数组的特点：构造类型之一，连续存放

**一维数组**

定义

[存储类型] 数据类型 标识符 [下标]（下标的中括号是一个标识，可不能省略）

存储类型还是在存储类型专题里讲到的那些，auto，register，static，extern这些

数据类型，标识符也都跟前面一样

下标必须是整型的常量或整型的表达式

初始化（以int arr[3]为例）

如果不初始化，arr中的值为随机的，因为它的存储类型没有指定，默认为auto

如果要初始化，可以全部初始化，如int arr[3] = {1, 2, 3};（跟int arr[] = {1, 2, 3};效果是一样的），也可以部分初始化，如int arr[3] = {1};，用了部分初始化的话，该初始化的按正常情况初始化，其它元素为0

正常情况默认是auto，还有一种特殊情况，那就是static，如static int arr[3];，这时里面的元素会被自动初始化成0

初始化，int arr[3] = {1, 2, 3};，不初始化，int arr[3];，不论你是否初始化后面都不允许出现arr = {4, 5, 6};这种情况了，因为arr是数组名，表示当前地址的常量，常量不会出现在赋值左边

元素引用

数组名[下标]

数组名

数组名既是表示地址的常量，也是数组的起始位置，&arr与&arr[0]是相等的

数组越界

比如arr只有3个元素，没有arr[3]这个东西，但你可以arr[3] = 10;，并且可以打印arr [3];，这个arr[3]怎么找过来的呢，是由指针偏移找过来的，arr[i] = \*(arr + i)，数组越界这件事，没有完美的解决办法，只能靠程序员的经验和写程序的敏感程度和细心程度来人为地检查和判断，不能够依赖当前的寄存器来做这项工作，因为arr[i] = \*(arr + i)这个语法到哪都合法

**二维数组**

定义

[存储类型] 数据类型 标识符 [行下标][列下标]

初始化（以arr[2][3]为例）

如果不初始化，如int arr[3];，那里面的值就是随机的，如果全初始化，如int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};，那就按常规的来，如果部分初始化，如int arr[2][3] = {{1, 2, 3}};或int arr[2][3] = {{1, 2}, {4}};，那么剩下的没有初始化到的自动补0

还有一种初始化方式，就是只用一个大括号，如int arr[2][3] = {1, 3, 4, 6, 7}，那就按顺序来，前面5个都正常，最后一个arr[1][2]就为0了

对于二维数组来说，初始化时，行号可以省略，列号不可以

元素引用

数组名[行标][列标]

存储形式

顺序存储，按行存储

深入理解二维数组

对于一维数组a[3] = {1, 2, 3};，来说，a是一个列指针，a指向的是1，a+1指向的是2

对于二维数组a[2][3] = {{1, 2, 3}, {4, 5, 6}};，来说，a是一个行指针，a指向的是{1, 2, 3}，a+1指向的是{4, 5, 6}

**字符数组**

定义

[存储类型] 数据类型 标识符 [下标]…

初始化

单个字符初始化，如char str[3] = {‘a’, ‘b’, ‘c’};

用字符串常量初始化，如char str[3] = “ab”;（“ab”这个串带\0哦）

char str[32];

str = “hello”;，是错误的，因为str是数组名，是一个常量（注意，这里指的是str是常量，而不是hello哦，一定要注意，切记），区别于上面说的，那里是初始化，这里是赋值

存储特点

输入输出

常用函数（要包含头文件string.h）

strlen

描述的是当前这个串所占的长度，以尾0作为结束标记，不包含尾0，尾0后面的东西都看不到了

sizeof

计算出来的是某一个传参传进来的内容在内存中真正所占的字节数

如，char str[] = “hello”;，你用strlen(str);，结果就是5，你用sizeof(str);，结果就是6

如，char str[] = “hello\0abc”， 你用strlen(str);，结果还是5，你用sizeof(str);，结果就是10

strcpy

函数原型，char \*strcpy(char \*dest, const char \*src);，意思是要把src所指向的内容，从起始地址开始一点点往外拷贝，拷贝到dest所指向的位置，以src所指向的字符串的尾0作为结束，尾0也会被拷贝过去哦，该函数的返回值是dest所指向的起始位置。使用该函数首先得保证这两个串不会重叠，并且保证你的dest指向的串足够长，长到能够容纳要拷贝过来的src所指向的串，但即使你容纳不下，你也能拷过去，不过这时已经犯了越界错误

strncpy

函数原型，char \*strncpy(char \*dest, const char \*src, size\_t n);，原理与strcpy差不多，只不过多了一个n，意思就是我最多从src中拷贝n个字节到dest中去（如果src中不足n个，那就取到尾0为止），一般情况下，我们习惯把n值设置成和dest总的容量一样大，这样就能确保从src中拿n-1个字节到dest中去（因为n里面还得包含一个尾0呢），这样就防止了越界现象

strcat

函数原型，char \*strcat(char \*dest, const char \*src);，意思是我要往dest这块空间连上src，如果dest这块空间没内容，那就算了，如果有内容，那么会把dest里的尾0拿掉（说覆盖掉也行），然后把src里的东西拿过来，最后再补上一个尾0。这个函数是追加函数，使用该函数首先得保证这两个串不会重叠，并且保证你的dest指向的串足够长，长到能够容纳下追加之后结果，但即使你容纳不下，你也能追加过去，不过这时已经犯了越界错误

strncat

函数原型，char \*strncat(char \*dest, const char \*src, size\_t n);，原理与strcat差不多，只不过多了一个n，意思就是我从src中取n个字节追加到dest后面去

strcmp

函数原型，int strcmp(const char \*s1，const char \*s2);，比较s1和s2两个字符串（比的是ascii码），返回一个整型值，如果相同，返回0，如果s1>s2，返回正值，如果s1<s2，返回负值

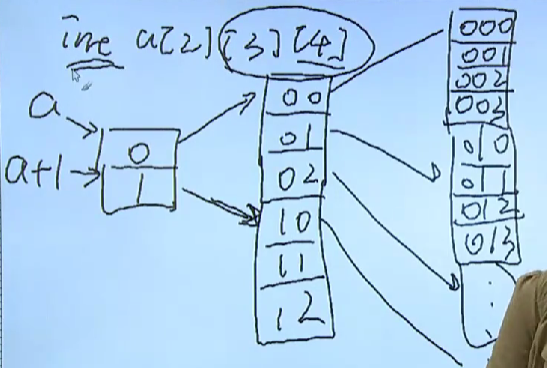
strncmp

函数原型，int strncmp(const char \*s1，const char \*s2, size\_t n);，意思是我只比较两个字符串的前n位

以上都是干一件事情有两套函数，为什么呢，因为你当前无法保证目标串足够大，为什么无法保证呢，指针和函数部分再讲

**多维数组**

以int a[2][3][4];为例



一维数组a[2]中，a+1所指向的位置，就是从a所指向的位置往后跳1个字节

二维数组a[2][3]中，a+1所指向的位置，就是从a所指向的位置往后跳3个字节

三维数组a[2][3][4]中，a+1所指向的位置，就是从a所指向的位置往后跳3\*4=12个字节