《计算概论A》课程 程序设计部分

指针（2）

李 戈

北京大学信息科学技术学院

lige@sei.pku.edu.cn

本节内容

- 字符串与指针
- 指向二维数组的指针
    - 再谈一维数组的地址
    - 指向二维数组的指针
    - 利用指针遍历二维数组

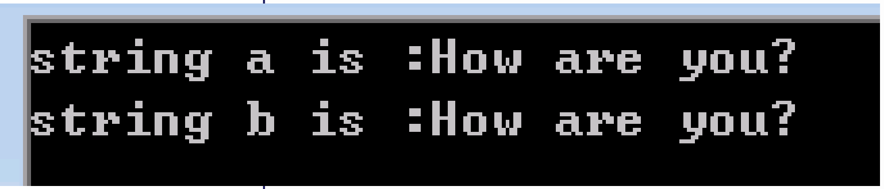
字符串与指针

- 指向数组的指针
  - int a[10];   int *p; p = a;
- 指向字符串的指针
  - 指向字符串的指针变量：
  - char a[10];   char *p; p = a;

字符串指针举例

请说明一下程序完成了什么任务：

```
#include<iostream>
#include <iomanip>
using namespace std;
int main()
{
    char a[] = "How are you?", b[20];
    char *p1, *p2;
    for (p1 = a, p2 = b; *p1 != '\0'; p1++, p2++)
        *p2 = *p1;
    *p2 = '\0';
    cout << "string a is :" << a << endl;
    cout << "string b is :" << b << endl;
    return 0;
}
```



```
string a is :How are you?
string b is :How are you?
```

```
int main()
{
    int a = 5;
    int *pa = &a;

    int b[6] = {1, 2, 3, 4, 5, 6};
    int *pb = b;

    char c[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
    char *pc = c;

    cout<< a <<endl;
    cout<< pa <<endl<<endl;

    cout<< b <<endl;
    cout<< pb <<endl<<endl;

    cout<< c <<endl;
    cout<< pc <<endl;

    return 0;
}
```

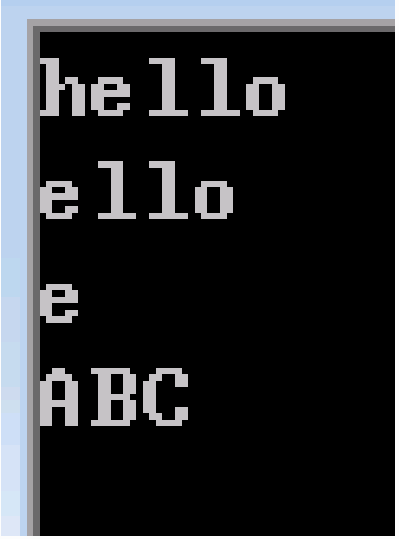
```
5
0x0013FF7C

0x0013FF60
0x0013FF60

hello
hello
```

```
int main()
{
    int a = 5;
    int *pa = &a;

    int b[6] = {1,2,3,4,5,6};
    int *pb = b;

    char c[6] = {'h','e','l','l','o','\0'};
    char *pc = c;

    cout<< a <<endl;
    cout<< pa <<endl<<endl;

    cout<< b <<endl;
    cout<< pb <<endl<<endl;

    cout<<static_cast<void*>(c)<<endl;
    cout<<static_cast<void*>(pc)<<endl;

    return 0;
}
```

```
5
0x0013FF7C

0x0013FF60
0x0013FF60

0x0013FF54
0x0013FF54
```

字符串指针举例

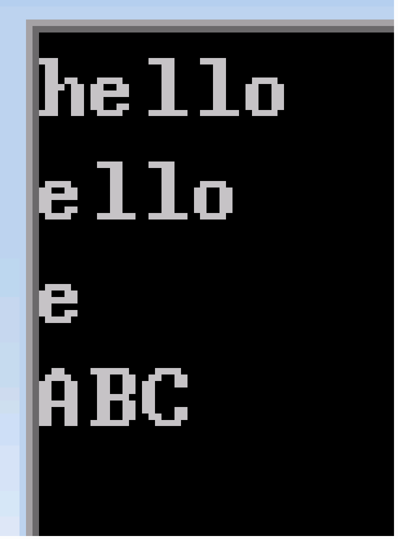
```
#include<iostream>
using namespace std;
int main()
{
    char buffer[10] = "ABC";
    char *pc;
    pc = "hello";
    cout << pc << endl;
    pc++;
    cout << pc << endl;
    cout << *pc << endl;
    pc = buffer;
    cout << pc;
    return 0;
}
```



```
hello
ello
e
ABC
```

字符串指针举例

```
#include<iostream>
using namespace std;
int main()
{
    char buffer[10] = "ABC";
    char *pc;
    pc = "hello";
    cout << pc << endl;
    pc++;
    cout << pc << endl;
    cout << *pc << endl;
    pc = buffer;
    cout << pc;
    return 0;
}
```

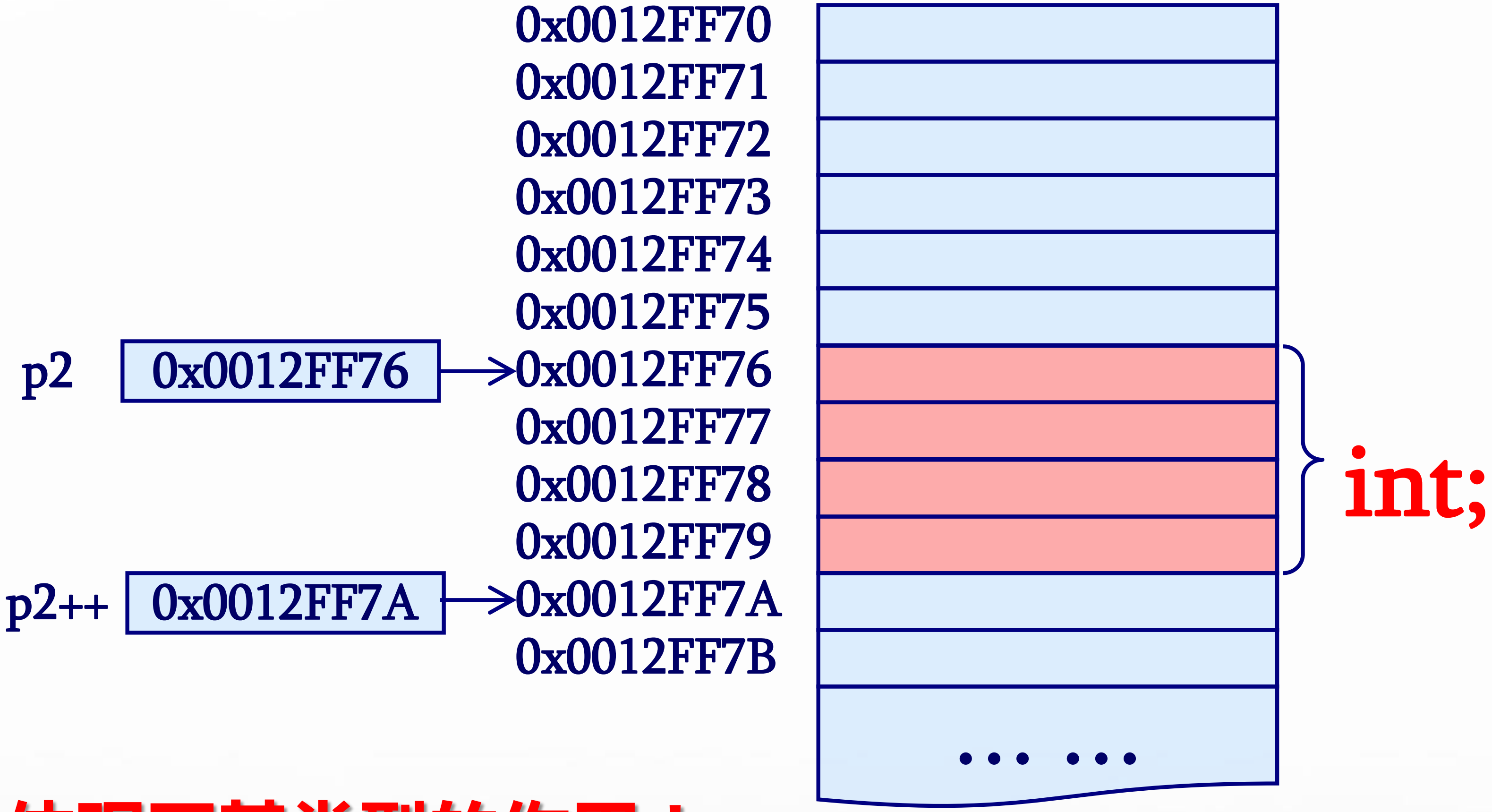


```
hello
ello
e
ABC
```

本节内容

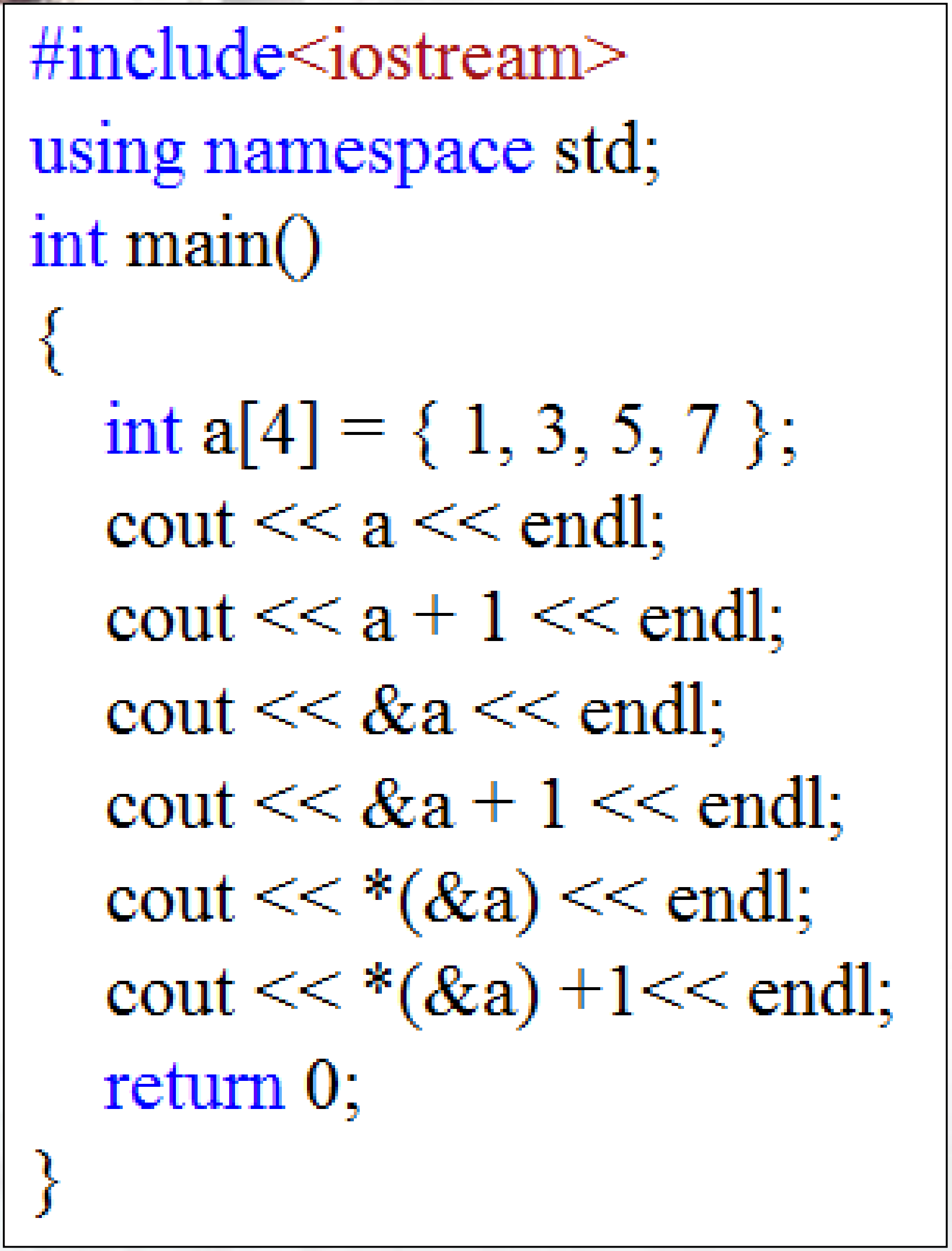
- 字符串与指针
- 指向二维数组的指针
  - 再谈一维数组的地址
  - 指向二维数组的指针
  - 利用指针遍历二维数组

pointer++ 的含义



体现了基类型的作用！

再谈一维数组的地址

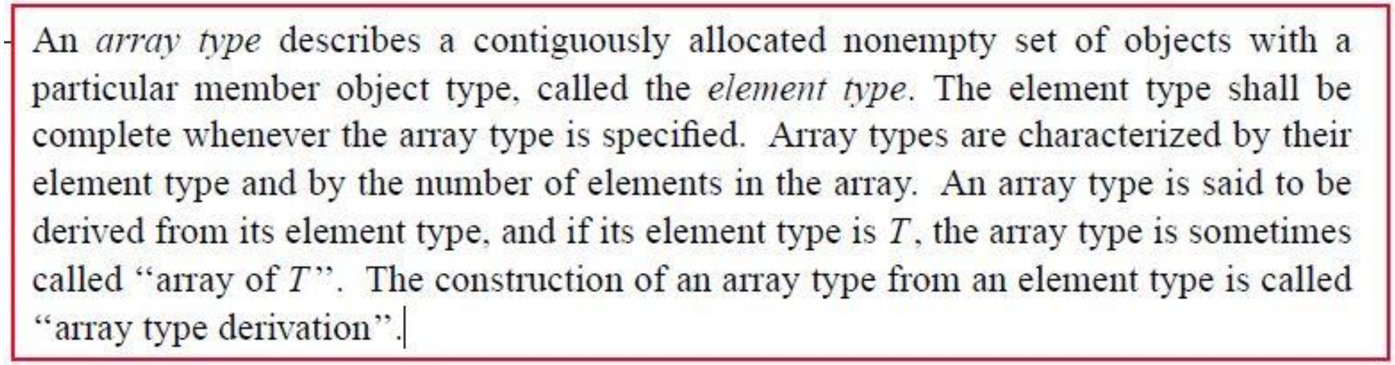


```
#include<iostream>
using namespace std;
int main()
{
    int a[4] = { 1, 3, 5, 7 };
    cout << a << endl;
    cout << a + 1 << endl;
    cout << &a << endl;
    cout << &a + 1 << endl;
    cout << *(&a) << endl;
    cout << *(&a) +1<< endl;
    return 0;
}
```

0028F7C2	
0028F7C3	
0028F7C4	a[0]
0028F7C5	
0028F7C6	
0028F7C7	
0028F7C8	a[1]
0028F7C9	
0028F7CA	
0028F7CB	
0028F7CC	a[2]
0028F7CD	
0028F7CE	
0028F7CF	
0028F7D0	a[3]
0028F7D1	
0028F7D2	
0028F7D3	
0028F7D4	
0028F7D5	
0028F7D6	
0028F7D7	
0028F7D8	
0028F7D9	

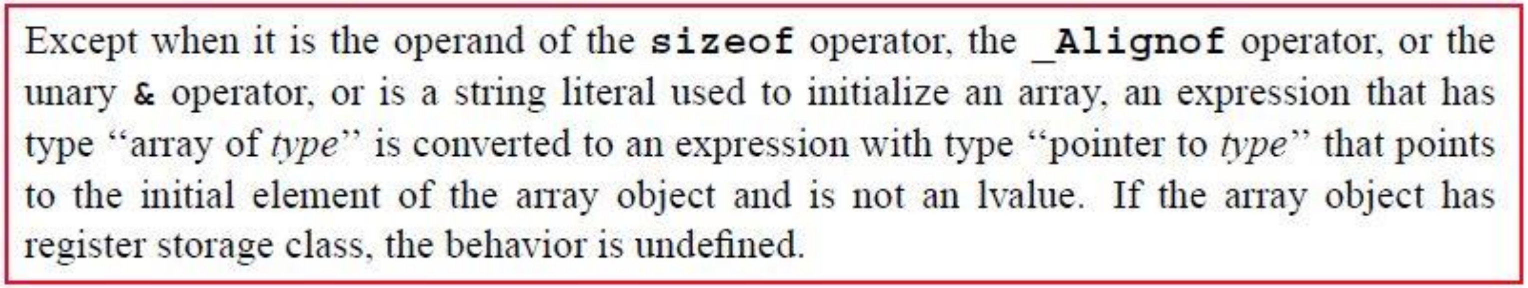
数组的地址

§6.2.5 Types



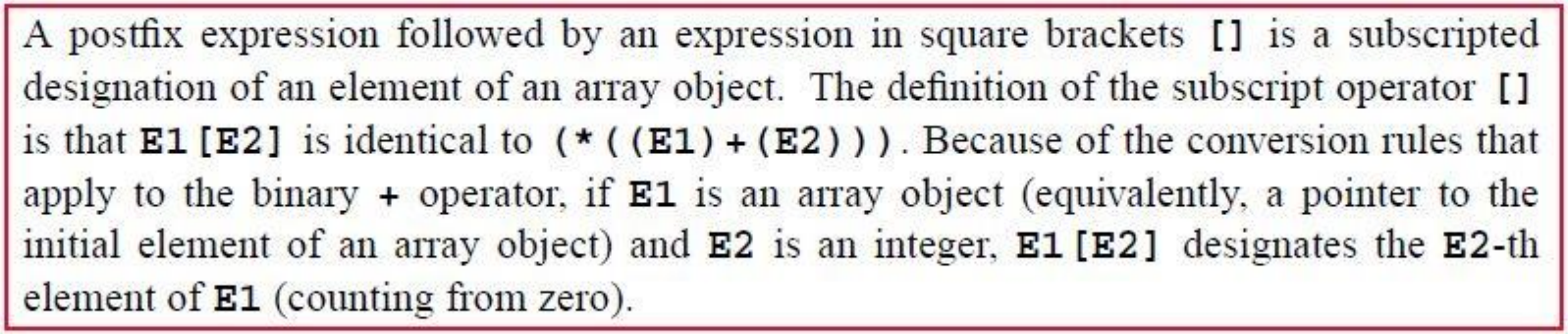
An *array type* describes a contiguously allocated nonempty set of objects with a particular member object type, called the *element type*. The element type shall be complete whenever the array type is specified. Array types are characterized by their element type and by the number of elements in the array. An array type is said to be derived from its element type, and if its element type is *T*, the array type is sometimes called "array of *T*". The construction of an array type from an element type is called "array type derivation".

§6.3.2.1 Lvalues, arrays, and function designators



Except when it is the operand of the `sizeof` operator, the `_Alignof` operator, or the unary `&` operator, or is a string literal used to initialize an array, an expression that has type "array of *type*" is converted to an expression with type "pointer to *type*" that points to the initial element of the array object and is not an lvalue. If the array object has register storage class, the behavior is undefined.

§6.5.2.1 Array subscripting



A postfix expression followed by an expression in square brackets `[]` is a subscripted designation of an element of an array object. The definition of the subscript operator `[]` is that `E1[E2]` is identical to `(*((E1)+(E2)))`. Because of the conversion rules that apply to the binary `+` operator, if `E1` is an array object (equivalently, a pointer to the initial element of an array object) and `E2` is an integer, `E1[E2]` designates the `E2`-th element of `E1` (counting from zero).

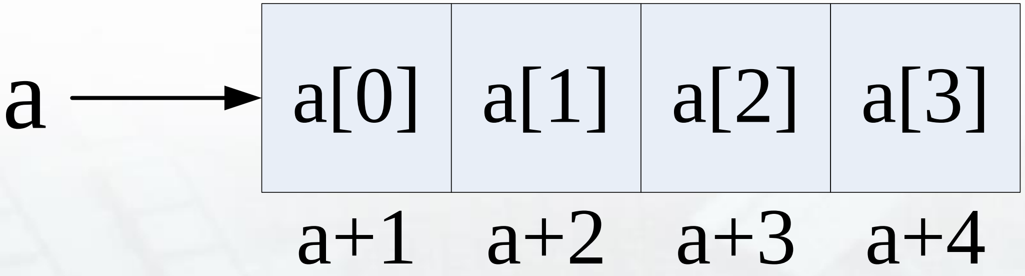
ISO/IEC 9899:2011, Information technology -- Programming languages -- C

再谈一维数组

【数组名相当于指向数组第一个元素的指针】

若 a 是 指向数组第一个元素 的指针，即a相当于&a[0]；

- **&a**是“指向数组”的指针；&a+1将跨越16个字节；
  - ●**&a 相当于 管辖范围“上升”了一级；**
- ***a** 是数组的第一个元素a[0]；即*a等价于a[0] ；
  - ●***a 相当于 管辖范围“下降”了一级；**



本节内容

- 字符串与指针
- 指向二维数组的指针
  - 再谈一维数组的地址
  - 指向二维数组的指针
  - 利用指针遍历二维数组

再谈一维数组

【数组名相当于指向数组第一个元素的指针】

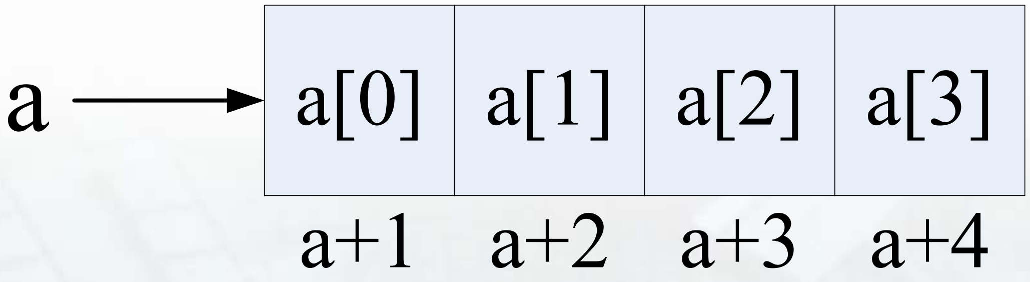
若 a 是 指向数组第一个元素 的指针，即a相当于&a[0]；

◆ &a是“指向数组”的指针；&a+1将跨越16个字节；

●&a 相当于 管辖范围“上升”了一级；

◆ *a 是数组的第一个元素a[0]；即*a等价于a[0] ；

●*a 相当于 管辖范围“下降”了一级；



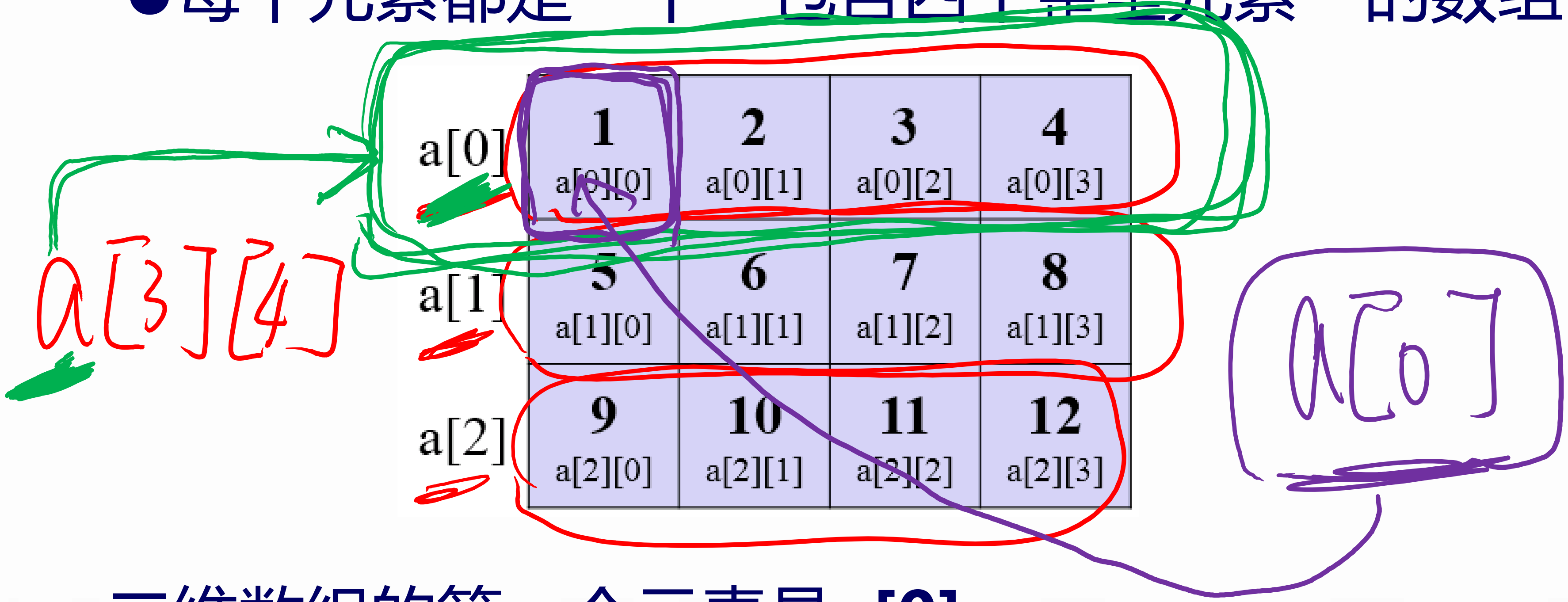
a

a[0] a[1] a[2] a[3]

a+1 a+2 a+3 a+4

二维数组 也一样！

- 二维数组的定义
  - ◆二维数组a[3][4]包含三个元素：a[0], a[1], a[2]
    - ●每个元素都是一个“包含四个整型元素”的数组

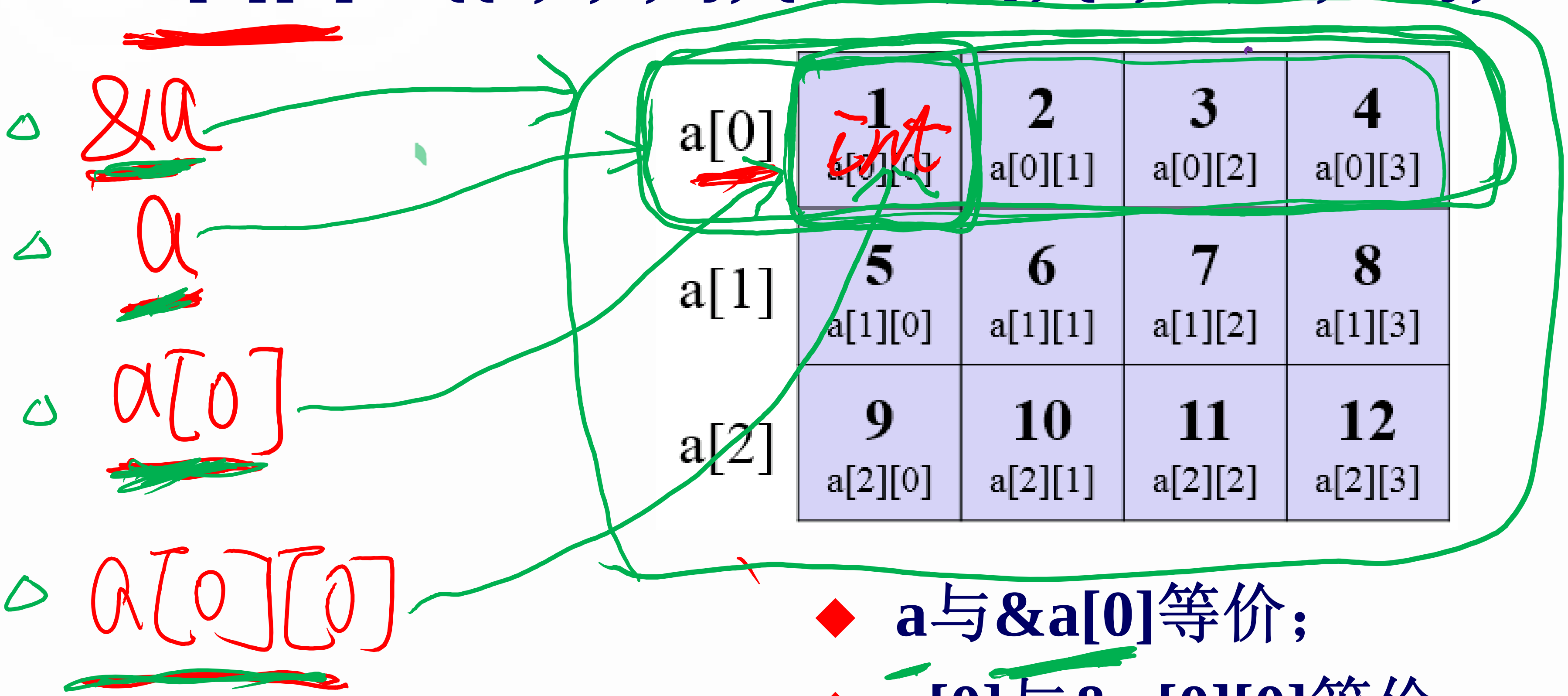


a[0]	1 a[0][0]	2 a[0][1]	3 a[0][2]	4 a[0][3]
a[1]	5 a[1][0]	6 a[1][1]	7 a[1][2]	8 a[1][3]
a[2]	9 a[2][0]	10 a[2][1]	11 a[2][2]	12 a[2][3]

- ◆ 二维数组的第一个元素是**a[0]**
- ◆ a[0]是一个“包含四个整型元素”的一维数组；

二维数组 也一样！

int a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};



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

- **a与&a[0]等价；**
- **a[0]与&a[0][0]等价；**
- **a[0]与*a等价；**
- **a[0][0]与**a等价；**

小结

- 三条规律
  - 数组名相当于指向数组第一个元素的指针；
  - &E 相当于把E的管辖范围上升了一个级别
  - *E 相当于把E的管辖范围下降了一个级别

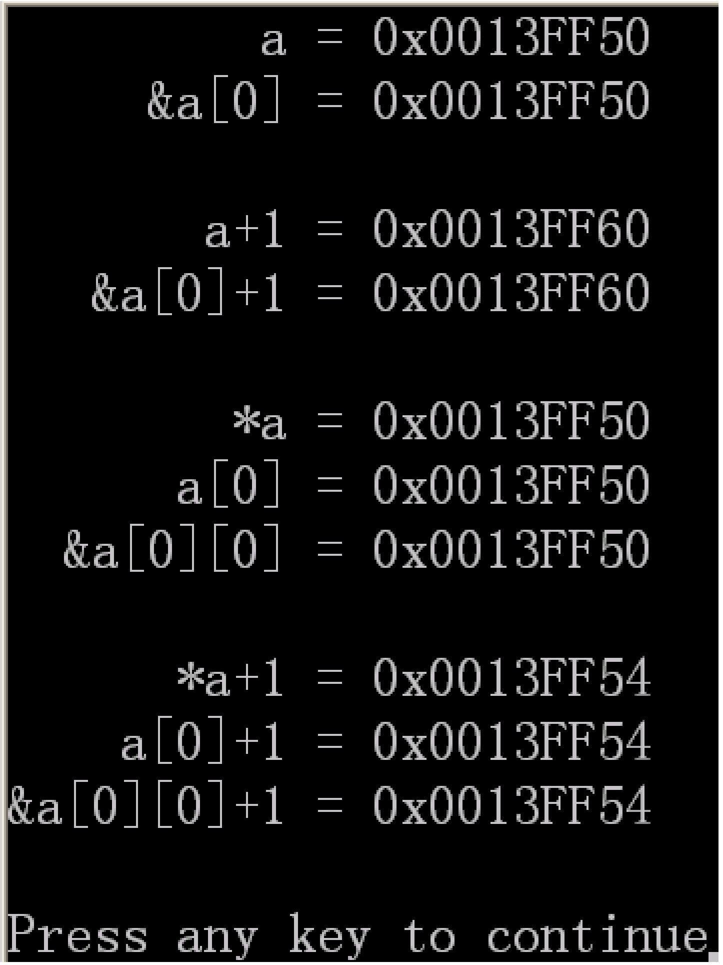
二维数组的地址

```
int main()
{
    int a[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

    cout<<"        a = "<<a<<endl;
    cout<<"    &a[0] = "<<&a[0]<<endl<<endl;

    cout<<"      a+1 = "<<a+1<<endl;
    cout<<"  &a[0]+1 = "<<&a[0]+1<<endl<<endl;

    cout<<"       *a = "<<*a<<endl;
    cout<<"     a[0] = "<<a[0]<<endl;
    cout<<" &a[0][0] = "<<&a[0][0]<<endl<<endl;

    cout<<"     *a+1 = "<<*a+1<<endl;
    cout<<"   a[0]+1 = "<<a[0]+1<<endl;
    cout<<"&a[0][0]+1 = "<<&a[0][0]+1<<endl<<endl;
    return 0;
}
```

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

二维数组的地址

```
int main()
{
   int a[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

   cout<<"        a = "<<a<<endl;
   cout<<"    &a[0] = "<<&a[0]<<endl<<endl;

   cout<<"      a+1 = "<<a+1<<endl;
   cout<<"  &a[0]+1 = "<<&a[0]+1<<endl<<e

   cout<<"       *a = "<<*a<<endl;
   cout<<"     a[0] = "<<a[0]<<endl;
   cout<<" &a[0][0] = "<<&a[0][0]<<endl<<e

   cout<<"     *a+1 = "<<*a+1<<endl;
   cout<<"   a[0]+1 = "<<a[0]+1<<endl;
   cout<<"&a[0][0]+1 = "<<&a[0][0]+1<<endl
   return 0;
}
```



```
         a = 0x0013FF50
     &a[0] = 0x0013FF50

       a+1 = 0x0013FF60
   &a[0]+1 = 0x0013FF60

        *a = 0x0013FF50
      a[0] = 0x0013FF50
  &a[0][0] = 0x0013FF50

      *a+1 = 0x0013FF54
    a[0]+1 = 0x0013FF54
&a[0][0]+1 = 0x0013FF54

Press any key to continue
```

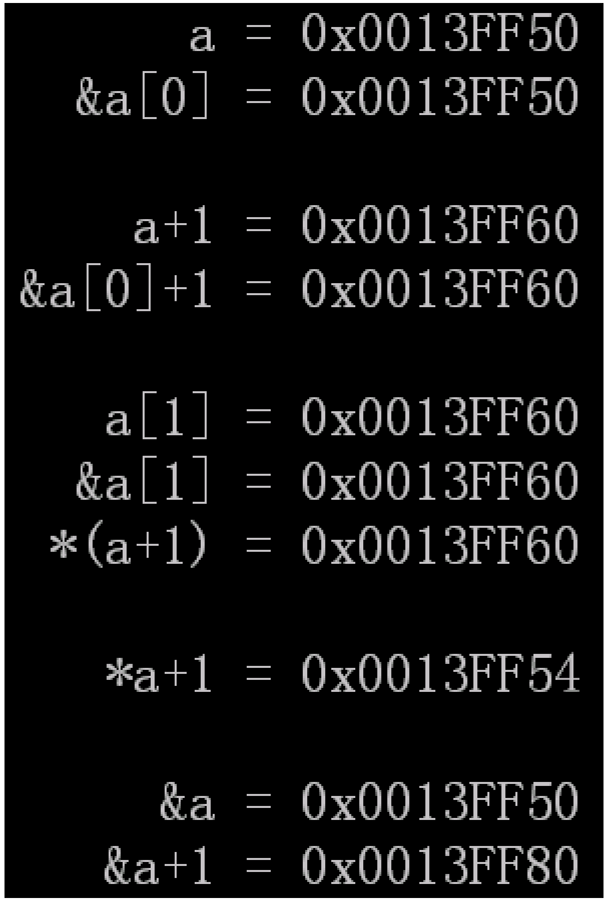
二维数组的地址

```
#include<iostream.h>
void main()
{
    int a[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

    cout<<"        a = "<<a<<endl;
    cout<<"     &a[0] = "<<&a[0]<<endl<<endl;

    cout<<"      a+1 = "<<a+1<<endl;
    cout<<"   &a[0]+1 = "<<&a[0]+1<<endl<<endl;

    cout<<"      a[1] = "<<a[1]<<endl;
    cout<<"     &a[1] = "<<&a[1]<<endl;
    cout<<"    *(a+1) = "<<*(a+1)<<endl<<endl;

    cout<<"      *a+1 = "<<*a+1<<endl<<endl;

    cout<<"        &a = "<<&a<<endl;
    cout<<"      &a+1 = "<<&a+1<<endl;
}
```

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

二维数组的地址

```
#include<iostream.h>
void main()
{
    int a[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

    cout<<"        a = "<<a<<endl;
    cout<<"    &a[0] = "<<&a[0]<<endl<<endl;

    cout<<"      a+1 = "<<a+1<<endl;
    cout<<"  &a[0]+1 = "<<&a[0]+1<<endl<<endl;

    cout<<"     a[1] = "<<a[1]<<endl;
    cout<<"    &a[1] = "<<&a[1]<<endl;
    cout<<"   *(a+1) = "<<*(a+1)<<endl<<endl;

    cout<<"     *a+1 = "<<*a+1<<endl<<endl;

    cout<<"       &a = "<<&a<<endl;
    cout<<"     &a+1 = "<<&a+1<<endl;
}
```



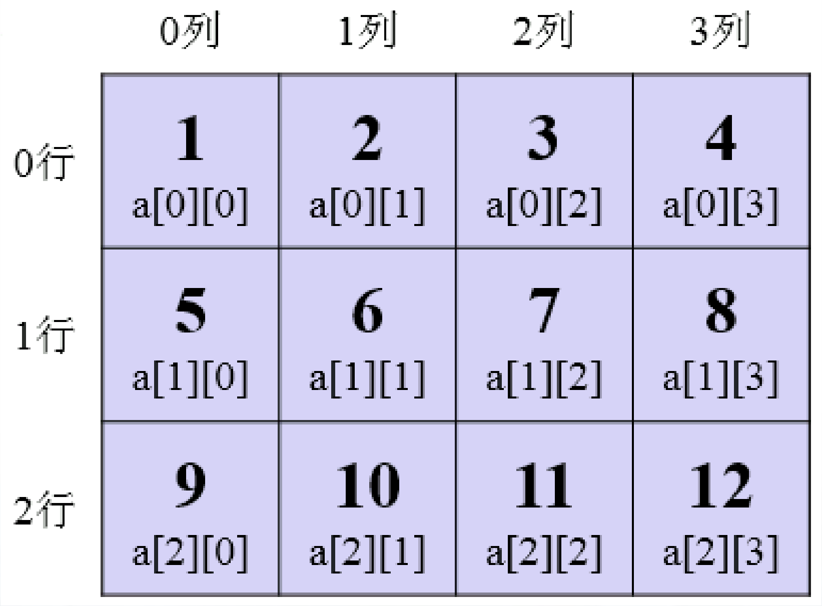
```
        a = 0x0013FF50
    &a[0] = 0x0013FF50

      a+1 = 0x0013FF60
  &a[0]+1 = 0x0013FF60

     a[1] = 0x0013FF60
    &a[1] = 0x0013FF60
   *(a+1) = 0x0013FF60

     *a+1 = 0x0013FF54

       &a = 0x0013FF50
     &a+1 = 0x0013FF80
```

小结

【数组名相当于指向数组第一个元素的指针】

- *a 等价于a[0]，相当于让a下降了一级；
- &a 表示“指向二维数组”的指针，相当于上升了一级；

```
int a[3][4] = {{1,3,5,7},{9,11,13,15},{17,19,21,23}};
```



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

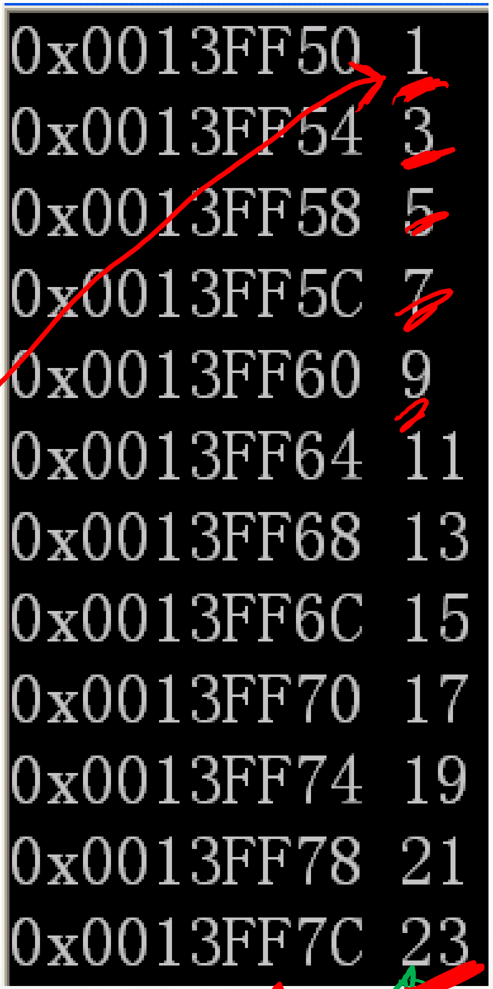
本节内容

- 字符串与指针
- 指向二维数组的指针
  - 再谈一维数组的地址
  - 指向二维数组的指针
  - 利用指针遍历二维数组

遍历数组元素

```
#include<iostream>
using namespace std;
int main( )
{
    int a[3][4] ={1, 3, 5, 7, 9, 11, 13, 15, 17,
    19, 21, 23};
    int *p;
    for(p=&a[0][0]; p<&a[0][0]+12; p++)
    {
        cout<<p<<" "<<*p<<endl;
    }
    return 0;
}
```

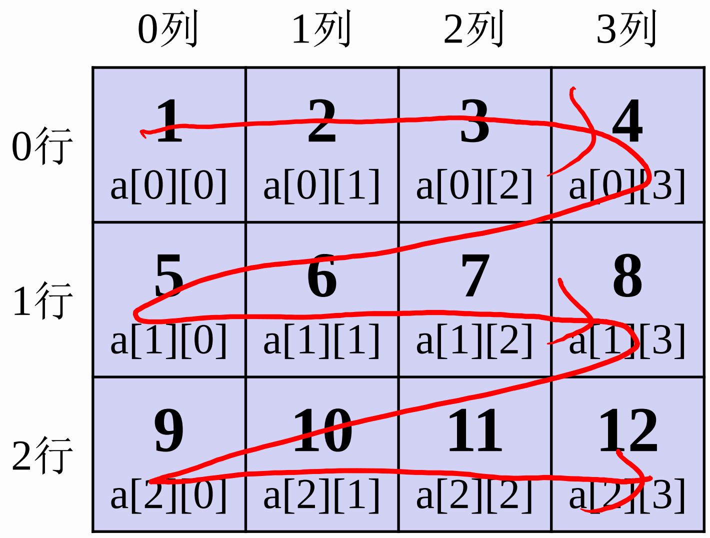
p 元素



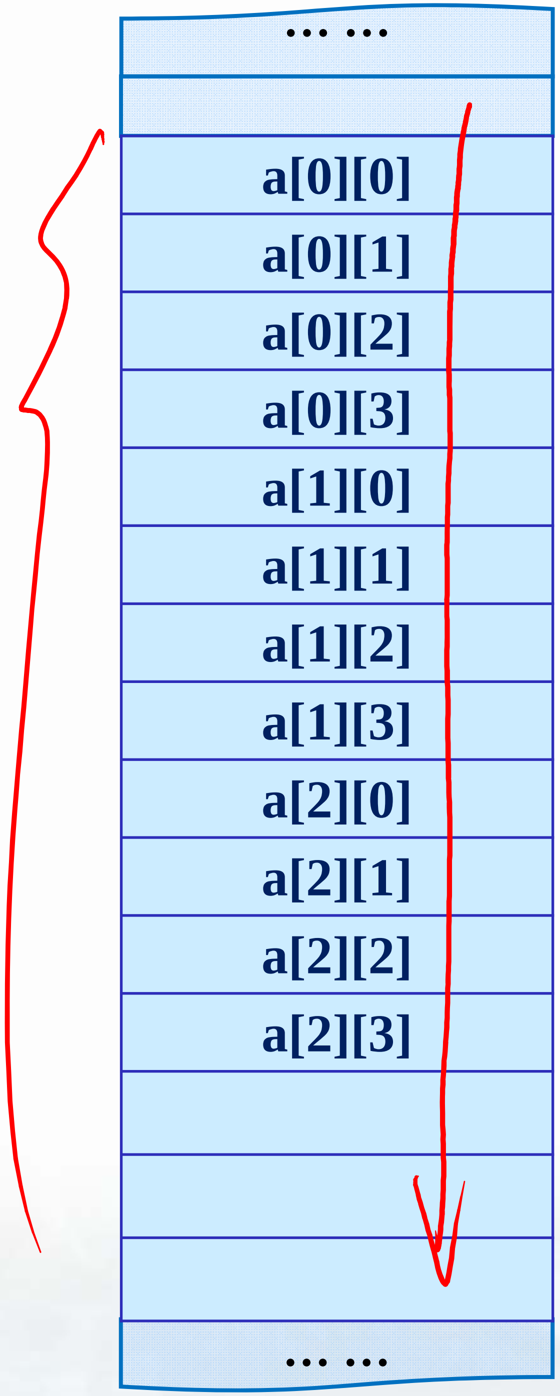
```
0x0013FF50 1
0x0013FF54 3
0x0013FF58 5
0x0013FF5C 7
0x0013FF60 9
0x0013FF64 11
0x0013FF68 13
0x0013FF6C 15
0x0013FF70 17
0x0013FF74 19
0x0013FF78 21
0x0013FF7C 23
```

二维数组

int a[3][4];



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



… …

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]

… …

程序填空&判断正误

- 输入 i, j; 输出a[i][j];

```
int main( )
{
    int a[3][4]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};
    int ________, i,  j;  // 填空：完成变量p的定义
    p = a;
    cin>>i>>j;                // i<3 代表行号，j<4代表列号
    cout<<setw(4)<<*(*(p+i)+j);  //正误判断：访问元素a[i][j]
    return 0;
}
```

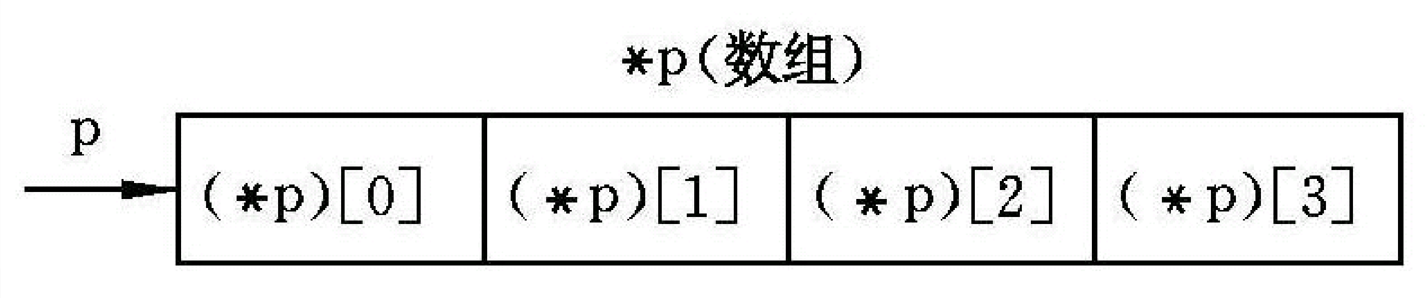
问题分析

- 从 p = a 开始
  - a 相当于指向a[3][4]的“第一个元素”的指针；
  - 所谓“第一个元素”是指一个“包含4个int型元素的一维数组”；
  - 所以，a相当于一个“包含4个int型元素的一维数组”的地址；
  - 因此，p的基类型应该是：

“包含4个int型元素的一维数组”

利用指针变量引用多维数组中的数组

- 问题
  - 如何定义一个指向“包含4个int型元素的一维数组”的指针变量?
- 解答
  - 变量定义语句：**int (*p)[4];**



程序填空&判断正误

- 输入 i, j; 输出a[i][j];

```
int main( )
{
    int a[3][4]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};
     int (*p)[4], i,  j;  // 填空：完成变量p的定义
    p = a;
    cin>>i>>j;              // i<3 代表行号，j<4代表列号
    cout<<setw(4)<<*(*(p+i)+j);  //正误判断：访问元素a[i][j]
    return 0;
}
```

利用指针变量引用多维数组中的数组

- *(*(p + i) + j) 是什么？
  - p指向一个“包含4个int型元素的一维数组”；
  - p + i 是第i+1个“包含4个int型元素的一维数组”的地址。
  - p + i 等价于 &a[i];
  - *(p + i) 等价于a[i];
  - *(p + i) + j 等价于 a[i] + j
  - 因为：a[i] + j 等价于 &a[i][j]
  - *(*(p + i) + j) 等价于 a[i][j]

利用指针变量引用多维数组中的数组

- 输入 i, j; 输出a[i][j];

```
main( )
{
   int a[3][4]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};
   int (*p)[4],  i,  j;
   p = a;
   cin>>i>>j;
   cout<<setw(4)<<p[i][j];
}
```

谢 谢 ！