**指针与多维数组**

1、实际上并不存在多维数组，所谓的多维数组本质上是用一维数组模拟的。

2、数组名是一个常量（意味着不允许对其进行赋值操作），其代表数组首元素的首地址。,

3、数组与指针的关系是因为数组下标操作符[]，比如，int a[3][2]相当于\*(\*(a+3)+2) 。

4、指针是一种变量，也具有类型，其占用内存空间大小和系统有关，一般32位系统下，sizeof(指针变量)=4。

5、指针可以进行加减算术运算，加减的基本单位是sizeof(指针所指向的数据类型)。

6、对数组的数组名进行取地址(&)操作，其类型为整个数组类型。

7、对数组的数组名进行sizeof运算符操作，其值为整个数组的大小(以字节为单位)。

8、数组作为函数形参时会退化为指针

9、int i； i=sizeof（a）/sizeof(a[0])；可以计算数组元素的个数。

**一、一维数组与数组指针**

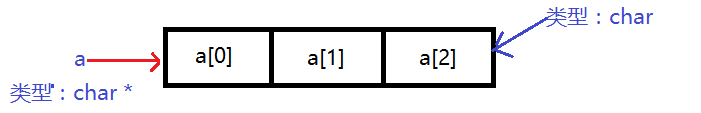
      假如有一维数组如下：

char a[3];

      该数组一共有3个元素，元素的类型为char，如果想定义一个指针指向该数组，也就是如果想把数组名a赋值给一个指针变量，那么该指针变量的类型应该是什么呢？前文说过，一个数组的数组名代表其首元素的首地址，也就是相当于&a[0]，而a[0]的类型为char，因此&a[0]类型为char \*，因此，可以定义如下的指针变量：

char \* p = a;//相当于char \* p = &a[0]

      以上文字可用如下内存模型图表示。



      大家都应该知道，a和&a[0]代表的都是数组首元素的首地址，而如果你将&a的值打印出来，会发现**该值也等于数组首元素的首地址**。请注意我这里的措辞，也就是说，&a虽然在数值上也等于数组首元素首地址的值，但是其类型并不是数组首元素首地址类型，也就是char \*p = &a是错误的。

      前文第6条常识已经说过，对数组名进行取地址操作，其类型为整个数组，因此，&a的类型是char (\*)[3]，所以正确的赋值方式如下:

char (\*p)[3] = &a;

**注：很多人对类似于a+1,&a+1,&a[0]+1,sizeof(a),sizeof(&a)等感到迷惑，其实只要搞清楚指针的类型就可以迎刃而解。比如在面对a+1和&a+1的区别时，由于a表示数组首元素首地址，其类型为char \*，因此a+1相当于数组首地址值+sizeof(char)；而&a的类型为char (\*)[3]，代表整个数组，因此&a+1相当于数组首地址值+sizeof(a)。（**sizeof(a)代表整个数组大小，前文第7条说明，但是无论数组大小如何，sizeof(&a)永远等于一个指针变量占用空间的大小，具体与系统平台有关**）**

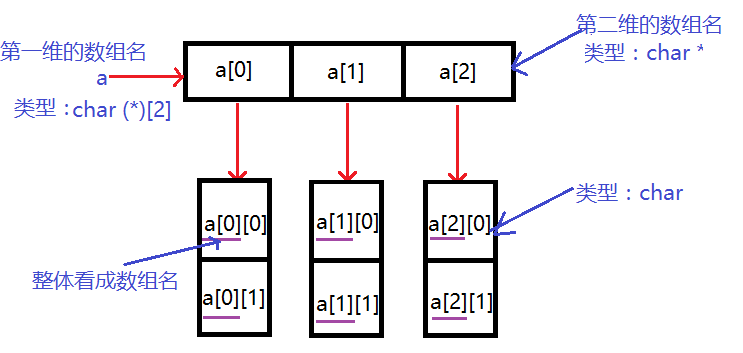
**二、二维数组与数组指针**

   假如有如下二维数组：

char a[3][2];

      由于实际上并不存在多维数组，因此，可以将a[3][2]看成是一个具有3个元素的一维数组，只是这三个元素分别又是一个一维数组。实际上，在内存中，该数组的确是按照一维数组的形式存储的，存储顺序为(低地址在前)：a[0][0]、a[0][1]、a[1][0]、a[1][1]、a[2][0]、a[2][1]。(此种方式也不是绝对，也有按列优先存储的模式)

      为了方便理解，我画了一张逻辑上的内存图，之所以说是逻辑上的，是因为该图只是便于理解，并不是数组在内存中实际的存储模型（实际模型为前文所述）。



      如上图所示，我们可以将数组分成两个维度来看，首先是第一维，将a[3][2]看成一个具有三个元素的一维数组，元素分别为：a[0]、a[1]、a[2]，其中，a[0]、a[1]、a[2]又分别是一个具有两个元素的一维数组(元素类型为char)。从第二个维度看，此处可以**将a[0]、a[1]、a[2]看成自己代表”第二维”数组的数组名，**以a[0]为例，a[0](数组名)代表的一维数组是一个具有两个char类型元素的数组，而a[0]是这个数组的数组名(代表数组首元素首地址)，因此a[0]类型为char \*，同理a[1]和a[2]类型都是char \*。而a是第一维数组的数组名，代表首元素首地址，而首元素是一个具有两个char类型元素的一维数组，因此a就是一个指向具有两个char类型元素数组的数组指针，也就是char(\*)[2]。

     也就是说，如下的赋值是正确的:

char (\*p)[2] = a;//a为第一维数组的数组名，类型为char (\*)[2]

char \* p = a[0];//a[0]维第二维uuju8数组的数组名，类型为char \*

      同样，对a取地址操作代表整个数组的首地址，类型为数组类型(请允许我暂且这么称呼)，也就是char (\*)[3][2]，所以如下赋值是正确的：

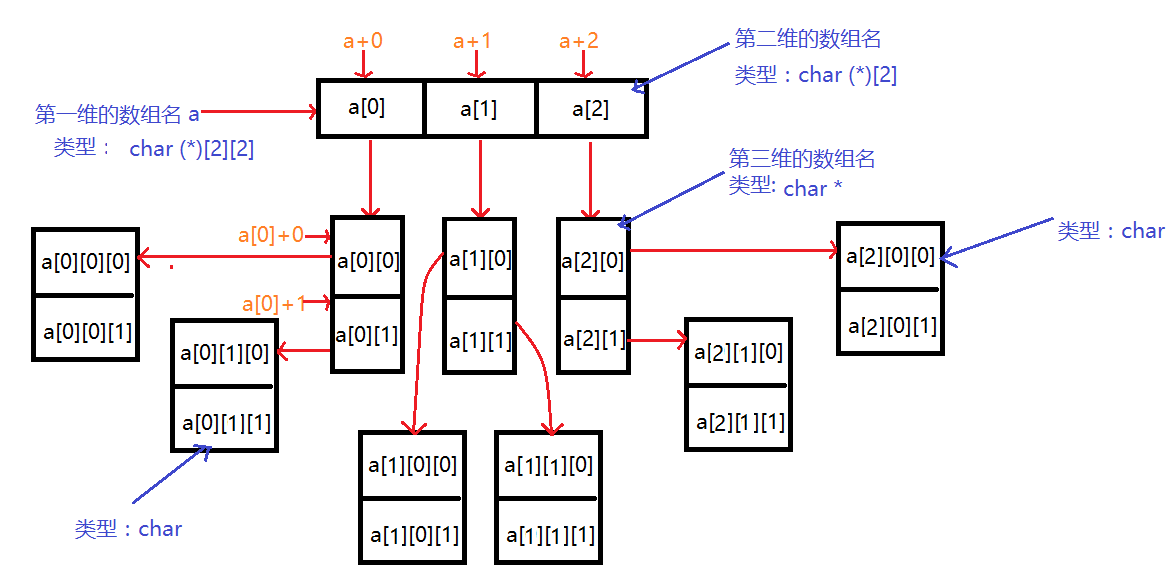
char (\*p)[3][2] = &a;

**三、三维数组与数组指针**

     假设有三维数组：

char a[3][2][2];

     同样，为了便于理解，特意画了如下的逻辑内存图。分析方法和二维数组类似，首先，从第一维角度看过去，a[3][2][2]是一个具有三个元素a[0]、a[1]、a[2]的一维数组，`只是这三个元素分别又是一个"二维"数组,a作为第一维数组的数组名，代表数组首元素的首地址，也就是一个指向一个二维数组的数组指针，其类型为char (\*)[2][2]。从第二维角度看过去，a[0]、a[1]、a[2]分别是第二维数组的数组名，代表第二维数组的首元素的首地址，也就是一个指向一维数组的数组指针，类型为char(\*)[2]；同理，从第三维角度看过去，a[0][0]、a[0][1]、a[1][0]、a[1][1]、a[2][0]、a[2][1]又分别是第三维数组的数组名，代表第三维数组的首元素的首地址，也就是一个指向char类型的指针，类型为char \*。



            由上可知，以下的赋值是正确的：

char (\*p)[3][2][2] = &a;//对数组名取地址类型为整个数组

char (\*p)[2][2] = a;

char (\*p) [2] = a[0];//或者a[1]、a[2]

char \*p = a[0][0];//或者a[0][1]、a[1][0]...

**二维数组的下标访问**

例1：

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

2 int (\*p)[5]; // 定义一个数组指针

3 p = a;

4

5 printf(" a[1][2] = %d.\n", a[1][2]); // a[1][2] = 8

6 printf("(\*(p+1)+1) = %d.\n", \*(\*(p+1)+2)); // a[1][2]

运行结果：

https://images2015.cnblogs.com/blog/646301/201510/646301-20151025223751677-2141925347.jpg

**四、关于二维数据必须要明白的几个符号**

例2：理解 a 、&a、 a[0]、 &a[0]、 a[0][0]、 &a[0][0]

[[复制代码](javascript:void(0);)](javascript:void(0);" \o "复制代码)

1 /\*

2 二维数组的几个符号的测试

3 1、a等同于&a[0]

4 2、a[0]等同于&a[0][0]

5 3、在数值上 a、&a、a[0]、&a[0]、&a[0][0] 是相等的，但是在类型上面是有区别的。

6 \*/

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

8

9 printf("a = %p.\n", a); // a 类型是 int (\*)[5]

10 printf("&a = %p.\n", &a); // &a 类型是 int a(\*)[2][5]

11 printf("a[0] = %p.\n", a[0]); // a[0] 类型是 int \*

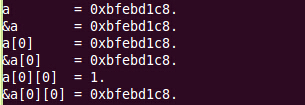
12 printf("&a[0] = %p.\n", &a[0]); // &a[0] 类型是 int (\*)[5]

13 printf("a[0][0] = %d.\n", a[0][0]); // a[0][0] 类型是 int

14 printf("&a[0][0] = %p.\n", &a[0][0]); // &a[0][0]类型是 int \*

[[复制代码](javascript:void(0);)](javascript:void(0);" \o "复制代码)

运行结果：



**例3：第一维和第二维的数组指针的使用**

[[复制代码](javascript:void(0);)](javascript:void(0);" \o "复制代码)

1 // 二维数组与指针的结合使用

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

3

4 int (\*p1)[5]; // 数组指针

5 int \*p2; // 一般指针

6 p1 = a; // 等同 p1 = &a[0]; // 指向二维数组的数组名

7 p2 = a[0]; // 等同 p2 = &a[0][0]; // 指向二维数组的第一维数组

8

9 printf("a[0][2] = %d.\n", \*(\*(p1+0)+2)); // a[0][2] = 3

10 printf("a[1][2] = %d.\n", \*(\*(p1+1)+2)); // a[1][2] = 8

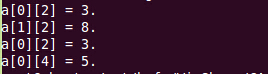
11

12 printf("a[0][2] = %d.\n", \*(p2+2)); // a[0][2] = 3

13 printf("a[0][4] = %d.\n", \*(p2+4)); // a[0][4] = 5

[[复制代码](javascript:void(0);)](javascript:void(0);" \o "复制代码)

运行结果：



 五．关于指针数组和数组指针的区别

指针数组：首先它是一个数组，数组的元素都是指针，数组占多少字节由数组本身决定，它是储存指针的数组的简称。

数组指针：首先他说是一个指针，它指向一个数组。在32位的系统下永远占用的4个字节，至于它指向的数组占多少字节，不知道。他是“指向数组的指针”的简称。

Int \*p1[10];

int (\*p2)[10];

这里首先要明白一个符号优先级的问题。

“[]”的优先级比“\*”高；p1先于“[]”结合，构成一个数组的定义，数组名为p1，int\*修饰的是数组的内容，即数组的每一个元素。因此这是一个数组，其包含10个指向指向int类型数据的指针，其包含10个指向int类数据的指针。即指针数组；

至于p2，“（）”的优先级比“【】”高，因此“\*”和p2构成了一个指针的定义，指针变量名是p2，int修饰的是数组的内容，即数组的每一个元素，数组在这里没有名字，是一个匿名数组。它指向的是一个包含10个int类型数据的数组，即数组指针。

