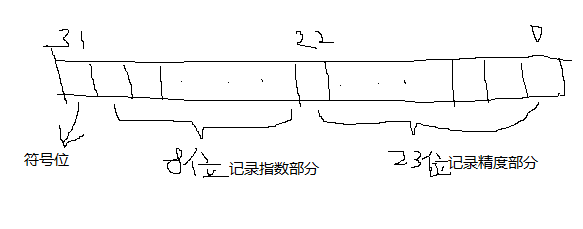
**数据类型**



整型数的存储都是以补码的形式来存储的，正数的补码就是它的二进制的本身，负数的补码是它绝对值的数的二进制形式取反再加1

float类型存储方式：比如0.000789，它是这样存的，0.789\* 10^-3，0.789是精度部分，-3是指数部分，在存储空间中也是这样来存的，如下图



double类型比float多出来的位数全部用于扩充存储精度部分，别的都一样

字符类型看ascall表

NULL表示空字符，十进制的ascii码为000，字符串以一个尾0作为结束标记，尾0就是ascii码为0的这样一个特殊字符

回车的ascii码十进制表示是010

0的ascii码十进制表示是048，可以累加推到9

A的ascii码十进制表示是065

a的ascii码十进制表示是097

Z和a之间不是连续的哦

char型的存储方式和整形一样，都是以补码的形式来存储的，可以把它当成一个短小精悍的整型数来用，比如，char a=64;printf(“%d”,a)，我就是不用%c，而用%d是可以的

看着表里好像写了char型可以规定有无符号，但实际上标准C当中对char型到底是有符号还是无符号居然是一个未定义行为

不同类型的数据间进行转换，分为隐式和显式（强制类型转换）

隐式：不同的数据类型混合运算时，实际上是往所占字节数多的类型靠拢

显式

float本身不是一个特别精确的表示，比如0.999999和1.000000001都约等于1，没办法去与一个精确的值做对比，比如f是一个float数据，不能f == 0，要想判断它是否和0相等，可以用|f-0| <= 10^-6，也就是fabs(f-0) <= 1e-6

不同形式的0值：0 ‘0’（ascii） “0” ‘\0’ NULL（就相当于‘\0’）

数据类型与后续代码中所使用的输入输出要相匹配，防止自相矛盾，比如

%d是把当前某一个值按照有符号的十进制输出，如果你原先定义的数是unsigned，结果输出时用了%d，那原先的接近最大表示范围的数在输出时就有精度丢失，变成负的了

**常量与变量**

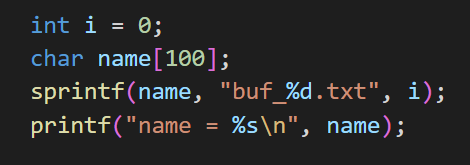
常量：在程序执行过程中值不会发生变量的量，一定不会出现在赋值左边，有整型常量，实型常量，字符常量，字符串常量，标识常量

整型常量：1，790，76，52，保存时需要找整型变量

实型常量：3.14，2.26，1.9999，保存时需要找实型变量

字符常量：由单引号引起来的单个字符或转义字符，如‘a’，’A‘，’\n’，‘\t’，像 ‘\015’和‘\x7f’也属于转义字符，前一个是八进制的，后一个是十六进制的

字符串常量：由双引号引起来的一个或多个字符组成的序列，如“”（空串也属于字符串，以尾0为结束标志），“a”，“abXYZ”，”abc\n021\018”，保存时由于不存在字符串变量类型，所以需要借助构造类型来保存，也就是数组



标识常量：#define，处理在程序的预处理阶段，占编译时间，特点是一改全改，缺点是不检查语法，只是单纯的宏体与宏名之间的替换

宏替换是发生在预处理阶段，如果把时间笼统的划分成编译时间和运行时间，那宏所占用的时间是编译时间，函数不一样，函数是要在调用的位置对当前的执行现场做一个压栈保存，然后去到你所指定的一个函数去执行，也就是跳往另外一个入口地址，然后回来弹栈恢复刚才现场的一个过程，这个过程占用的是你的运行时间，那怎么来区分什么时候该用宏什么时候该用函数呢，简单来讲，宏相对于完成同样功能的函数来讲是比较危险的（具体例子回看视频，不好解释），但它比较节省你的运行时间，比如你在写内核的某个模块，我在内核中节省了零点多秒的时间，就能够为用户层面或者应用层来节省多少时间，那你就用宏来写，你看内核链表，都是各种各样的宏，没有什么真正意义上的函数，什么时候用函数呢，你在写应用层的内容的时候，或者是在写应用层面上的系统开发的时候，那么多用函数，因为在写应用层内容的时候，更加要求的是稳定，对于实时性来讲要求就比较松散一些，但内核不一样，内核中多消耗一秒钟的时间，反应到你的用户体验上来讲就不是一秒钟了，所以说，讲究效率，节约一点一滴的时间可以用宏，其它情况要求稳定，可以用函数

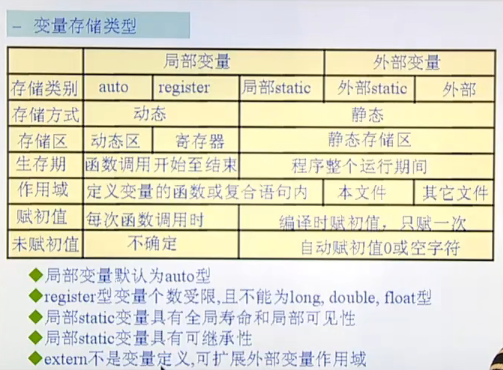
变量：用来保存一些特定内容，并且在程序执行过程中值随时会发生变化的量，不能重复定义哦

[存储类型] 数据类型 标识符 = 值，也就是TYPE NAME = VALUE;

标识符对于程序员的含义：给某一块空间取了一个名字来命名这块空间或者是整个这段内存的起始位置，但是你的编译器未必把它这样称呼，所以你的标识符其实是给人看的，在你定义标识符的时候尽量做到见名生义

数据类型：可以是上面讲的基本数据类型，也可以是构造类型

值：注意匹配，比如用浮点类型的常量来给一个整型的变量赋值的话，肯定有精度丢失



存储类型：auto、static、register、extern（不属于定义型的关键字，属于说明型的关键字）

auto：默认型，一般存储类型可以省略，如果省略了，那就默认是用的auto，特点是自动分配空间，自动回收空间，是在内存的栈上分配的哦，用auto来定义变量时，如果不对它进行初始化，那这个变量的值应该是随机的，但高版本的gcc会有优化，给它初始化为0

register：建议型，是寄存器类型，不做过高要求，因为寄存器存在于CPU中，速度快，但数量吃紧，是一个比较抢手的资源，它能随便分配给你吗

什么时候会用到呢，比如说rigister int i = 1；我这个i的值在程序执行过程中要不停地用，要连续地使用超过10W次，那我就认为，这个变量要反复地去内存中取数做计算再放回去，非常地消耗时间和资源，那我就申请或者建议把i这个值的存储空间放到寄存器当中，这样就方便取，建议谁呢，建议编译器，当然只是建议，至于你i的存储空间是不是放到了寄存器里去了，由你的gcc（gcc就是编译器吧）来确定，而gcc是不会轻易地把你所申请放到寄存器当中的变量放到寄存器当中的，原因就是寄存器是一个非常宝贵的资源，要想用，必须满足非常苛刻的条件，一般来说是这几点：只能定义局部变量，不能定义全局变量；大小有限制，只能定义32位大小的数据类型（当前环境为32位），如double（64位）就不可以。而且寄存器中没有地址，所以一个寄存器类型的变量无法打印出地址查看或者使用

static：静态型，会自动初始化为0值或者空值，只被定义一次，并且这种类型的值有继承性（对这种变量做运算，结果会累积起来），另外，常用于修饰变量或函数（非常常用哦），用了static修饰的变量自始至终只有一块空间，生命周期也是从定义开始直到程序结束，而且只能在当前范围内（当前的.c文件）用，跑了文件引用不到，用来修饰多个.c文件的全局变量的时候特别有用，你如果各个存在引用关系的.c文件同名的全局变量不加static，那就犯了变量重复定义的错了，所以说一般情况下，在使用全局变量时，会习惯性加上static，防止跟别的.c当中同名的内容冲突； static修饰函数时，它作用是防止这个函数对外扩展，也就是说除了当前.c能用这个函数，其它位置谁也用不到，要想用可以在当前.c再定义一个不加static的专门用于调用该静态函数的函数，这个函数能被外面调用，相当于对外提供了一个接口，但我就是不想让你直接看到这个被定义为静态函数的函数

extern：说明型（前面三种属于定义型），说明我有一个定义，但是不在这，去别的位置找，既然是说明型那就意味着不能改变被说明的变量的值或类型，它可不是用来做变量定义的，而是用于扩展外部变量的作用域，比如一个全局变量的作用范围只会在当前.c文件，如果你在别的.c文件对该全局变量加了extern，那它在这个文件中也能用，相当于它的作用范围被延长了

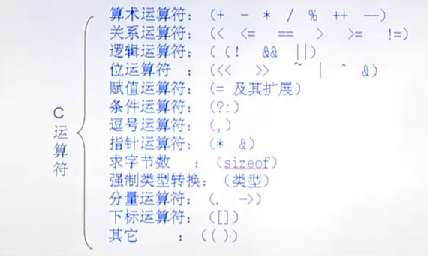
全局变量和局部变量：

全局变量和局部变量的作用范围均是从定义开始（不是整个代码哦，是从定义开始）到当前程序结束，生命周期亦是如此，而且一定是内部的作用范围屏蔽外部的作用范围

全局变量慎用，它存在使用缺陷，任何位置对当前全局变量的改变都会影响其它模块的使用

**运算符和表达式**

不加分号就是表达式，加一分号就是语句

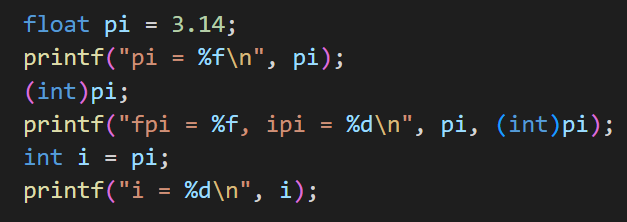


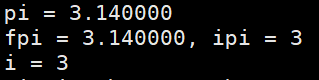
%运算要求两边的操作数必须是整型

逗号运算符这个表达式的值，就是整个逗号运算符最后一项的值

sizeof，很多人认为它是一个关键字，其实不是，它是一个运算符，它用来测试或者反映出来你当前指针的变量或者是类型在目前环境下真正所占的字节数，一个字节占8位哦。这个还能嵌入到一些动态内存的管理，比如说，我的程序运行到某一个位置，我才能够确定要多大的字节数，这个时候我们可以临时地用sizeof来计算一下，然后再malloc一个这样的内存来用

强转只是一种中间过程，对于你转之前的那个内容的类型啊，值啊等都没有任何影响，程序中使用强转是一种很丑的做法





位运算在当前嵌入式开发尤其重要，因为位运算经常用来操作硬件，比如直接针对某一位进行处理，或者说来处理缓冲以及底层对图像的一些处理，这些基本都是用位运算来搞定的。而且在嵌入式开发当中，位运算有非常重要的意义，嵌入式板子上的资源是有限的，在分配状态位的时候，由于状态数量有限，所以完全不会用int来定义，太浪费了，比如flag只有0和1两种状态，这时用一个二进制的位就能表示它了，如果有四种状态，用两个二进制位也完全能够表示它

位运算中的~是按位取反的意思，就是把对应变量的二进制形式，0变1，1变0

位运算中的|是按位或的意思

位运算中的&是按位与的意思

位运算中的^是按位异或的意思

将操作数中第n位置1，其他位不变：num = num | 1 << n;

将操作数中第n位置0，其他位不变：num = num & !(1 << n);

测试第n位：num & 1 << n

从一个指定宽度的数中取出其中的某几位（比如从m到n，m < n）：？？