



规格严格 功夫到家



第11章 指针和数组



哈尔滨工业大学

计算机科学与技术学院

苏小红

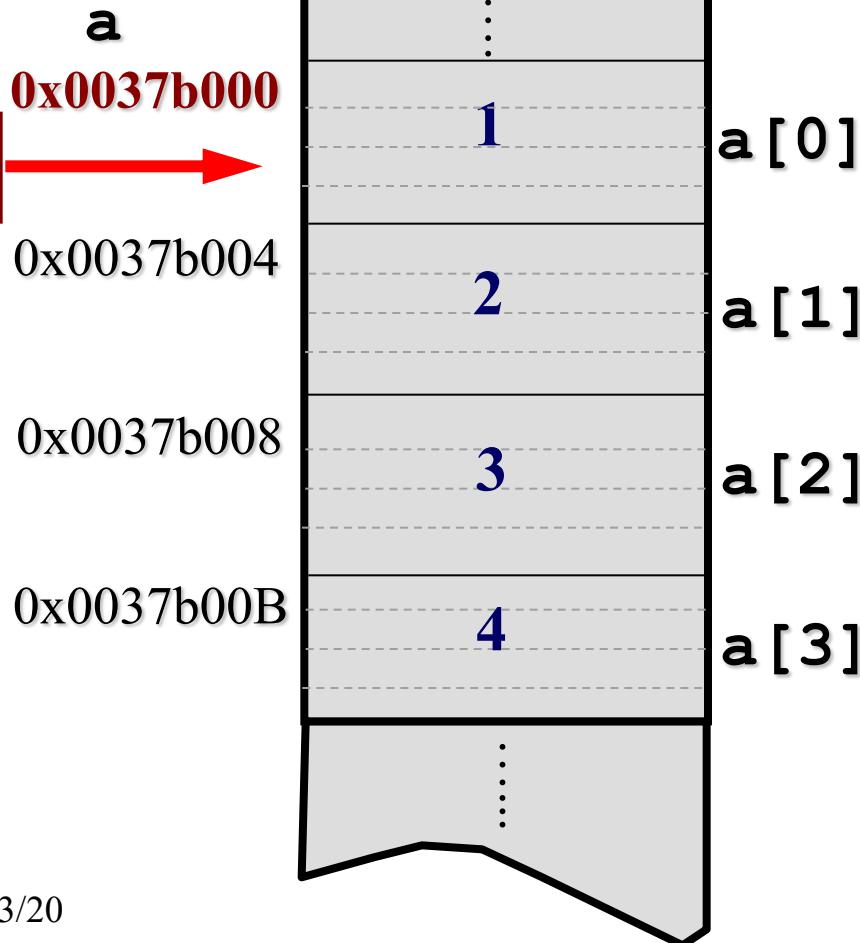
sxh@hit.edu.cn

本章学习内容

- ☞ 指针与一维数组间的关系，指针与二维数组间的关系
- ☞ 向函数传递一维数组和二维数组
- ☞ 指针数组，命令行参数
- ☞ 动态数组，动态内存分配

11.1 指针和一维数组间的关系

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



```
int *pa=a;
```

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

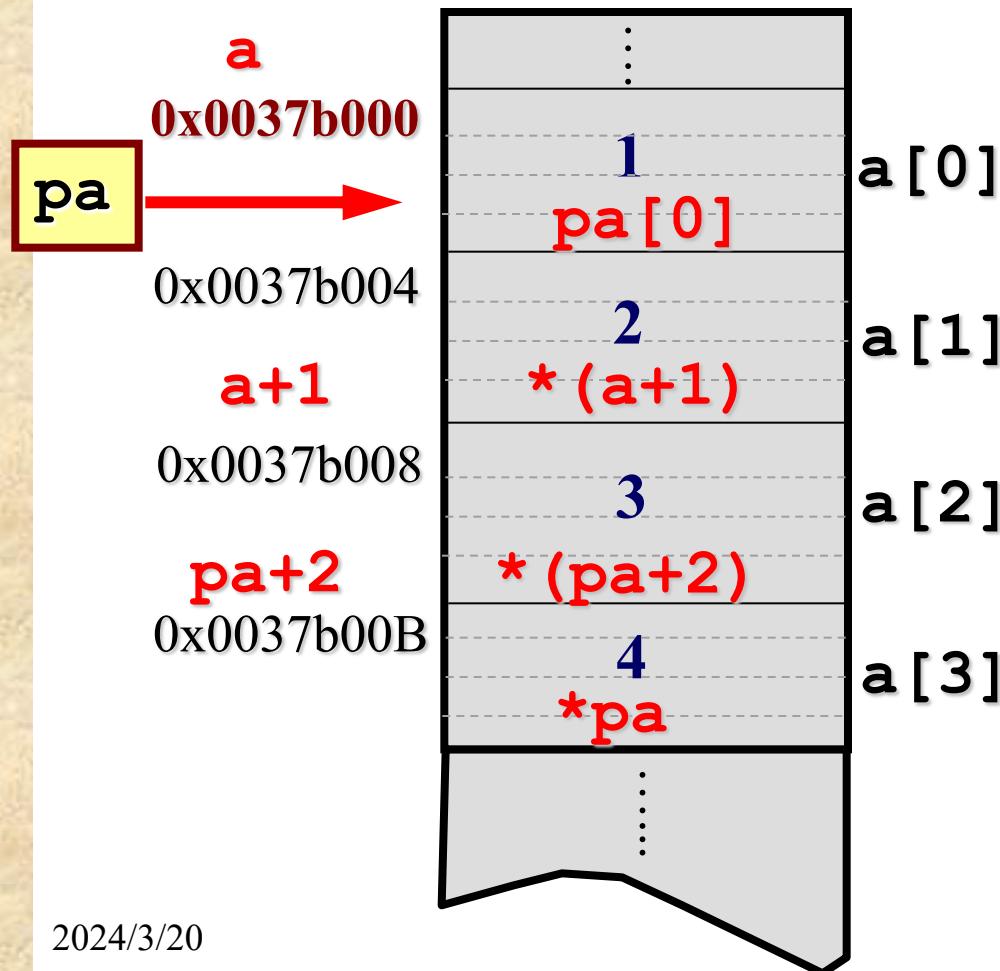
数组名是一个常量指针
不能修改该指针的指向

指针可当数组名使用

11.1 指针和一维数组间的关系

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

```
int *pa=a;
```



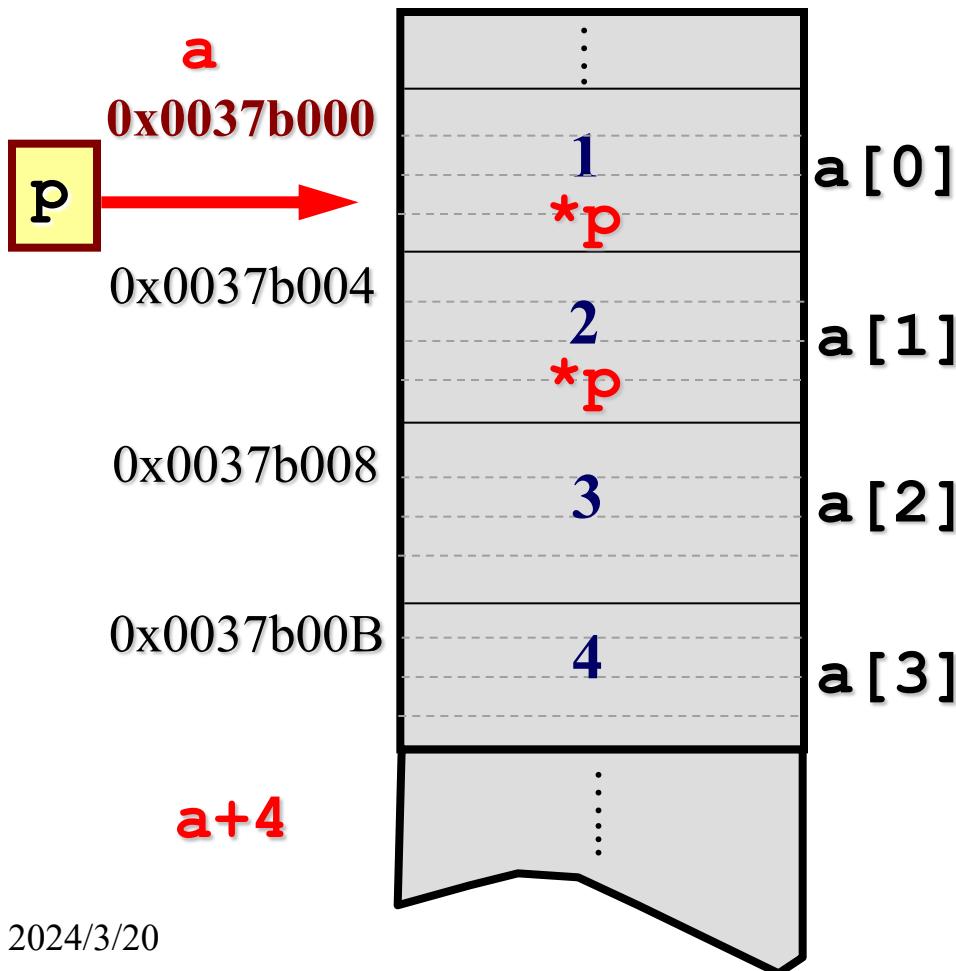
```
int *pa=&a[0];
```

数组元素的等价引用形式

$a[i]$
 $*(a+i)$
 $pa[i]$
 $*(pa+i)$

11.1 指针和一维数组间的关系

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



```
for (i=0; i<4; i++)  
    scanf("%d", &a[i]);
```

```
for (i=0; i<4; i++)  
    printf("%d ", a[i]);
```

```
for (p=a; p<(a+4); p++)  
    scanf("%d", p);
```

~~for (p=a; p<(a+4); p++)
 printf("%d ", *p);~~

11.1 指针和一维数组间的关系

【例11.1】演示数组元素的引用方法

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5], i;
6     printf("Input five numbers:");
7     for (i=0; i<5; i++)
8     {
9         scanf("%d", &a[i]);
10    }
11    for (i=0; i<5; i++)
12    {
13        printf("%4d", a[i]);
14    }
15    printf("\n");
16    return 0;
}
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5], i;
6     printf("Input five numbers:");
7     for (i=0; i<5; i++)
8     {
9         scanf("%d", a+i);
10    }
11    for (i=0; i<5; i++)
12    {
13        printf("%4d", *(a+i));
14    }
15    printf("\n");
16    return 0;
}
```

11.1 指针和一维数组间的关系

【例11.1】演示数组元素的引用方法

```
1 #include <stdio.h>
2 int main()
3 {
4     int a[5], *p;
5     printf("Input five numbers:");
6     for (p = a; p<a+5; p++)
7     {
8         scanf("%d", p);
9     }
10    for (p = a; p<a+5; p++)
11    {
12        printf("%4d", *p);
13    }
14    printf("\n");
15    return 0;
16 }
```

```
1 #include <stdio.h>
2 int main()
3 {
4     int a[5], *p = NULL, i;
5     printf("Input five numbers:");
6     p = a;
7     for (i=0; i<5; i++)
8     {
9         scanf("%d", &p[i]);
10    }
11    p = a;
12    for (i=0; i<5; i++)
13    {
14        printf("%4d", p[i]);
15    }
16    printf("\n");
17    return 0;
18 }
```

11.1 指针和一维数组间的关系

【例11.2】演示数组和指针变量作函数参数

```
3 void InputArray(int a[], int n)
4 {
5     int i;
6     for (i=0; i<n; i++)
7     {
8         scanf("%d", &a[i]);
9     }
10 }
```

被调函数的形参声明为
数组类型，用下标法访
问数组元素

```
11 void OutputArray(int a[], int n)
12 {
13     int i;
14     for (i=0; i<n; i++)
15     {
16         printf("%4d", a[i]);
17     }
18     printf("\n");
19 }
```

11.1 指针和一维数组间的关系

【例11.2】演示数组和指针变量作函数参数

```
3 void InputArray(int *pa, int n)
4 {
5     int i;
6     for (i=0; i<n; i++, pa++)
7     {
8         scanf("%d", pa);
9     }
10 }
```

被调函数的形参声明为
指针类型，用指针法访
问数组元素

```
11 void OutputArray(int *pa, int n)
12 {
13     int i;
14     for (i=0; i<n; i++, pa++)
15     {
16         printf("%4d", *pa);
17     }
18     printf("\n");
19 }
```

11.1 指针和一维数组间的关系

【例11.2】演示数组和指针变量作函数参数

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5];
6     printf("Input five numbers:");
7     InputArray(a, 5);
8     OutputArray(a, 5);
9     return 0;
}
```

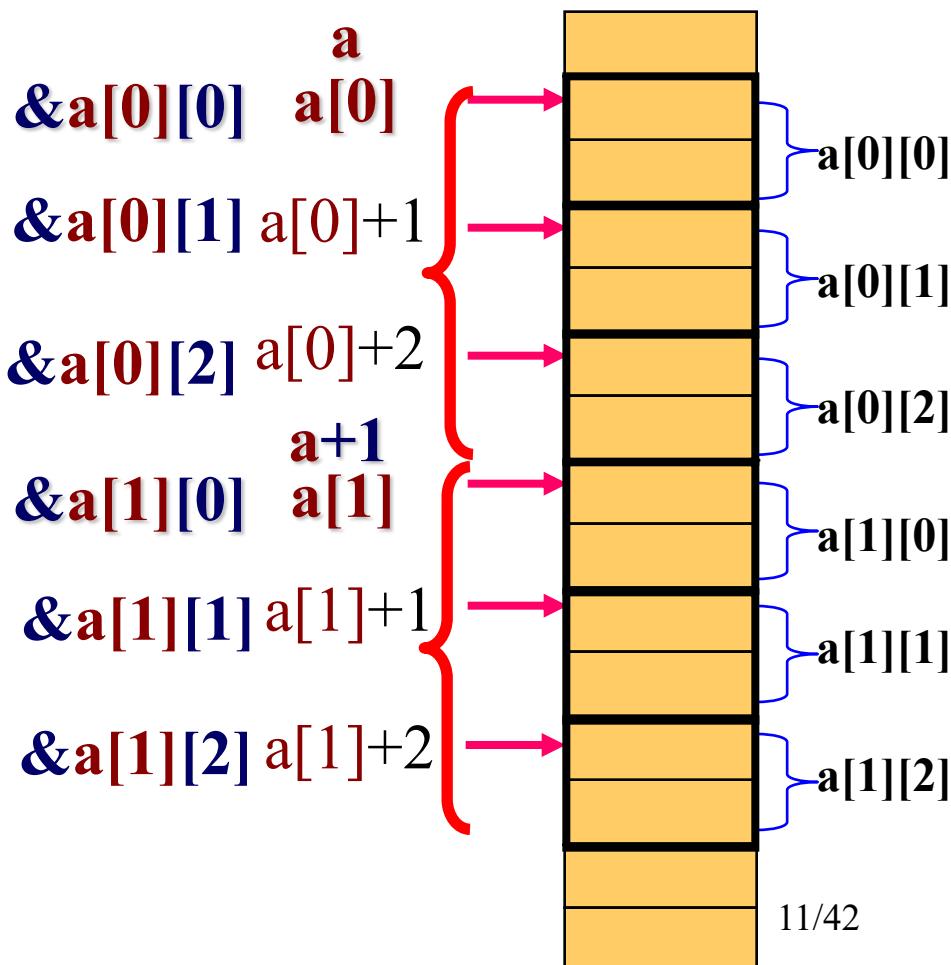
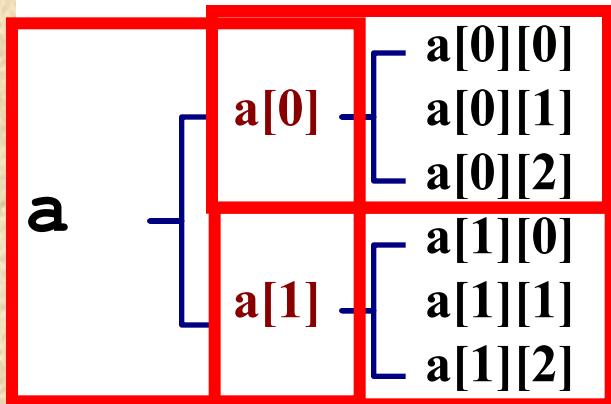
在主函数中这样做没有多大的实际意义

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5];
6     int *p = a;
7     printf("Input five numbers:");
8     InputArray(p, 5);
9     OutputArray(p, 5);
10    return 0;
}
```

11.2 指针和二维数组间的关系

- 可将二维数组看作一维数组，其每个数组元素又是一个一维数组
- 按行顺序存放所有元素 $\&a[0][0]$ $a[0]$

```
short a[2][3];
```

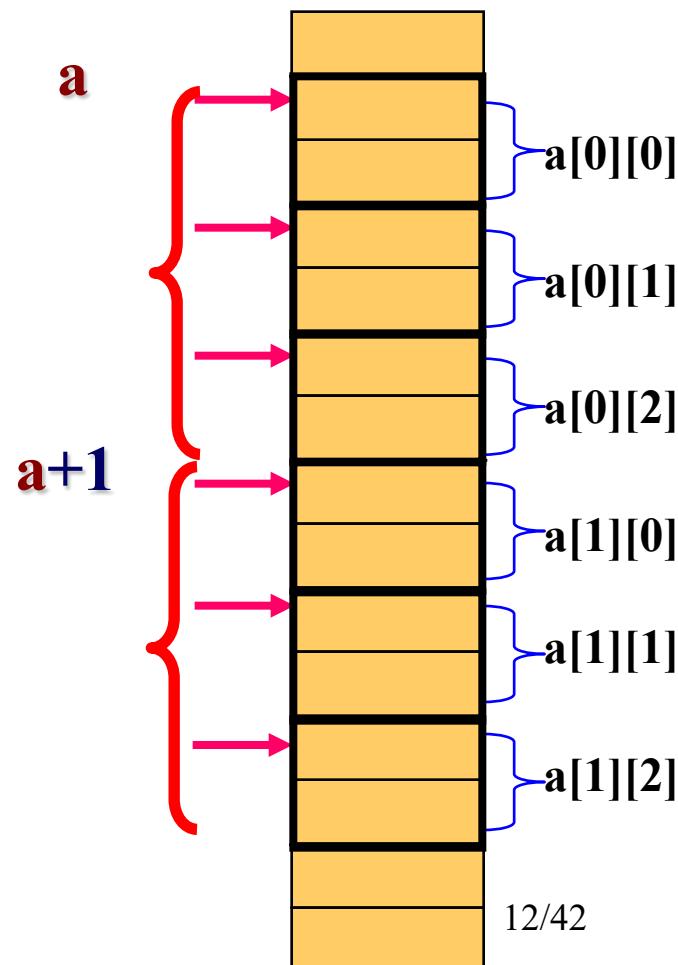
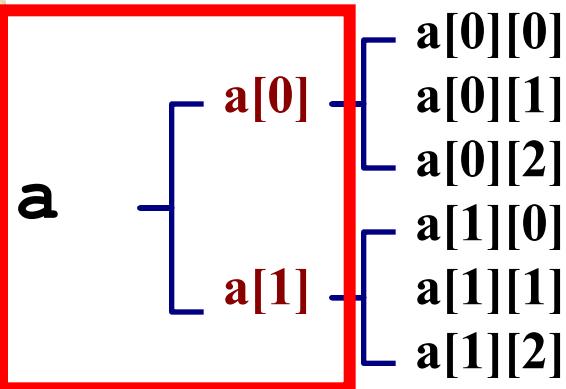


11.2 指针和二维数组间的关系

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！

```
short a[2][3];
```

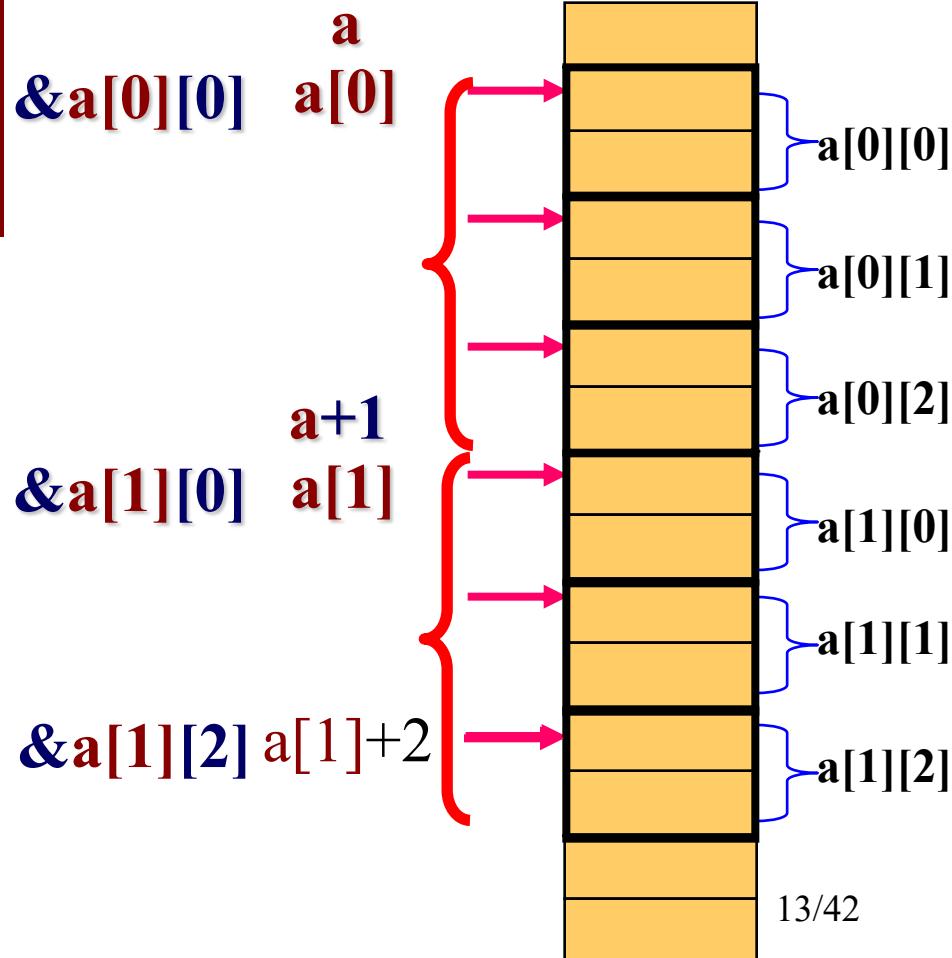
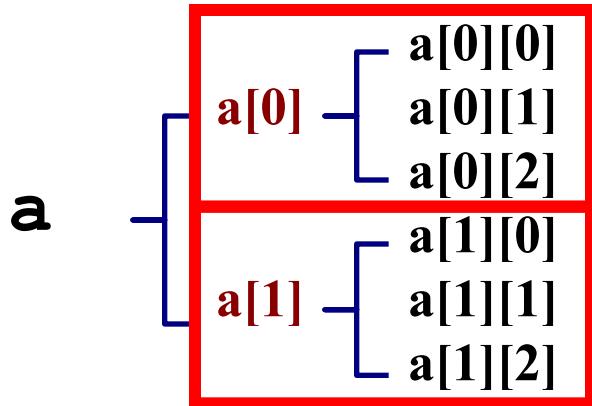


11.2 指针和二维数组间的关系

$*(a + i)$ 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 $\&a[i][j]$

```
short a[2][3];
```

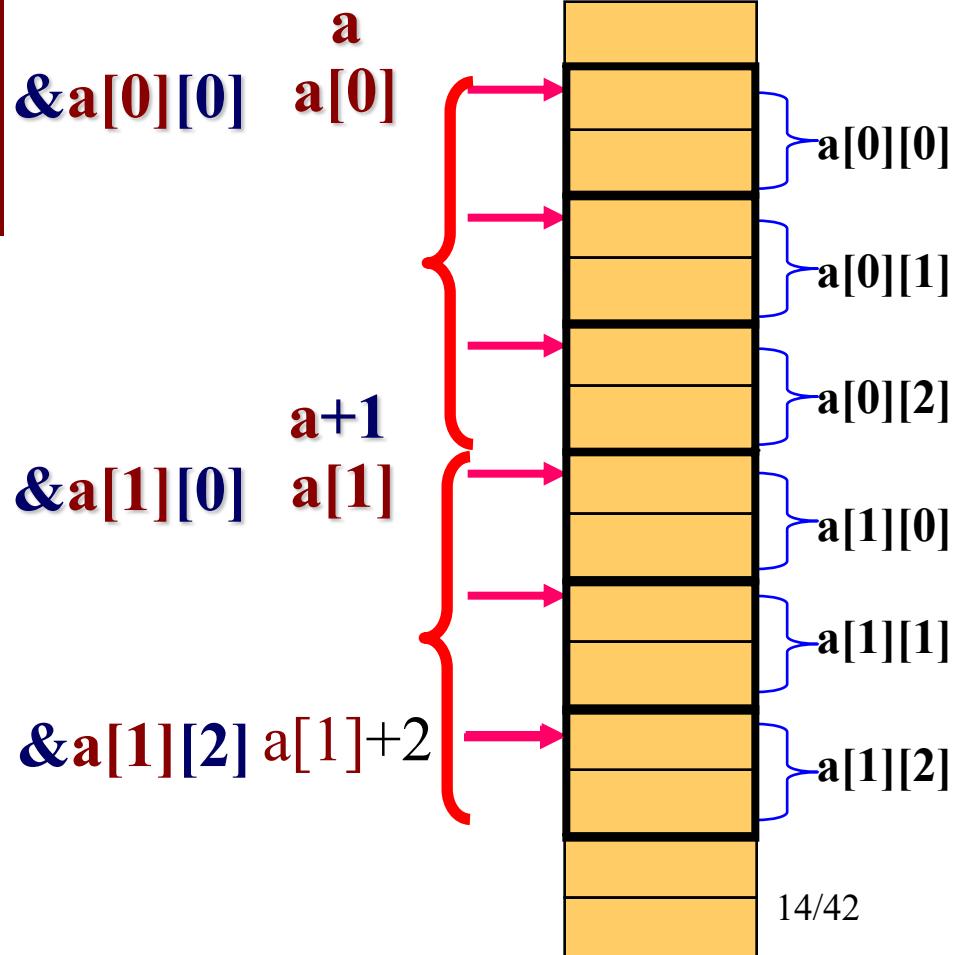
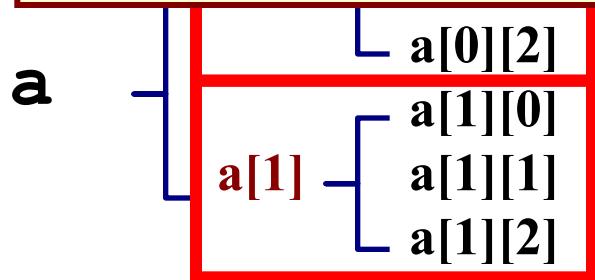


11.2 指针和二维数组间的关系

$*(a + i)$ 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



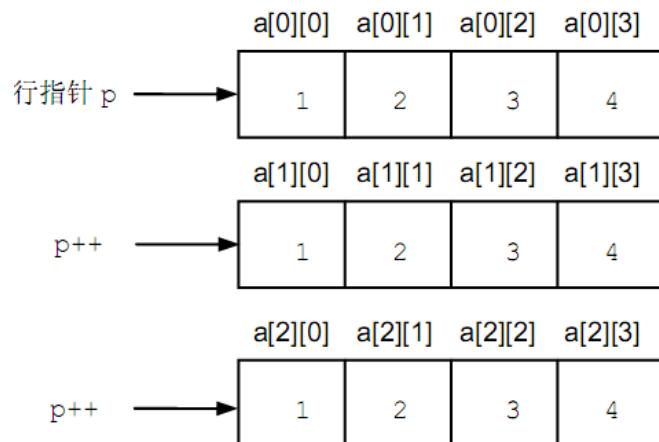
11.2 指针和二维数组间的关系

■ 二维数组的行指针

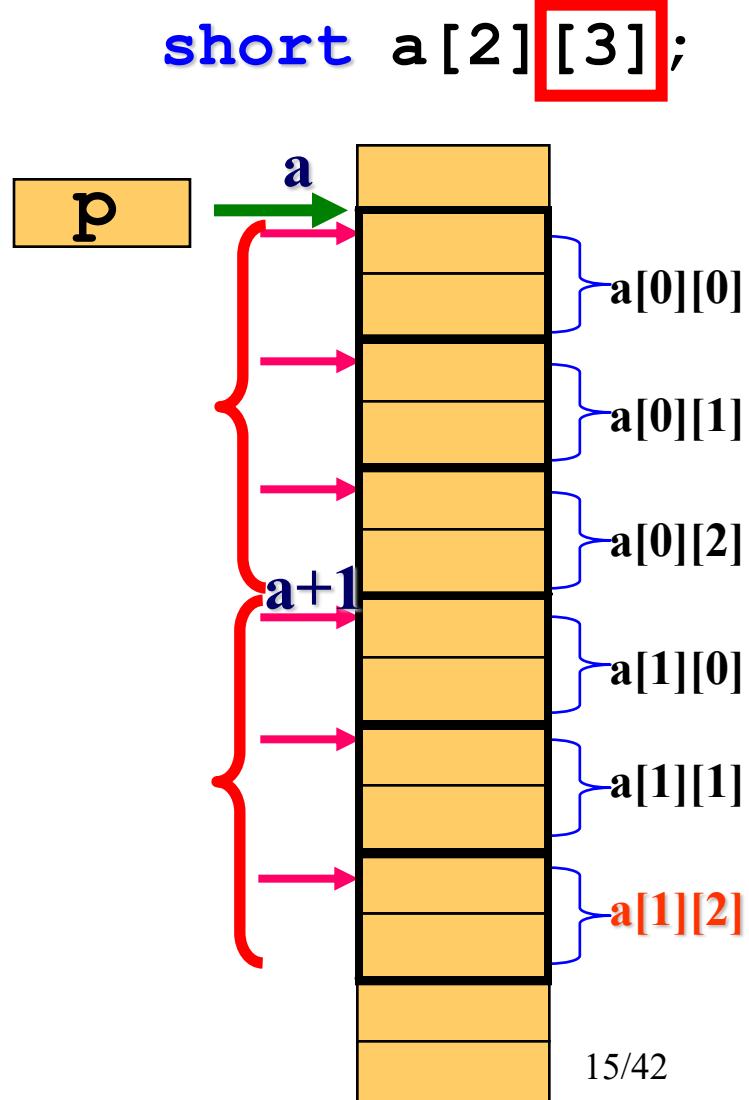
`int (*p)[3];`

`p = a; //用行地址初始化`

■ 逐行查找-> 逐列查找



`short a[2][3];`



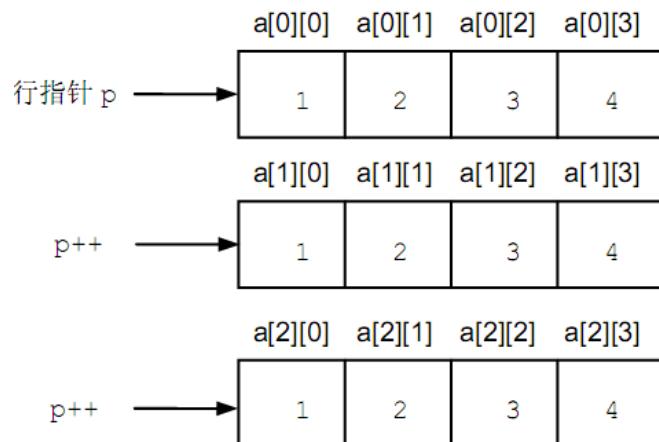
11.2 指针和二维数组间的关系

■ 二维数组的行指针

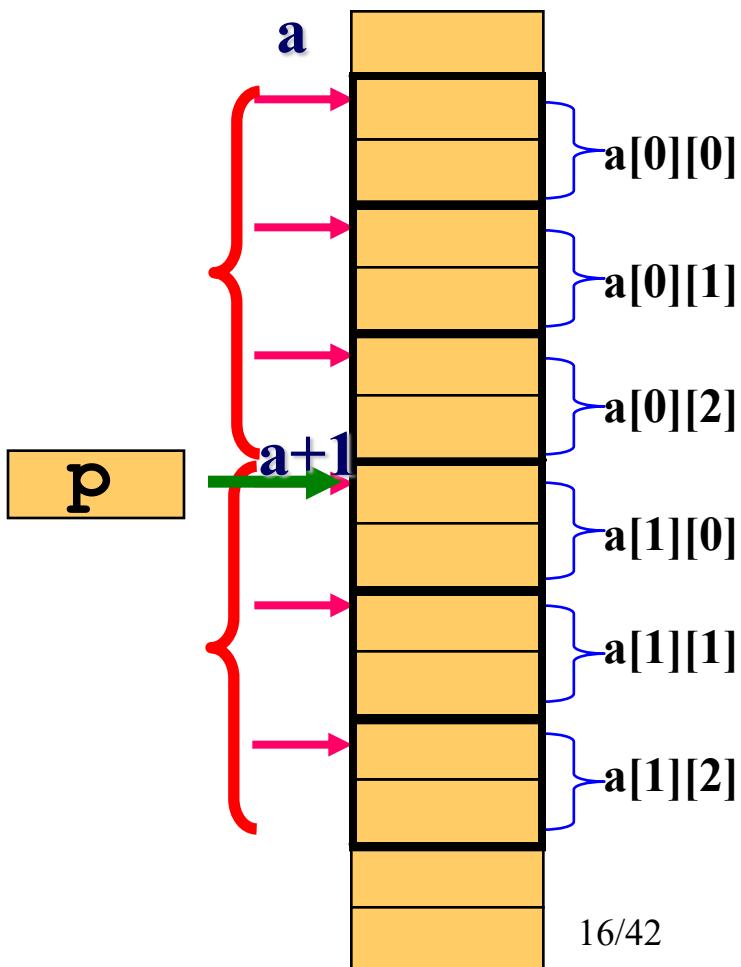
```
int (*p) [3];
```

p = a; //用行地址初始化

■ 逐行查找-> 逐列查找



```
short a[2][3];
```



11.2 指针和二维数组间的关系

■ 二维数组的行指针

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

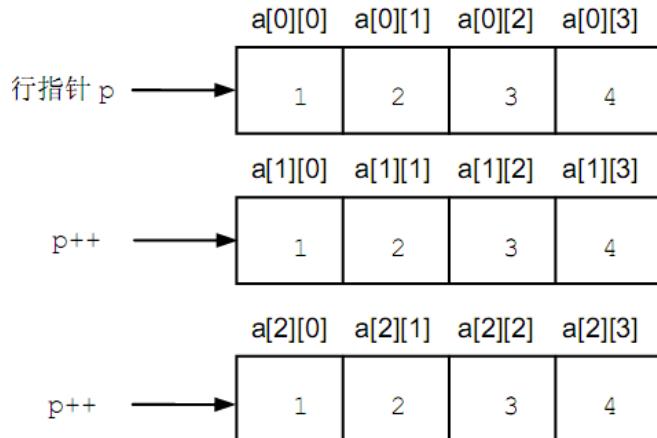
p = a; //用行地址初始化

■ 逐行查找-> 逐列查找

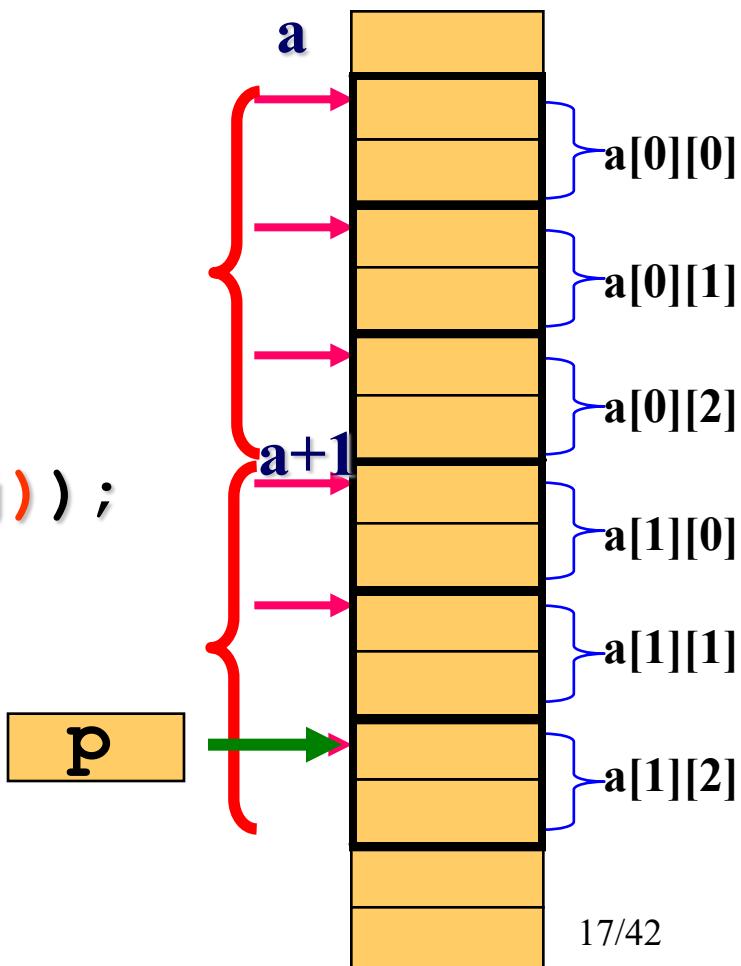
```
for (i=0; i<m; i++)
```

```
for (j=0; j<n; j++)
```

```
printf ("%d", *(*(p+i)+j));
```



```
short a[2][3];
```



11.2 指针和二维数组间的关系

■ 二维数组的列指针

```
int *p;
```

```
p = *a; //用列地址初始化
```

■ 逐个查找——相对偏移量

```
for (i=0; i<m; i++)
```

```
for (j=0; j<n; j++)
```

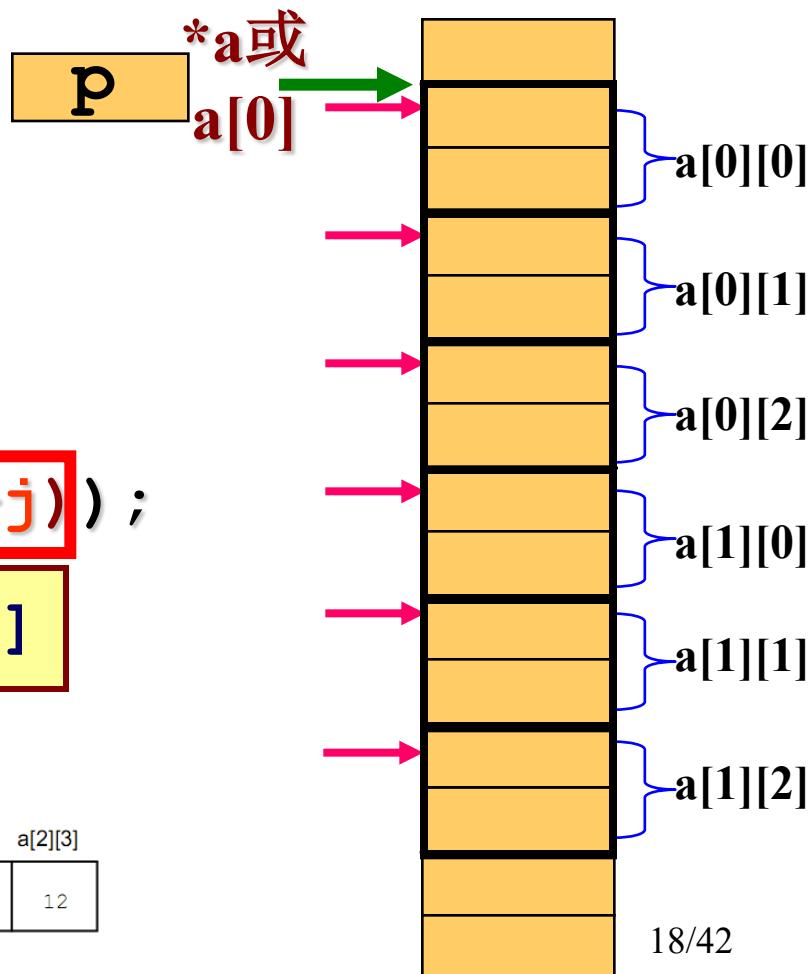
```
printf ("%d", *(p+i*n+j));
```

p[i*n+j]

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]
1	2	3	4	5	6	7	8	9	10	11	12

列指针 p p++ p++

```
short a[2][3];
```



11.2 指针和二维数组间的关系

【例11.3】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

```
Input 3*4 numbers:  
1 2 3 4 5 6 7 8 9 10 11 12  
1 2 3 4  
5 6 7 8  
9 10 11 12
```

```
1 #include <stdio.h>  
2  
3 #define N 4  
4  
5 void InputArray(int p[][N], int m, int n);  
6 void OutputArray(int p[][N], int m, int n);  
7  
8 int main()  
9 {  
10     int a[3][4];  
11     printf("Input 3*4 numbers:\n");  
12     InputArray(a, 3, 4); /* 向函数传递二维数组的第0行的地址 */  
13     OutputArray(a, 3, 4); /* 向函数传递二维数组的第0行的地址 */  
14     return 0;  
15 }
```

11.2 指针和二维数组间的关系

【例11.3】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]);
        }
    }
}
```

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

形参声明为列数已知的
二维数组

```
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");
    }
}
```

11.2 指针和二维数组间的关系

【例11.3】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);
        }
    }
}
```

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

形参声明为指向列数已知的二维数组的行指针

```
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");
    }
}
```

11.2 指针和二维数组间的关系

【例11.3】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]);
        }
    }
}
```

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

形参声明为指向二维数组
的列指针

```
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");
    }
}
```

11.2 指针和二维数组间的关系

【例11.3】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

```
1 #include <stdio.h>
2
3 void InputArray(int *p, int m, int n);
4 void OutputArray(int *p, int m, int n);
5
6 int main()
7 {
8     int a[3][4];
9     printf("Input 3*4 numbers:\n");
10    InputArray(*a, 3, 4); /* 向函数传递二维数组的第0行第0列的地址 */
11    OutputArray(*a, 3, 4); /* 向函数传递二维数组的第0行第0列的地址 */
12    return 0;
13 }
```

指针和数组作函数参数

- 通过指针或数组参数，使调用者获得修改后的数据
- 通过一个参数把大量的数据送到函数内
 - 如果只向内传送数据，就把参数定义为**const**，防止意外修改数据，也让函数的功能更明确
- **void PrintArray(**const int** *p, **int** n)**
{

}
- **void PrintArray(**const int** a[], **int** n)**
{

}

指针、数组以及其他的数据类型混合

- 基本数据类型
 - `int`、`long`、`char`、`short`、`float`、`double`.....
- 数组是一种数据类型
 - 是从其他类型派生的类型
 - 每个元素都有一个类型
- 指针是一种数据类型
 - 是从其他类型派生的类型
 - XX类型的指针
- 任何类型都可以作指针或者数组的基类型

11.3 指针数组及其应用

■ 用指针作数组的基类型——？

■ 指针数组 (Pointer Array)

— 元素均为指针类型数据的数组

■ 定义形式为：

数据类型 *数组名[数组长度];

■ 例如

char *ptr[5];

ptr → [5] → * → char



第10章【例10.4】国名字字符串排序—二维数组

```
char name[N][MAX_LEN];
```

```
...
```

```
for (i=0; i<n-1; i++)
```

```
{
```

```
    for (j = i+1; j<n; j++)
```

```
{
```

```
    if(strcmp(str[j], str[i])<0)
```

```
{
```

```
        strcpy(temp, str[i]);  
        strcpy(str[i], str[j]);  
        strcpy(str[j], temp);
```

```
}
```

交换字符数组中的字符串

2024/3/20

字符串排序前

A	m	e	r	i	c	a	\0	\0	\0
E	n	g	l	a	n	d	\0	\0	\0
A	u	s	t	r	a	l	i	a	\0
S	w	e	d	e	n	\0	\0	\0	\0
F	i	n	l	a	n	d	\0	\0	\0

N

MAX_LEN

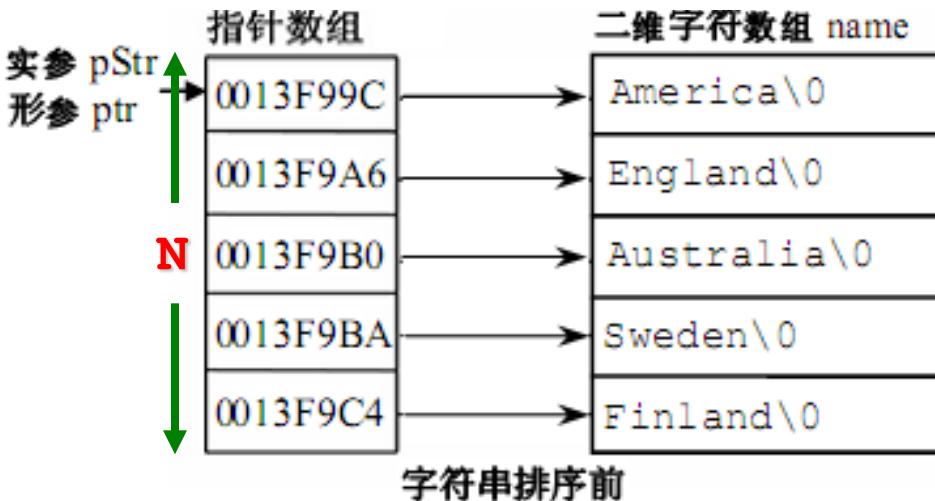
物理排序

字符串排序后

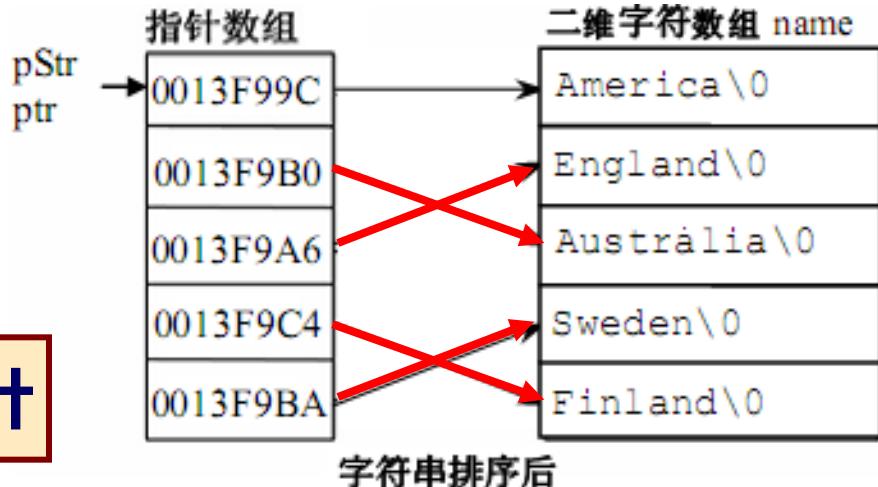
A	m	e	r	i	c	a	\0	\0	\0
A	u	s	t	r	a	l	i	a	\0
E	n	g	l	a	n	d	\0	\0	\0
F	i	n	l	a	n	d	\0	\0	\0
S	w	e	d	e	n	\0	\0	\0	\0

【例11.4】国名字符串排序——指针数组

```
char *ptr[N];  
...  
  
for (i=0; i<n-1; i++)  
{  
    for (j = i+1; j<n; j++)  
    {  
        if(strcmp(ptr[j],ptr[i])<0)  
        {  
            temp = ptr[i];  
            ptr[i] = ptr[j];  
            ptr[j] = temp;  
        }  
    }  
}
```



索引排序



交换指针数组中的字符串指针

【例11.4】国名字符串排序—指针数组

```
1 #include <stdio.h>
2 #include <string.h>
3 #define MAX_LEN 10
4 #define N 150
```

```
How many countries? 5↙
Input their names:
America ↴
England ↴
Australia ↴
Sweden ↴
Finland ↴
Sorted results:
America
Australia
England
Finland
Sweden
```

```
6 int main()
7 {
8     int i, n;
9     char name[N][MAX_LEN];
10    char *pStr[N];
11    printf("How many countries?");
12    scanf("%d", &n);
13    gets(pStr[0]);
14    printf("Input their names:\n");
15    for (i=0; i<n; i++)
16    {
17        pStr[i] = name[i];
18        gets(pStr[i]);
19    }
20    SortString(pStr, n);
21    printf("Sorted results:\n");
22    for (i=0; i<n; i++)
23    {
24        puts(pStr[i]);
25    }
26    return 0;
27 }
```

在使用指针数组之前
必须对数组元素进行初始化

【例11.4】国名字符串排序

```
1 #include <stdio.h>
2 #include <string.h>
3 #define MAX_LEN 10          /* 字符串的最大长度 */
4 #define N      150           /* 字符串的个数 */
```

```
How many countries? 5✓
Input their names:
America ✓
England ✓
Australia ✓
Sweden ✓
Finland ✓
Sorted ✓
America
Australia
Sweden
Finland
England
```

b.exe 遇到问题需要关闭。我们对此引起的不便表示抱歉。

如果您正处于进程当中，信息有可能丢失。

请将此问题报告给 Microsoft。
我们已经创建了一个错误报告，您可以将它发送给我们。我们将此报告视为保密的和匿名的。

要查看这个错误报告包含的数据，请单击此处。

调试(D) | 发送错误报告(S) | 不发送(W)

```
6 int main()
7 {
8     int i, n;
9     char name[N][MAX_LEN];
10    char *pStr[N];
11    printf("How many countries?");
12    scanf("%d", &n);
13
14
15    for (i=0; i<n; i++)
16    {
17        gets(pStr[i]);
18        /* pStr[i] = name[i]; */
19    }
20    SortString(pStr, n);
21    printf("Sorted results:\n");
22    for (i=0; i<n; i++)
23    {
24        puts(pStr[i]);
25    }
26    return 0;
27 }
```

能这样输入字符串吗？Why？



11.3.2 指针数组用于表示命令行参数

- GUI界面之前，计算机的操作界面都是字符式的命令行界面（DOS、UNIX、Linux）
- 通过命令行参数（Command Line Arguments），使用户可以根据需要来决定程序干什么、怎么干
- **main(int argc, char* argv[])**
 - 当你把**main**函数写成这样时
 - **argc**的值为：参数的数目+1
 - **argv[0]**为指向命令名的字符指针
 - **argv[x] (x>1)**为指向每个参数的字符指针

【例11.5】演示命令行参数与main函数各形参之间的关系

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

```
{
```

```
    int i;
```

```
    printf("The number of command line arguments is:%d\n",argc);
```

```
    printf("The program name is:%s\n", argv[0]);
```

```
    if (argc > 1)
```

```
{
```

```
        printf("The other arguments are following:\n");
```

```
        for (i = 1; i<argc; i++)
```

```
{
```

```
            printf("%s\n", argv[i]);
```

```
}
```

```
}
```

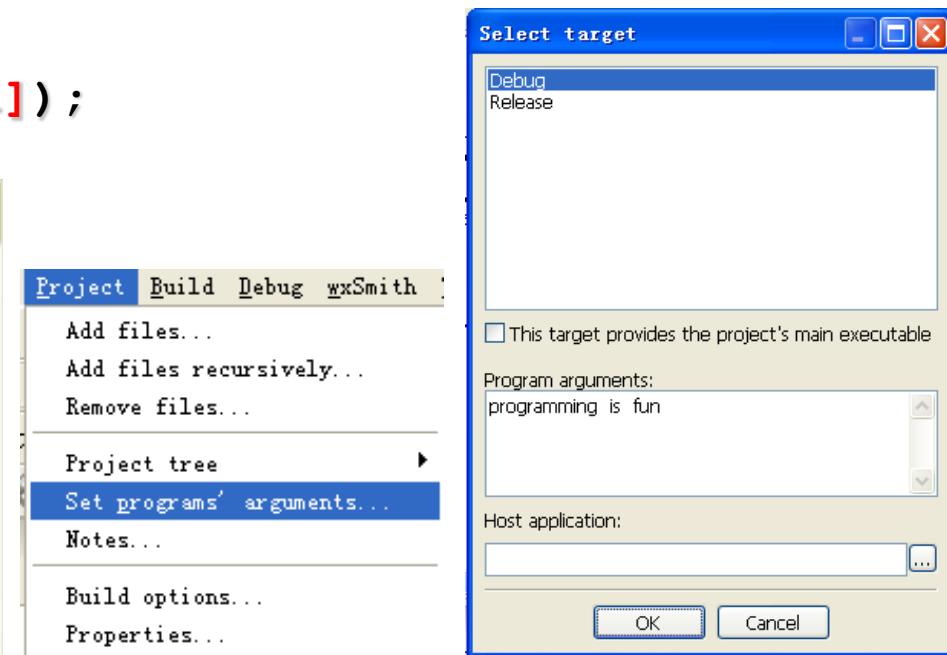
```
return
```



```
命令提示符  
Microsoft Windows XP [版本 5.1.2600]  
<C> 版权所有 1985-2001 Microsoft Corp.  
  
C:\Documents and Settings\ibm>d:  
  
D:\>echo.exe programming is fun  
The number of command line arguments is:4  
The program name is:echo.exe  
The other arguments are following:  
programming  
is  
fun  
  
D:\>
```

如何输入命令行参数?

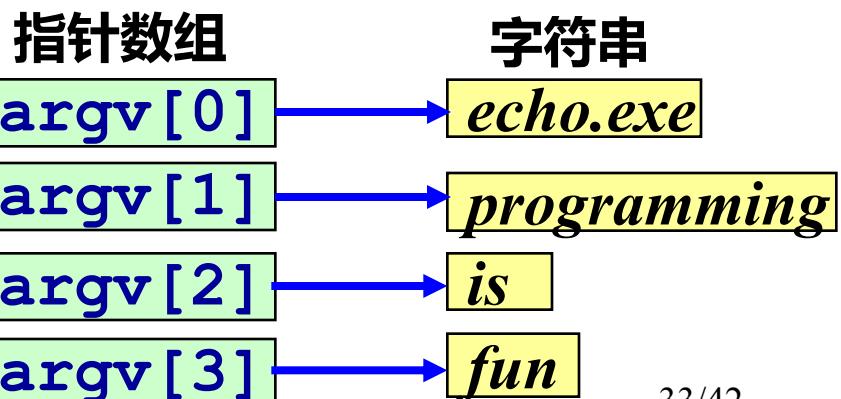
echo.exe programming is fun



【例11.5】演示命令行参数与main函数各形参之间的关系

```
int main(int argc, char *argv[])
{
    int i;
    printf("The number of command line arguments is:%d\n",argc);
    printf("The program name is:%s\n", argv[0]);
    if (argc > 1)
    {
        printf("The other arguments are following:\n");
        for (i = 1; i<argc; i++)
        {
            printf("%s\n", argv[i]);
        }
    }
    return 0;
}
```

The number of command line arguments is: 4
The program name is: echo.exe
The other arguments are following:
programming
is
fun



11.4.1 C程序的内存映像

C程序中变量的内存分配方式

- 从静态存储区分配

- 全局变量和静态变量

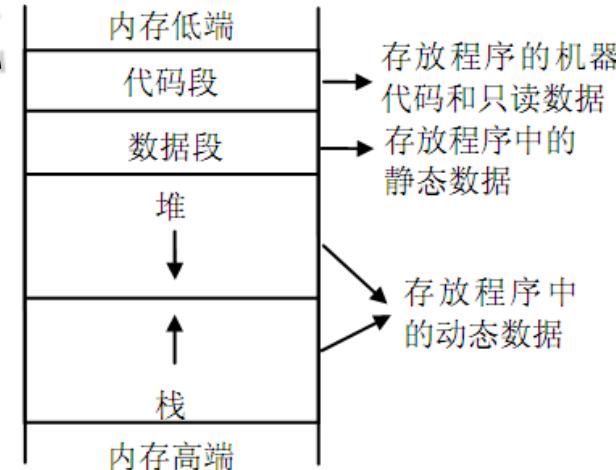
- 在栈上创建

- 存放函数参数值、局部变量值等

- 在执行函数调用时，系统在栈上为函数内的局部变量及形参分配内存，函数执行结束时，自动释放这些内存

- 从堆上分配

- 在程序运行期间，用动态内存分配函数来申请的内存都是从堆上分配的，动态内存的生存期由程序员自己来决定



11.4.2 动态内存分配函数

- Two primary methods of allocating memory:

```
#include <stdlib.h>
#include <alloc.h>
```

```
void* malloc(unsigned int size);
```

```
void* calloc(unsigned int num,
             unsigned int size);
```

void*类型的指针可以指向任意类型的变量，通常强转(**Type***)为其他类型

11.4.2 动态内存分配函数

- Two primary methods of allocating memory:

```
void* malloc(unsigned int size);
```

向系统申请大小为size的内存块
把首地址返回，若申请不成功则返回NULL

```
void* calloc(unsigned int num,  
            unsigned int size);
```

向系统申请num个size大小的内存块
把首地址返回，若申请不成功则返回NULL

11.4.2 动态内存分配函数

- Method of deallocating memory:

```
void* free(void* p);
```

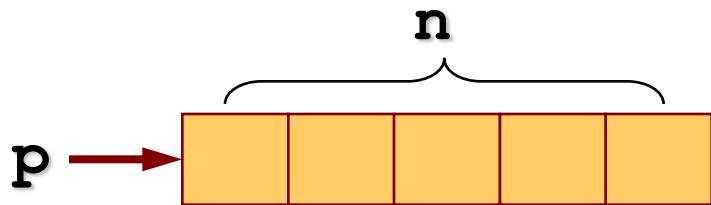
释放由malloc()和calloc()申请的内存块
p是指向此块内存的指针
free时系统标记此块内存为未占用，可被重新分配

【例11.6】 一维动态数组

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void InputArray(int *p, int n);
4 double Average(int *p, int n);
5 int main()
6 {
7     int *p = NULL, n;
8     double aver;
9     printf("How many students?");
10    scanf("%d", &n);
11    p = (int *) malloc(n * sizeof(int));
12    if (p == NULL)
13    {
14        printf("No enough memory!\n");
15        exit(1);
16    }
17    printf("Input %d score:", n);
18    InputArray(p, n);
19    aver = Average(p, n);
20    printf("aver = %.1f\n", aver);
21    free(p);
22    return 0;
23 }

```



确保指针使用前是非空指针

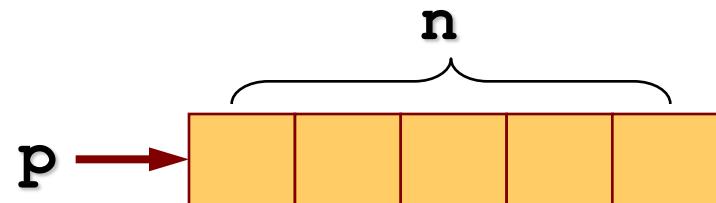
释放向系统申请的存储空间

【例11.6】 一维动态数组

```

24  /* 形参声明为指针变量，输入数组元素值 */
25  void InputArray(int *p, int n)
26  {
27      int i;
28      for (i=0; i<n; i++)
29      {
30          scanf("%d", &p[i]);
31      }
32  }
33  /* 形参声明为指针变量，计算数组元素的平均值 */
34  double Average(int *p, int n)
35  {
36      int i, sum = 0;
37      for (i=0; i<n; i++)
38      {
39          sum = sum + p[i];
40      }
41      return (double)sum / n;
42  }

```



像使用一维数组一样
使用动态数组

```

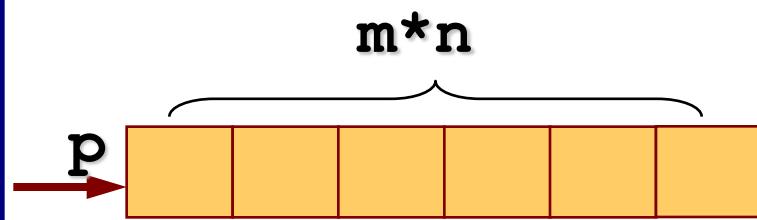
How many students? 5↙
Input 5 score: 90 85 70 95 80↙
aver = 84.0

```

11.4.4

【例11.7】 二维动态数组

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void InputArray(int *p, int m, int n);
4 double Average(int *p, int m, int n);
5 int main()
6 {
7     int *p = NULL, m, n;
8     double aver;
9     printf("How many classes?");
10    scanf("%d", &m);
11    printf("How many students in a class?");
12    scanf("%d", &n);
13    p = (int *)calloc(m*n, sizeof(int));
14    if (p == NULL)
15    {
16        printf("No enough memory!\n");
17        exit(1);
18    }
19    InputArray(p, m, n);
20    aver = Average(p, m, n);
21    printf("aver = %.1f\n", aver);
22    free(p);
23    return 0;
24 }
```



确保指针使用前是
非空指针

释放向系统申请的
存储空间

11.4.4

【例11.7】 二维动态数组

仍当作一维数组
来使用

$m \times n$



```
How many classes? 3✓
How many students in a class? 4✓
Please enter scores of class 1:
81 72 73 64✓
Please enter scores of class 2:
65 86 77 88✓
Please enter scores of class 3:
91 90 85 92✓
aver = 80.3
```

```
25 /* 形参声明为指向二维数组的列指针，输入数组元素值 */
26 void InputArray(int *p, int m, int n)
27 {
28     int i, j;
29     for(i = 0; i<m; i++)          /* m 个班 */
30     {
31         printf("Please enter scores of class %d:\n", i+1);
32         for(j = 0; j<n; j++)    /* 每班 n 个学生 */
33         {
34             scanf("%d", &p[i*n+j]);
35         }
36     }
37 }
```

```
38 /* 形参声明为指针变量，计算数组元素的平均值 */
39 double Average(int *p, int m, int n)
40 {
41     int i, j, sum = 0;
42     for(i = 0; i<m; i++)          /* m 个班 */
43     {
44         for(j = 0; j<n; j++)    /* 每班 n 个学生 */
45         {
46             sum = sum + p[i*n+j];
47         }
48     }
49     return (double)sum / (m*n);
50 }
```



■ Questions and answers

