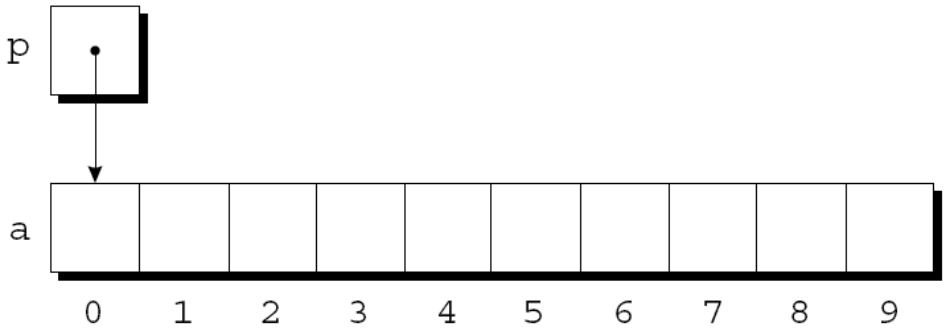
- C语言中指针和数组关系非常紧密。
- 指针可代替下标对数组进行处理，
  - eg，遍历数组元素。
- C允许数组元素指针进行算术运算：
  - 下标可运算，指针亦可运算
  - 加法和减法。

指针的算术运算

- 指针可指向数组元素：

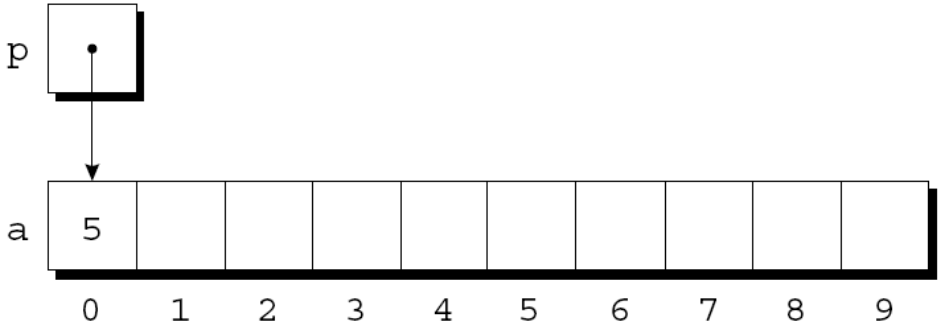
```
int a[10], *p;
p = &a[0];//p = a
```

- 用图形方式表示为：



指针的算术运算

- 通过p访问a[0]：
  - 读：printf(“%d”, *p);
  - 写：*p = 5;
- 更新后的图形表示为：



指针的算术运算

- 指针变量，与其他变量一样：
  - 变量：有存储空间，4Byte，数据可变
  - 数据：地址
- 对指针进行算术运算（或地址算术运算）可访问其他元素：
- 仅有三种指针算术运算
  - 指针加上整数
  - 指针减去整数

point move

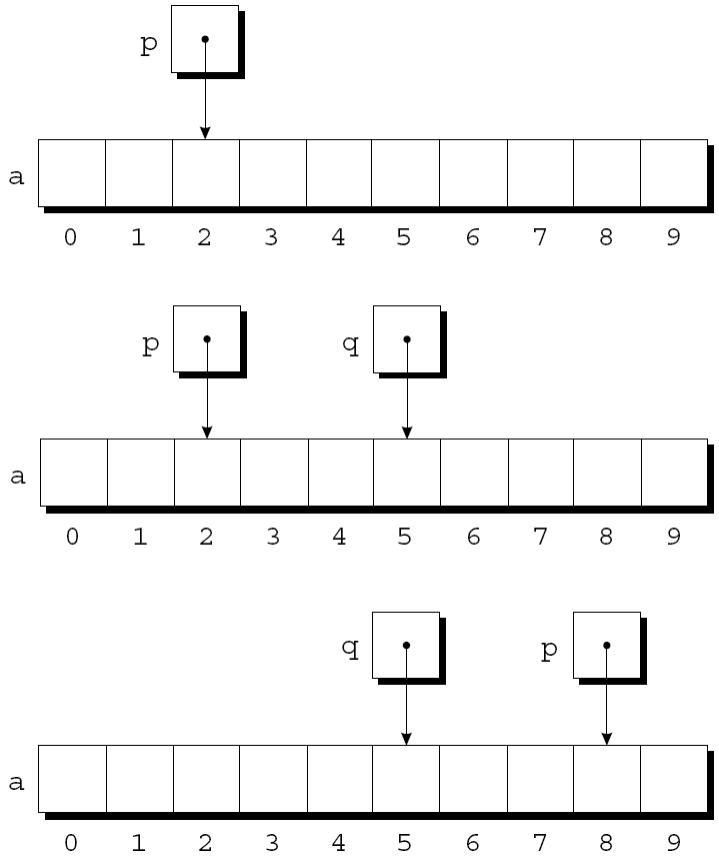
算术运算的前提：指向数组

指针加上整数

- **p+j：**
  - **p向前移动，指向p当前所指元素之后的j个元素。**
- 如，
  - **p指向a[i]，p+j指向a[i+j]。**

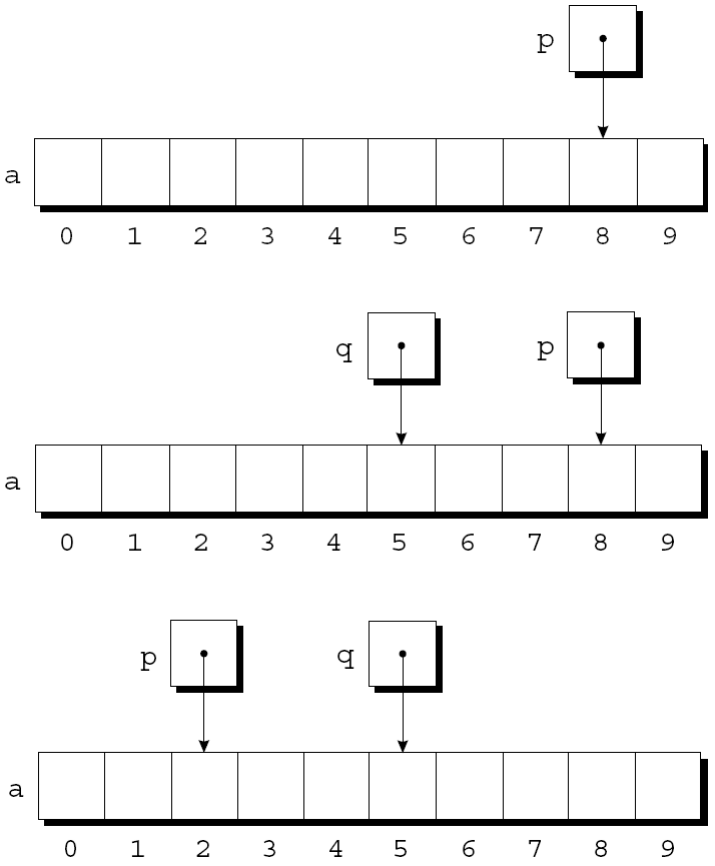
指针加上整数

- 声明：`int a[10], *p, *q, i;`
- 指针加法运算示例：
  - `p = &a[2];`
  - `q = p + 3;`
  - `  = &a[2+3]`
  - `p += 6;`



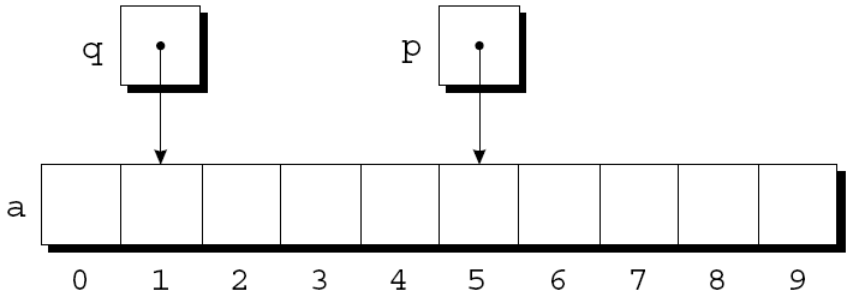
指针减去整数

- p指向a[i]，
- p - j：向后移动，指向a[i-j]。
- 示例：
  - p = &a[8];
  -
  - q = p - 3;
  -
  - p -= 6;



两个指针相减

- 结果：指针之间距离（间隔数组元素个数）。
- p指向a[i]，q指向a[j]，p - q 等于 i - j
- 示例：
  - p = &a[5];
  - q = &a[1];




```
i = p - q;    /* i is 4 */
i = q - p;    /* i is -4 */
```

两个指针相减

- 下列操作会导致未定义的行为：
  - 对并未指向数组元素的指针执行算术运算
  - 两个指针并非指向同一数组中元素时，对其执行减法操作

指针比较

- 指针可以采用关系运算符 (<, <=, >, >=) 和判等运算符 (== and !=)进行比较：
  - 当两指针指向同一数组，比较才有意义。
- 比较：
  - 指针指向元素在数组中的相对位置。
- 例如：通过如下的赋值操作

```
p = &a[5];
q = &a[1];
```

此时，p >= q。

数组访问方式总结

- int a[N];
- 下标：0~N-1
  - i：第i个元素
  - i++：下一个元素
- 指针：int *p=&a[0] // p = a;
  - p+i;指向a[i]
  - p++：下一个元素
- 下标i：
  - a[i]地址a+i*sizeof(int)
- 指针p：
  - p++：p+sizeof(int)
  - p+i：p+i*sizeof(int)

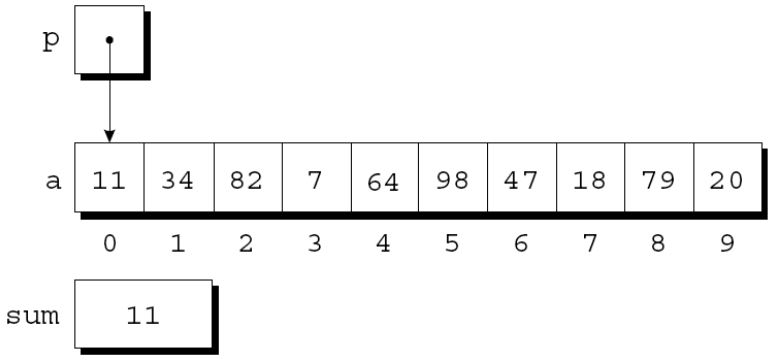
指针用于数组处理

- 指针算术运算：循环自增指针变量逐个访问数组元素。
- 数组求和：

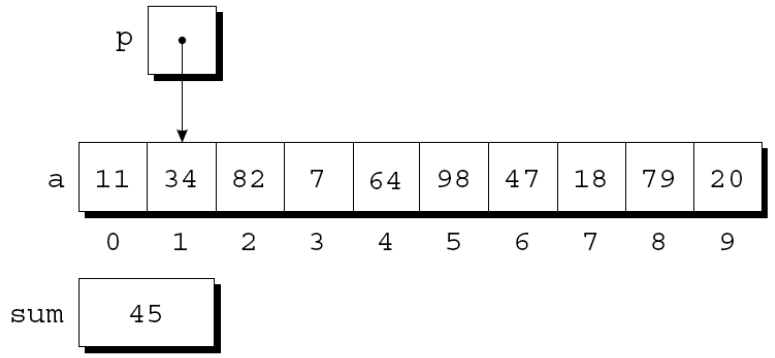
```
#define N 10
…
int a[N], sum=0, *p;
…
for (i = 0; i<N; i++) sum+=a[i];

for (p = &a[0]; p < &a[N]; p++)
    sum += *p;
```

指针用于数组处理

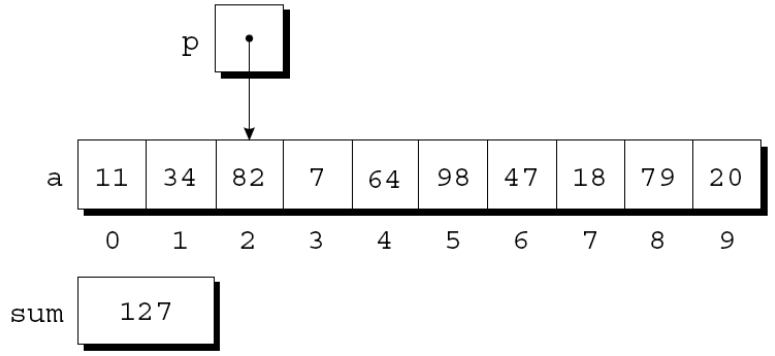
- 第一轮迭代结束时：



- 第二轮迭代结束时：



- 第三轮迭代结束时：



指针用于数组处理

- 循环条件 `p < &a[N]`
- 下标N越界，`a[N]`不存在，但仅取址，所以合法。
  - `&a[N] = a + N[*sizeof(type)];`
- 指针算术运算通常可以节省执行时间。
  - 直接寻址
  - 依赖具体实现，某些编译器，数组下标来遍历数组效率可能更高。

* 和 ++ 运算符的组合

- 常用于数组元素处理。
- 下标修改数组元素，并移动下标到下一个元素：

```
a[i++] = j;
```

- 指针可实现同样功能：

```
*p++ = j;（p当前已指向a[i]）
```

- 后缀 ++ 优先级高于*，等价于

```
*(p++) = j;
```

取p，访问*p，后p++

* 和 ++ 运算符的组合

- 可能的 * 和 ++组合：
  - *p++: 先取*p，然后自增p；
  - (*p)++: 先取*p，然后自增*p；
  - *++p: 先自增 p，后取*p；
  - ++*p: 右结合，自增*p；

* 和 ++ 运算符的组合

- 最常见组合*p++，在循环中十分方便
  - 先取当前元素，再后移指针，与a[i++]同
- 例如：对数组a求和

```
for (p = &a[0]; p < &a[N]; )
    sum += *p++;
```

- 可以改写为：

```
p = &a[0];
while (p < &a[N])
    sum += *p++;
```

* 和 ++ 运算符的组合

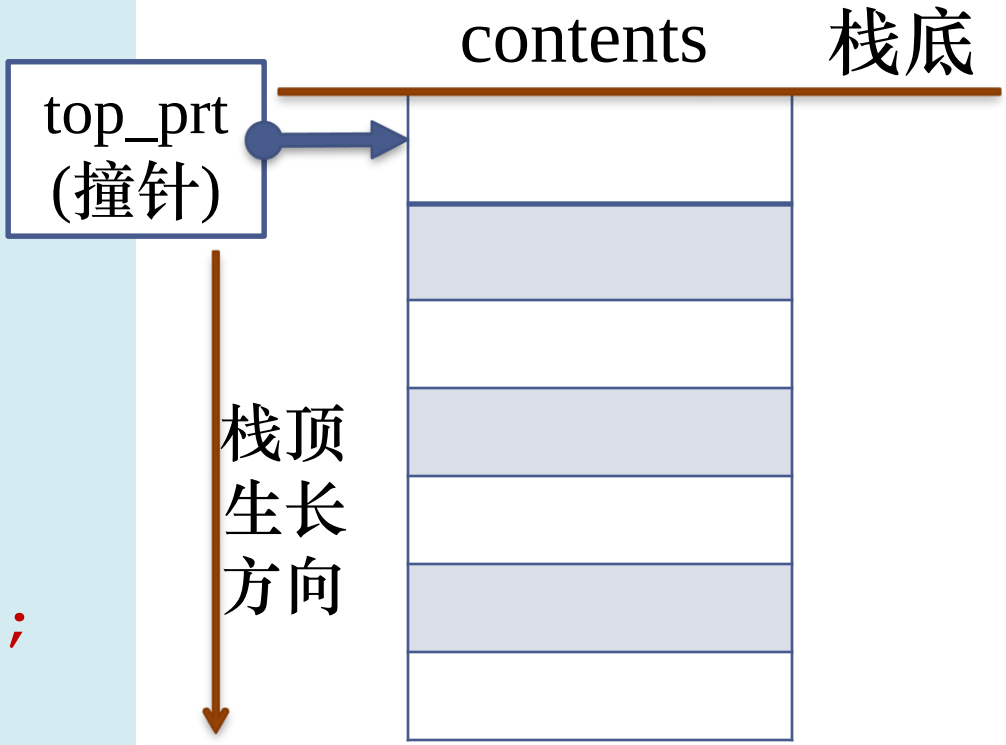
- * 和 --组合类似于* 和 ++的组合
- “栈”例：
  - 整型变量top跟踪contents数组“栈顶”
- 指针变量top_ptr替换top，初始指向contents数组的第0个元素：

```
int *top_ptr = &contents[0];
             = contents;
```

* 和 ++ 运算符的组合

- 新的 push 和 pop 函数：

```
void push(int i)
{
  if (is_full())
    stack_overflow();
  else
    *top_ptr++ = i;
}
int pop(void)
{
  if (is_empty())
    stack_underflow();
  else
    return *--top_ptr;
}
```



用数组名作为指针

- 数组与指针关联关系：
  - 指针操作数组
  - 数组名本质上是数组（首元素）地址
  - 数组名可当做指针使用
  - 指针算术运算

用数组名作为指针

- 声明数组：

```
int a[10];
```

- 数组名是地址，可以指针方式来使用

```
*a = 7;    /* a等价于&a[0]*/
*(a+1) = 12;    /* a[1]=12 */
a + i 和 &a[i]等同
*(a+i) 与 a[i]也等同
```

用数组名作为指针

- 数组名作指针，编写遍历数组循环更加容易：

```
for (p =&a[0]; p < &a[N]; p++)
   sum += *p;
```

- 用数组名简化：

```
for (p = a; p < a + N; p++)
   sum += *p;
```

用数组名作为指针

- 数组名可作指针，但不是指针变量，而是常指针。
  - 数组内存空间不可移动
  - eg,单位或学校地址不可改动，而快递单可填任何地址
- 不能对数组名（地址）进行修改：

```
a++;                    /*** WRONG ***/
a = p;                  //修改数组位置
```

- 通常用其它指针变量操作数组，而不是把数组名当指针变量来操作：

```
p = a;
while (*p != 0)
  p++;
```

程序：数列反向（指针版）

- 第八章**reverse**程序，读入10个数，后逆序输出 。
- 读入10数字存储数组，利用指针访问数组元素。

```c
#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], *p;
  printf("Enter %d numbers: ", N);
  for (p = a; p < a + N; p++)
    scanf("%d", p);

  printf("In reverse order:");
  for (p = a + N - 1; p >= a; p--)
    printf(" %d", *p);

  printf("\n");
  return 0;
}
```

数组型参数 (改进版)

- 数组名传递给函数，总是被视为指针：

```
int find_largest(int a[], int n){
  int i, max;
  max = a[0];
  for (i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;}
```

- 调用：

```
largest = find_largest(b, N);
```

- 数组b地址（指针）赋给a
  - 形参数组a可作数组名，也可作指针

数组型参数影响与后果

- 将数组作为参数，会产生一些重要的影响和后果。
- 1、函数可读/写数组
  - 普通变量作为参数，形参是实参副本，函数修改形参不影响实参。
  - 数组名作参数（原件地址），函数可以修改数组元素。
- 2、传参效率高
  - 只传递数组地址，不复制整个数组。
  - 向函数传递数组参数所需时间与数组大小无关。

温故而知新——指针与数组

- 指针——处理数组的第二种方式
  - 声明：`int a[N], *p;`
  - 指向：`p = &a[0];p = a;`
  - 访问元素：`*p = 5;`
  - 移动（指针算术运算）
    - 前移：`p+i; p++;`
    - 后移：`p-j; p--;`
    - 指针间距：`p-q;`
  - 循环处理数组（指针作为循环控制变量）
    - `for(p=&a[0]; p<&a[N]; p++) sum+=*p;`
    - `for(p=a; p<a+N; p++) sum+=*p;//`数组名做指针
- 数组名做指针（常指针）
  - `*(a+i)=5;`
- 数组做函数参数，传地址（视作指针）

数组型参数影响与后果

- 3、可声明为指针
  - 数组名本质是地址，可将数组形参声明为指针：

```
int find_largest(int *a, int n)
```

等价于

```
int find_largest(int a[], int n)
```

- 作为形参，数组与指针等价；作为变量则不同
- eg. `int a[10],*p;`
  - 编译器为a预留10个整型变量的内存空间（40Byte）
  - 为p预留一个指针变量的空间（4Byte）

数组型参数影响与后果

- 4、可处理数组片段
  - 可传递数组“片段”：函数不处理完整数组。
- 例如：

```
int find_largest(int a[], int n)
```

查找数组b第5到第14号元素中的最大者

```
largest = find_largest(&b[5], 10);
```

使用指针作为数组名

- 前提：指针指向数组，此时指针是数组别名
  - 可将指针当作数组名，执行下标操作：

```
#define N 10
int a[N], i, sum = 0, *p = a;
//p=a，指向相同内存空间，p为a别名
for (i = 0; i < N; i++)
   sum += p[i];
```

- 编译器会将p[i]和*(p+i)等同进行处理

数组型参数总结(四种函数定义)

- 调用：

```
int b[N];
largest =
find_largest(b, N);
```

- 声明为数组，下标处理

```
int find_largest(int
a[], int n){
  int i, max;
  max = a[0];
  for (i = 1; i < n;
i++)
    if (a[i] > max)
      max = a[i];
  return max;}
```

- 调用：

```
int b[N];
largest =
find_largest(b, N);
```

- 声明为指针，指针处理

```
int find_largest(int
*a, int n){
  int *p;
  max = *a;
  for (p = a; p <
a+n; p++)
    if (*p > max)
      max = *p;
  return max;}
```

数组型参数(四种函数定义)

- 调用：

```
largest =
find_largest(b, N);
```

- 声明为指针，下标处理

```
int find_largest(int
*a, int n){
  int i, max;
  max = a[0];
  return max;}
```

- 调用：

```
largest =
find_largest(b, N);
```

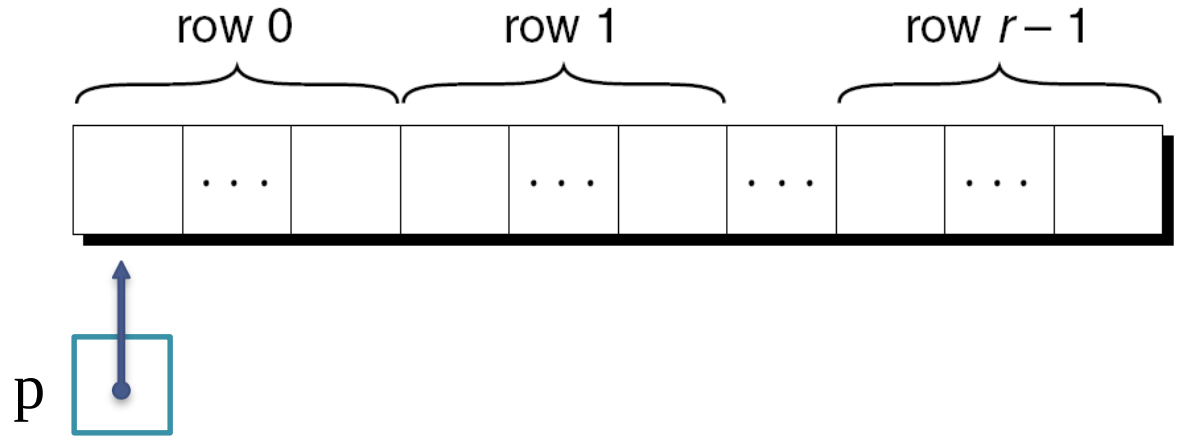
- 声明为数组，指针处理

```
int find_largest(int
a[], int n){
  int *p=a;
  max = *p;
  return max;}
```

作为形参：
指针等同于数组，
编译器将这两种声明方式看做是完全一样的。

指针处理多维数组的元素

- 指针也可指向多维数组的元素。
- C按行主序存储二维数组
  - r行数组：



- 指针p指向二维数组的第一个元素（0行0列），反复自增p访问每一个元素。

处理多维数组的元素

- 示例：二维数组清零：

```
int a[RS][CS];
```

- 常规方法：嵌套for循环：

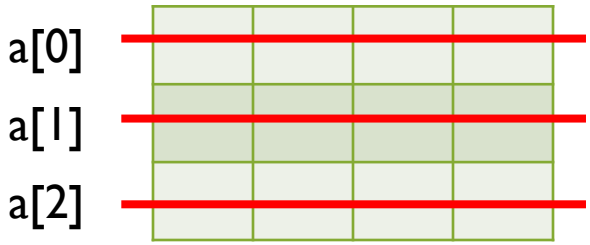
```
int row, col;
…
for (row = 0; row < RS; row++)
  for (col = 0; col < CS; col++)
    a[row][col] = 0;
```

- 利用指针，将a视作一维整型数组，一重循环就够了

```
int *p;
…
for (p = &a[0][0]; p <= &a[RS-1][CS-1];
p++)
  *p = 0
```

二维作一维“取巧”
破坏程序可读性

处理多维数组的行



- 数组嵌套（降维）：
  - N维数组：一维数组，元素为N-1维数组
  - type array[L1]
- 二维数组视作一维数组，元素为行（一维数组）：

```
int a[3][4]={a[0],a[1],a[2]}
```

  - a[i]：一维数组a[3][4]的第i个元素
  - 实质：二维数组a的第i行，一维数组
  - 数组名a[i]：指向数组a中第i行的首元素。等价于&a[i][0]

处理多维数组的行

- 指针也可处理（访问）二维数组中某一行元素，初始化：
  - `p = &a[i][0];`简化：`p = a[i];`
- 数组a第i行元素清零：

```
int a[NUM_ROWS][NUM_COLS], *p, i;
for (p = a[i]; p < a[i] + NUM_COLS; p++)
  *p = 0;
```

处理多维数组的行

- 二维数组的行可作为参数传入以一维数组为形参的函数。

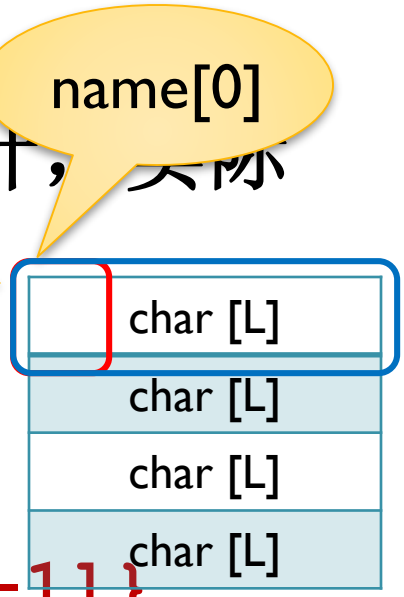
```
int find_largest(int a[], int n)
```

  - 可查找二维数组a中第i行的最大元素：

```
largest = find_largest(a[i], NUM_COLS);
```

用多维数组名作为指针

- 无论维数高低，数组名均可作指针，实际应用需小心。
- C将N维数组视为嵌套数组：Name[0]
  - 元素为N-1维数组的一维数组
  - char name[M][L];//名单
  - ={name[0],name[1],……name[M-1]}
- name：一维数组首元素地址
  - name=&name[0]
- name[0]：二维首元素地址
  - name[0]=&name[0][0]
- name与name[0]
  - 起始位置相同，所辖空间大小不同
  - eg.连长、1排长有职权大小之分

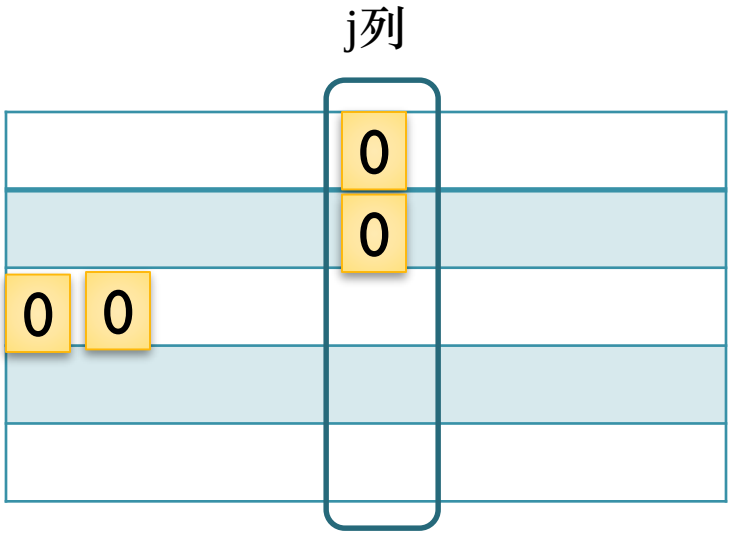


用多维数组名作为指针

- char name[M][L];
- Name：指向一行的指针
  - 类型：char (*)[L]的指针
  - 指向长度L的char数组
- 对比
  - 元素指针：
    - char *p=name[0]或&name[0][0]
    - p++：移动一个元素
  - 行指针：
    - char (*p)[L]=name或&name[0];
    - p++：移动一行（实质也是元素指针，元素为行）

处理多维数组的列

- 稍麻烦些，数组逐行存储（非逐列）。
  - 处理行：指针每次移动一个元素
  - 处理列：每次移动一行
  - 第0个元素，p→a[0][j]
  - 第1个元素，p→a[1][j]
  - 第2个元素，p→a[2][j]

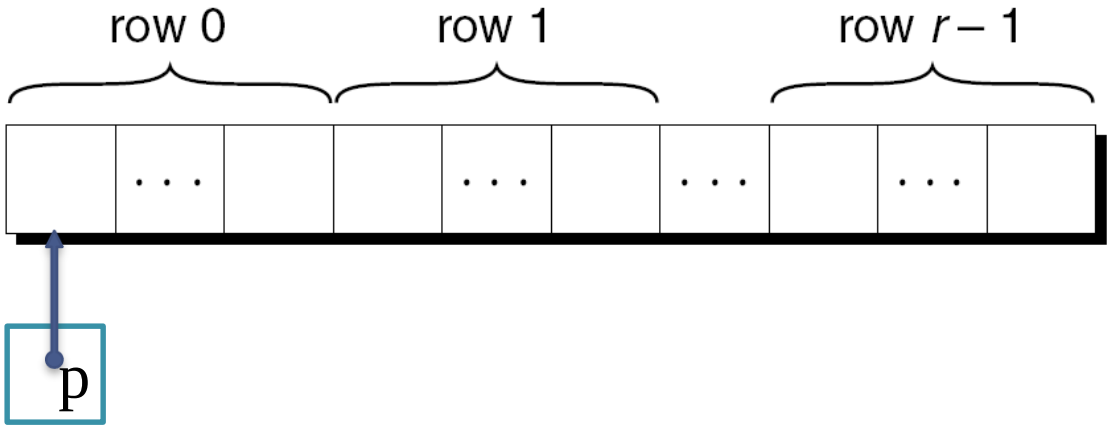


使用数组行指针

处理多维数组的列

- 二维数组a中第i列元素清零：

```
int a[NUM_ROWS][NUM_COLS],
(*p)[NUM_COLS], i;

…

for(p= a; p < a+NUM_ROWS; p++)
   (*p)[i] = 0;
```



用多维数组名作为指针

- 查找数组a[NUM_ROWS][NUM_COLS]最大元素：

```
int find_largest(int a[], int n){
  int i, max = a[0];
  for (i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max; }
```

- 调用：把二维数组看着NUM_ROWS * NUM_COLS个元素的一维数组

```
largest = find_largest(a, NUM_ROWS *
NUM_COLS);  /* WRONG */
```

- 实参a类型：
  - int(*)[NUM_COLS]
  - 而find_largest希望接受是int *

用多维数组名作为指针

- 正确调用：

```
largest = find_largest(a[0],
NUM_ROWS * NUM_COLS);
```

- a[0]指向a[0][0](类型int *)
- 结论：以数组为形参的函数，本质上以类型相同的指针来处理

本章作业

- 练习题自行练习；
- 编程题2、4、5、7。