# 数组名的类型

数组名的类型就是数组，例如  
type a[3];  
a的数据类型就是type [3]。  
  
但数组名在使用时，一种情况是作为左值使用，另一种是作为右值。  
  
作为左值使用时，例如作为sizeof或&的操作数，数组名的类型就是数组。  
sizeof 得到的是数组占据内存空间的大小，& a得到的是指向数组的指针。  
  
但作为右值使用时，比如用来赋值，传参……数组名的类型实际上是一个指向数组起始元素的指针（标准的说法是这时编译器做隐式类型转换，有的书说成是退化，也有人直接主张数组名在作为左值和右值时的类型分别为两种）

# C 二维数组（指针）动态分配和释放

## 先明确下概念：

所谓32位处理器就是一次只能处理32位，也就是4个字节的数据，而64位处理器一次就能处理64位，即8个字节的数据。如果我们将总长128位的指令分别按照16位、32位、64位为单位进行编辑的话：旧的16位处理器，比如Intel 80286 CPU需要8个指令，32位的处理器需要4个指令，而64位处理器则只要两个指令，显然，在工作频率相同的情况下，64位处理器的处理速度会比16位、32位的更快。而且除了运算能力之外，与32位处理器相比，64位处理器的优势还体现在系统对内存的控制上。由于地址使用的是特殊的整数，而64位处理器的一个ALU（算术逻辑运算器）和寄存器可以处理更大的整数，也就是更大的地址。传统32位处理器的寻址空间最大为4GB（2的32次方 = 4294967296bit = 4G左右），使得很多需要大容量内存的数据处理程序在这时都会显得捉襟见肘，形成了运行效率的瓶颈。而64位的处理器在理论上则可以达到1800万个TB，1TB等于1024GB，1GB等于1024MB，所以64位的处理器能够彻底解决32位计算系统所遇到的瓶颈现象，速度快人一等，对于那些要求多处理器可扩展性、更大的可寻址内存、视频/音频/三维处理或较高计算准确性的应用程序而言，64位处理器可提供卓越的性能。

32位（bit）和64位（bit）系统的指针占的内存不一样，注意B与b不同，B是Byte（字节），b是bit（位） 1GB=1024MB,1MB=1024KB,1KB=1024B,1B=8bit

在32位的系统中，所有指针都占4 bytes。cpu决定内存的编址,如32位CPU有32条地址总线,对应的地址格式为 10 01 .... 01 01 = 32bit =4byte，32位的系统其寻址能力为32个二进制位，应该是4个字节的长度，指针大小是4byte.

64位 -> 01 01 10 10 ....01 = 64bit =8byte，64位的系统其寻址能力为64个二进制位，应该是8个字节的长度，所以指针大小是8byte。以下都是内容都是32位系统指针。

## (1)已知第二维

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char** (\*a)[N];    //指向数组的指针
3. a = (**char** (\*)[N])malloc(**sizeof**(**char** \*) \* m);
5. printf("%d\n", **sizeof**(a));   //4，指针
7. printf("%d\n", **sizeof**(a[0]));//N，一维数组
9. free(a);

## (2)已知第一维

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char**\* a[M];    //指针的数组
3. **int** i;
5. **for**(i=0; i<M; i++)
7. a[i] = (**char** \*)malloc(**sizeof**(**char**) \* n);
9. printf("%d\n", **sizeof**(a));   //4\*M，指针数组
11. printf("%d\n", **sizeof**(a[0]));//4，指针
13. **for**(i=0; i<M; i++)
15. free(a[i]);

## (3)已知第一维，一次分配内存(保证内存的连续性)

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char**\* a[M];    //指针的数组
3. **int** i;
5. a[0] = (**char** \*)malloc(**sizeof**(**char**) \* M \* n);
7. **for**(i=1; i<M; i++)
9. a[i] = a[i-1] + n;
11. printf("%d\n", **sizeof**(a));   //4\*M，指针数组
13. printf("%d\n", **sizeof**(a[0]));//4，指针
15. free(a[0]);

## (4)两维都未知

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

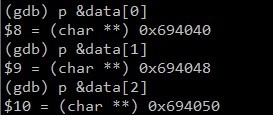
1. **char** \*\*a;
3. **int** i;
5. a = (**char** \*\*)malloc(**sizeof**(**char** \*) \* m);     //分配指针数组
7. **for**(i=0; i<m; i++)
9. {
11. a[i] = (**char** \*)malloc(**sizeof**(**char**) \* n); //分配每个指针所指向的数组
13. }
15. printf("%d\n", **sizeof**(a));    //4，指针
17. printf("%d\n", **sizeof**(a[0])); //4，指针
19. **for**(i=0; i<m; i++)
21. {
23. free(a[i]);
25. }
27. free(a);

## (5)两维都未知，一次分配内存(保证内存的连续性)

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char** \*\*a;
3. **int** i;
5. a = (**char** \*\*)malloc(**sizeof**(**char** \*) \* m);    //分配指针数组
7. a[0] = (**char** \*)malloc(**sizeof**(**char**) \* m \* n);//一次性分配所有空间
9. **for**(i=1; i<m; i++)
11. {
13. a[i] = a[i-1] + n;
15. }//采用如上内存分配方法，意味着将a的值初始化为m\*n的二维数组首地址,且这块内存连续
17. printf("%d\n", **sizeof**(a));    //4，指针
19. printf("%d\n", **sizeof**(a[0])); //4，指针
21. free(a[0]);
23. free(a);

### 采用（5）方法定义\*\*data，分配M\*256空间，调试如下： http://img.my.csdn.net/uploads/201301/09/1357697149_7242.jpg



换算为十进制分别为：

$1=140737353293840  $8  =6897728  
$2=140737353294096  $9  =6897736  
$3=140737353294352  $10=6897744

$1,$2,$3分别相差256

$8,$9,$10分别相差8（64位系统）

## **2. C++动态分配二维数组**2.C++动态分配二维数组

## (1)已知第二维

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char** (\*a)[N];    //指向数组的指针
3. a = **new** **char**[m][N];
5. printf("%d\n", **sizeof**(a));    //4，指针
7. printf("%d\n", **sizeof**(a[0])); //N，一维数组
9. **delete**[] a;

## (2)已知第一维

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char**\* a[M];    //指针的数组
3. **for**(**int** i=0; i<M; i++)
5. a[i] = **new** **char**[n];
7. printf("%d\n", **sizeof**(a));    //4\*M，指针数组
9. printf("%d\n", **sizeof**(a[0])); //4，指针
11. **for**(i=0; i<M; i++)
13. **delete**[] a[i];

## (3)已知第一维，一次分配内存(保证内存的连续性)

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char**\* a[M];    //指针的数组
3. a[0] = **new** **char**[M\*n];
5. **for**(**int** i=1; i<M; i++)
7. a[i] = a[i-1] + n;
9. printf("%d\n", **sizeof**(a));    //4\*M，指针数组
11. printf("%d\n", **sizeof**(a[0])); //4，指针
13. **delete**[] a[0];

## (4)两维都未知

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char** \*\*a;
3. a = **new** **char**\* [m];    //分配指针数组
5. **for**(**int** i=0; i<m; i++)
7. {
9. a[i] = **new** **char**[n];   //分配每个指针所指向的数组
11. }
13. printf("%d\n", **sizeof**(a));    //4，指针
15. printf("%d\n", **sizeof**(a[0])); //4，指针
17. **for**(i=0; i<m; i++)
19. **delete**[] a[i];
21. **delete**[] a;

## (5)两维都未知，一次分配内存(保证内存的连续性)

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **char** \*\*a;
3. a = **new** **char**\* [m];
5. a[0] = **new** **char**[m \* n];    //一次性分配所有空间
7. **for**(**int** i=1; i<m; i++)
9. {
11. a[i] = a[i-1] + n;         //分配每个指针所指向的数组
13. }
15. printf("%d\n", **sizeof**(a));    //4，指针
17. printf("%d\n", **sizeof**(a[0])); //4，指针
19. **delete**[] a[0];
21. **delete**[] a;

多说一句：new和delete要注意配对使用，即有多少个new就有多少个delete，这样才可以避免内存泄漏！

## 3. 静态二维数组作为函数参数传递

如果采用上述几种方法动态分配二维数组，那么将对应的数据类型作为函数参数就可以了。这里讨论静态二维数组作为函数参数传递，即按照以下的调用方式：

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **int** a[2][3];
3. func(a);

C语言中将静态二维数组作为参数传递比较麻烦，一般需要指明第二维的长度，如果不给定第二维长度，则只能先将其作为一维指针传递，然后利用二维数组的线性存储特性，在函数体内转化为对指定元素的访问。

首先写好测试代码，以验证参数传递的正确性：

## (1)给定第二维长度

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **void** func(**int** a[][N])
3. {
5. printf("%d\n", a[1][2]);
7. }

## (2)不给定第二维长度

**[cpp]** [view plain](http://blog.csdn.net/handsome_926/article/details/8233744) [copy](http://blog.csdn.net/handsome_926/article/details/8233744)

1. **void** func(**int**\* a)
3. {
5. printf("%d\n", a[1 \* N + 2]);//计算元素位置
7. }

注意：使用该函数时需要将二维数组首地址强制转换为一维指针，即func((int\*)a);注意：使用该函数时需要将二维数组首地址强制转换为一维指针，即func((int\*)a);