**结构体**

产生和意义

数组只能将同一类型的数据存放在一块连续的空间，结构体可以把多种不同类型的数据存放在同样的一块连续的空间

类型描述

struct 结构体名

{

数据类型 成员1;

数据类型 成员2;

……

};

还有匿名的结构体定义，就是没有结构体名，一般把它叫做无头结构体

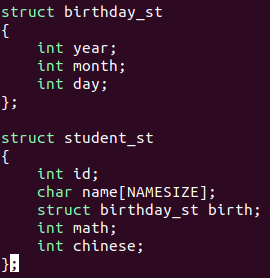
结构体的类型描述是不占用任何存储空间的，所以不要企图在说完，数据类型 成员1，之后就直接用一个等号给它初始化，不占用空间也就意味着你给它的值，它没有办法去放。类型描述只是在描述当前这个结构体里面有哪些成员，这些成员的类型是什么

举个例子，int和int i，int i大家非常熟悉，它就是定义了一个int类型的变量，如果用面向对象的话来说，那就是初始化了一个整型的实例，谁占空间啊，不能说int占空间，而是i占空间，i占多大空间，和它定义的数据类型是有关系的，结构体也一样，当前你所看到的结构体只是一个描述

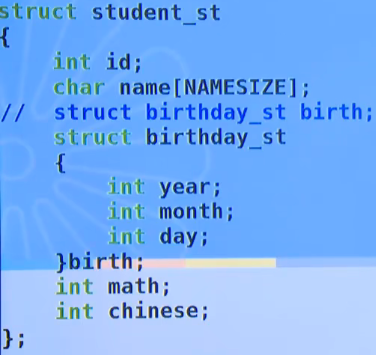
建议结构体定义在函数外，放在整个程序的声明位置，就没啥太大问题了

结构体嵌套

可以这么写



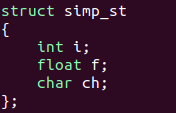
也可以这么写

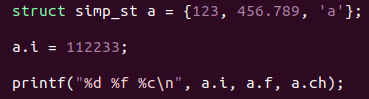


定义变量（变量，数组，指针），初始化及成员引用

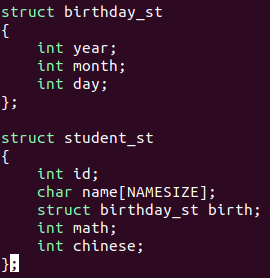
定义变量及

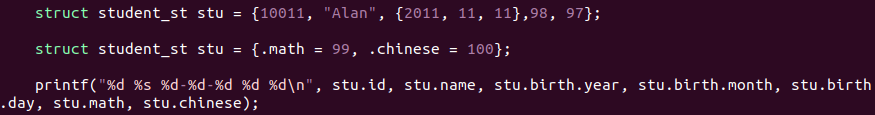
成员引用1：变量名.成员名





可以完整初始化，也可以用.math和.chinese这种形式，来对结构体部分初始化，其它没被初始化的都自动赋0





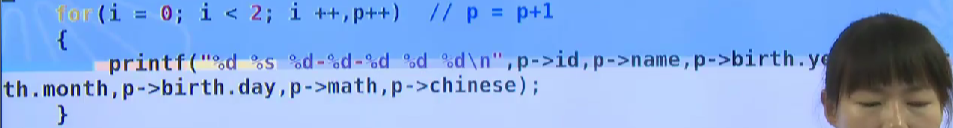
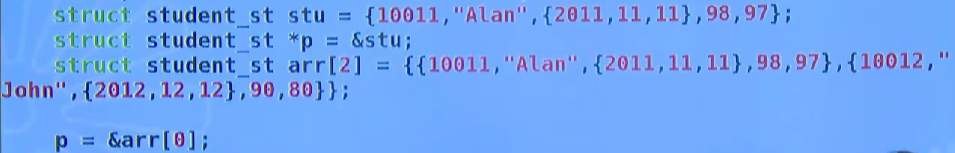
通过结构体名字引用到它内部所有成员是不可能的，它只是表示当前这块连续空间的起始位置是多少，而跟里面的元素没有其它的关联

定义指针及

成员引用2：指针->成员名，特殊的，（\*指针）.成员名



定义数组

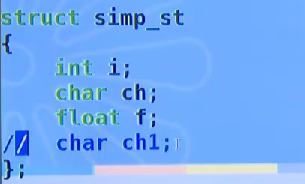


p是结构体指针变量，指向一个结构体，p+1的偏移量就是一个结构体在内存中所占的字节大小

占用内存空间大小

结构体当中的各个成员是由你描述的顺序在内存当中来进行顺序存放，所以如果我知道结构体的起始位置在哪，我又知道每一个成员的数据类型，其实我就可以向下移动，顺着地址向下偏移，来找到每一个成员

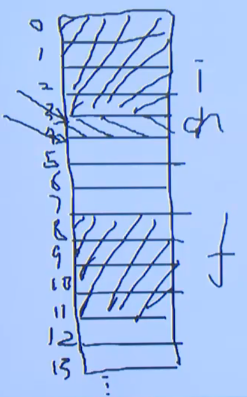
简单粗暴地理解结构体对齐，以int为准：



struct simp\_st a;

sizeof(a)的结果是12

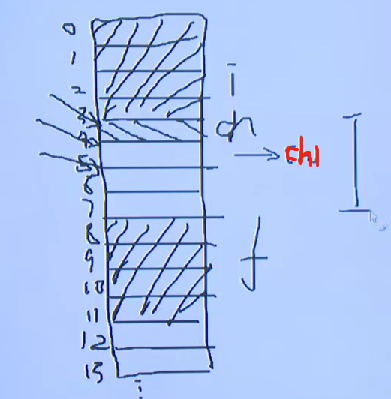
首先看你打算将变量存储到某个位置的地址能不能被sizeof(你要存储的变量类型)整除，也就是判断addr%sizeof是否为0，如果是，那当前变量就存放到这，如果不是，那你的addr就要继续向下+1，直到满足该表达式为0



假设地址是从0（实际上是0x…，这里简写）开始的，并且0能够被任何数整除，0%sizeof(int)==0成立，所以i被存储在0-3这个位置上，下一个值就要从存储房间号4开始存，4%sizeof(char)==0成立，所以ch就被存储在4这个位置上，再往下房间号为5开始，我就要用5%sizeof(float)==0，这显然不成立，所以arr+1，6也不行，继续+1，7不行，继续+1，8可以，8%sizeof(float)==0成立，所以我们把5-7跳过去了，从8开始存这个float类型，所以f被存储在8-11这个位置上

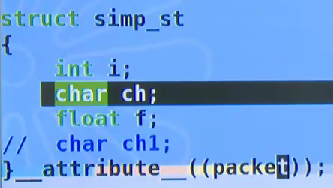
所以当我们把程序中的//打开，sizeof(a)的结果会是16，因为这个ch1所占的空间就是房间号12这块，而后起始位置变成13，你存什么东西都需要偏移，无形当中你最后还要偏移3个字节，所以你sizeof(a)的大小应该是对齐之后的数值

如果把//后面的ch1放在ch后面并且打开//的话，sizeof(a)的结果还会是12，这是因为，在存储ch1时，是从起始地址号5开始存ch1，5%sizof(char)==0成立，所以ch1被存储到5这个位置上，6-7存不了float类型，到8才行，f依然存储在8-11这个位置上



所以我们可以推测出来，中间这块空间最多可以放4个char型

在一些网络传输的环境当中，不同的平台上对齐的方式其实是略有差异的，那我怎么确定，我的数据到你那边你是当数据而不是当成填空白的，然后给我跳过去了，在网络传输，比如socket套接字的实现上就需要注意这些问题，我们的结构体类型的数据在传给对端目标设备的时候一定不能做对齐，因为你也不知道到底它是几个字节一对齐的，这跟你当前机器的环境是有关系的，所以有的时候，如果我们有需要的话，还需要告诉给编译器，我当前用到的这个结构体一定不能对齐，比如



现在

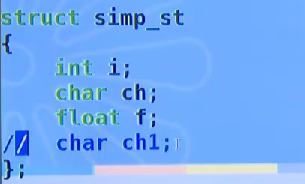
struct simp\_st a;

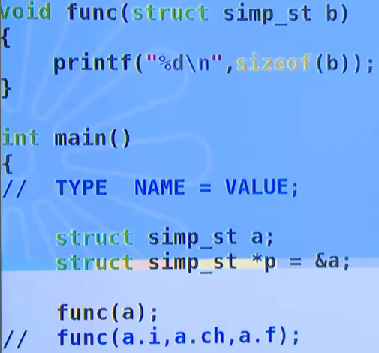
sizeof(a)的结果就是9了

函数传参

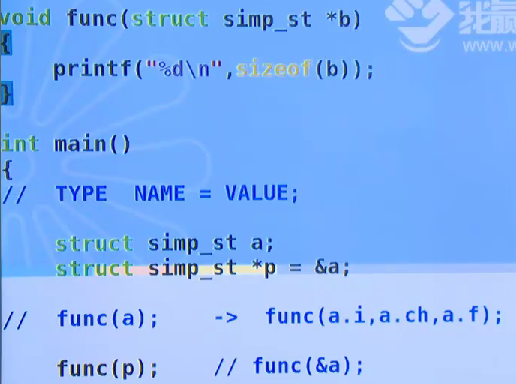
两种方式一种是值传参，一种是地址传参

值传参





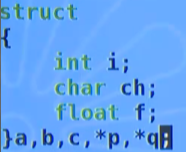
打印的是12，用值传参，把原先结构体里有的东西完整地拷贝过去，func(a)传参相当于func(a.i, a.ch, a.f)这种传参形式，是多值传递，会生成一个临时的跟原结构体一模一样的结构体，传参过程开销非常大，所以结构体值传参一般不用，更多的用的是地址传参



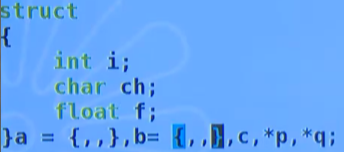
这时的开销就是当前环境下，一个指针的开销

如果是值传参，那你通过b来引用成员时用的是.，用b.来.出来，如果是地址传递，对方也是用指针接收的，那就用->来引用成员，没什么难的，只要你区分你当前用的是哪种传参方式，讲值传递和地址传递的主要目的是让你看到它们的开销

还有匿名的结构体定义，就是没有结构体名，一般把它叫做无头结构体，像这种结构体只能在声明的同时把变量定义好，比如



也可以在定义的时候初始化



使用起来与前面讲的没有差别，但有一点，你需要一次性把你将要用的全部都定义好，起码定义好，有了名字才能去赋值，引用

**共用体**

结构体到底占多大，跟它有多少成员还有成员的大小有关系，共用体不一样，共用体是共用某一块空间

产生及意义

共用体存在于，同时有多种情况存在，但实际上在某一个时刻只可能是一种情况存在，比如你约车，可以选上午也可以选下午，等你去坐车时只能选上午或下午，不能同时选，而结构体不一样，它可以多个情况共存，比如让你选你的兴趣爱好，可以同时选唱歌，跳舞，打篮球

类型描述

union 共用体名

{

数据类型 成员1;

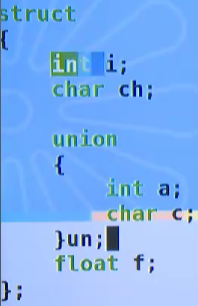
数据类型 成员2;

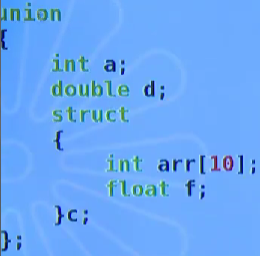
……

};

嵌套定义

跟结构体差不多，其实共用体可以嵌套结构体，结构体也可以嵌套共用体





大端存储：数据的低位保存在高地址当中

小端存储：数据的低位保存在低地址当中

常用的x86都是小端，c51是非常典型的大端，很arm的体系结构或者是dsp的都是小端

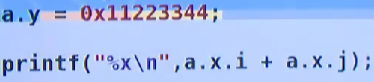
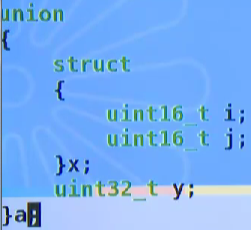
我有一个32位的无符号的数，我想知道高16位和低16位相加的和是多少

可以用位运算来实现



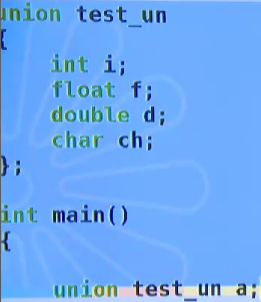
用uint32\_t，要包含头文件<stdint.h>

还可以用构造类型来实现

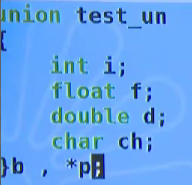


共用体的大小是4个字节，x和y共用这块空间，且不会共存，底下对a中的y赋值，高16位是1122，低16位是3344，用的时候用a中的x，依然是那块空间，i是占了低16位，j占了高16位

定义变量（变量，数组，指针），初始化及成员引用



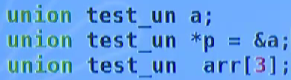
可以像上面那样定义，也可以跟结构体一样把变量，指针等放上去，如下



和结构体一样，共用体也可省略名字，成为无头共用体，但还是一样，你需要一次性把你将要用的全部都定义好，因为你以后再也找不到这个共用体的名字叫什么

其实定义，初始化和成员引用都跟结构体差不多





占用内存大小

共用体是多个成员共用同一块空间，所占空间大小为共用体内占空间最大的那个变量所占空间大小

函数传参（值，地址）

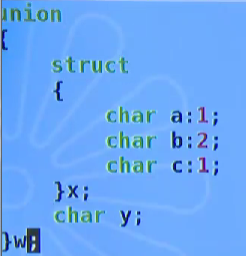
共用体看着并不大，为什么还需要地址传递呢，直接传值不就好了

答：共用体可以嵌套结构体啊，结构体可以很大，然后共用体所占空间又是根据它里面最大的变量所占的空间来分配的，这就导致共用体也很大，值传递开销也小不了，所以还是用地址传参更省开销

位域

在开发上没有实际意义，但面试可能会问

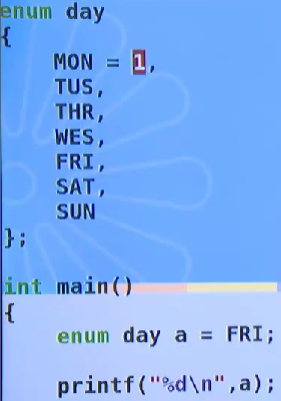
实际上也是共用体的一种形式，但是它来存放变量时就不以字节为单位了，而是以位为单位



a占1位，b占2位，c占1位，y占1个字节8位

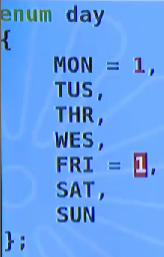
**枚举**

可以看成集合，一个事物总共有多少种情况，比如一年有12个月



第一个MON等于1了，底下的都会自动递增，所以，会打印出5

如果第一个不赋值，那就会自动赋0，底下的依然会自动递增，会打印出4



如果中间又赋了一个1，那从它往下重新递增计数

更常用的做法，当成有值的宏来使用

