C语言辅导:不定参数在C语言中的应用实例计算机二级考试
 2009年下半年全国计算机等级考试你准备好了没?考计算机等级二级考试的朋友,2009年下半年全国计算机等级考试时间是2009年9月19日至23日。更多优质资料尽在百考试题论坛 百考试题在线题库 不定参数在C语言中的应用实例：不定参数当年做为C/C 语言一个特长被很多人推崇，但是实际上这种技术并没有应用很多。除了格式化输出之外，我实在没看到多少应用。主要原因是这种技术比较麻烦，副作用也比较多，而一般情况下重载函数也足以替换它。尽管如此，既然大家对它比较感兴趣，我就简单总结一下它的使用和需要注意的常见问题。 刚学C语言的时候，一般人都会首先接触printf函数。通过这个函数，你可以打印不定个数的变量到屏幕，如：printf（"%d"，3）；printf（"%d，%d"，3，4）；上述代码看似简单，实际上却需要我们解决许多问题。在我们设计printf的时候，我们是不知道到底会传入几个参数的。在这种未知的情况下，我们需要解决下面几个问题：怎么告诉printf我们会传入几个参数 printf怎么去访问这些参数 函数调用完成后，系统怎么把参数从传递用的堆栈中释放 为了解决这些问题，我们首先要解释cdecl调用约定，所有使用不定参数的函数必须是使用cdecl（全局函数）或者this call（类成员函数）调用约定。该约定对于参数传递规定如下：参数从右向左入栈（也就是如果你调用f（a，b，c），则c先入栈，然后是b，最后是a入栈） 调用者负责清理堆栈 其中第二点直接解决了前面三个问

题中的第三个问题。我们来详细说说其他两个问题。 确定参数的个数 在一个函数中，一般有如下prelog代码： 00401020 push ebp 00401021 mov ebp，esp 00401023 sub esp，48h 执行上述代码之后，func（a，b，c）函数所处的堆栈上下文就变成如下布局：
http://img.ddvip.com/2009_04_08/1239161235_ddvip_4373.jpeg 其中，ebp指向保存旧的ebp的堆栈内存的下一个字的地址，ebp＋8指向eip地址，ebp 12则指向函数调用的第一个参数，而ebp和esp之间是用于临时变量（也就是堆栈变量）的空间。 注意，由于上述prelog代码的存在，我们很容易通过ebp得到第一个参数的地址，对于不定参数列表之前的类型固定的参数，我们也可以根据类型信息得到其实际的位置（例如，第一个参数的位置偏移第一个参数的大小，就是第二个参数的地址）。 注意不定参数函数有个限制，就是不定参数的列表必须在整个函数的参数列表的最后。我们不可以定义如下的函数： void func（int a， ……， int c） 所有类型固定的参数都必须出现在参数列表的开始。这样根据前面的论述，我们就可以得到所有类型固定的参数。 在设计具有不定参数列表的函数的时候，我们有两种方法来确定到底多少参数会被传递进来。 方法1是在类型固定的参数中指明后面有多少个参数以及他们的类型。printf就是采用的这种方法，它的format参数指明后面每个参数的类型。 方法2是指定一个结束参数。这种情况一般是不定参数拥有同样的类型，我们可以指定一个特定的值来表示参数列表结束。下面这个sum函数就是一个例子： int sumi(int c, ...) { va_list ap. va_start(ap,c). int i. int sum = c. c = va_arg(ap,int). while(0!=c) { sum = sum c. c =

va_arg(ap,int). } return sum. } 使用这个函数的代码为： int main(int argc, char* argv[]) { int i=sumi(1,2,3,4,5,6,7,8,9,0). return 0. } 访问各个参数 其实前文已经告诉我们怎么去访问不定参数。va_start和va_arg函数可以被结合起来用于依次访问每个函数，他们实际上都是宏函数。 在vc6，va_start函数定义为： #define _INTSIZEOF(n) ((sizeof(n) sizeof(int) - 1) amp.v _INTSIZEOF(v)) 其中_INTSIZEOF（n）计算比n大的sizeof（int）的最小倍数，如果n=101，则_INTSIZEOF（n）为104. va_start执行完毕后，ap指向变量v后第一个4字节对齐的地址。例如，v的地址为0x123456， v的大小为13，则v后面的下一个与字边界对齐的地址为0x123456 0x0D=0x123463再调整为与4字节对齐的下一个地址，也就是0x123464. va_arg函数定义为： #define va_arg(ap,t) (*(t *)((ap = _INTSIZEOF(t)) - _INTSIZEOF(t))) 分析与va_start一样，它的结果是使ap指向当前变量的下一个变量。 这样，我们只要在开始时使用va_start把不定参数列表赋值给ap，然后依次用va_arg获得不同参数即可。 潜在问题 使用不定参数列表，有两个问题特别需要注意。 问题1的理解相对简单：我们在重载一个函数的时候，不能依赖不定参数列表部分对函数进行区分。 假定我们定义两个重载函数如下： int func（int a， int b， ……） int func（int a， int b， float c） ；则上述函数会导致编译器不知道怎么去解释func（1，2， 3.3），因为当第三个参数为浮点数时，两个实现都可以满足匹配要求。一般情况，个人建议对于不定参数函数不要去做重载。 另外一个问题是关于类型问题。绝大多数情况下，C和C＋＋的变量都是强类型的，而不定参数列表属于一个特例。 当我们调用va_arg的时候，我们指明下

一个参数的类型，而在执行的时候，va_arg正是根据这个信息在堆栈上来找到对应的参数的。如果我们需要的类型和真实传递进来的参数完全一致时自然没有问题，但是假如类型不一样，则会有大麻烦。 假如上面的的sumi函数，我们用下面方法调用： int sum = sumi(1, 2.2, 3, 0) 注意第二个参数我们传入了一个double类型的2.2，我们希望sumi在做加法时可以做隐式类型转换，转换为int进行计算。但是实际情况时，当我们分析到这个参数时，调用的是： c=va_arg(ap,int) 据前文va_arg的定义，这个宏被翻译成： #define va_arg（ap，t）（*（int *）（（ap = _INTSIZEOF（int）） - _INTSIZEOF（int）） ） 如果后面的 =计算出正确的地址，最后就变成 *(int*)addr 如果希望能得到正确的整数值，必须要求addr所在的地址是一个真实的int类型。但是当我们传入double时，实际上其内存布局和int完全不同，因此我们得不到需要的整数。感兴趣的朋友可以用下面简单的代码做测试： double a. a=1.1. int b = *(int*) &amp. a. 因此，当我们调用有不定参数列表的函数时，不要期望系统做隐式类型转换，系统不会做这种检查或者转换，你给的参数类型必须严格和你希望的值一样。 2009年上半年全国计算机等级考试参考答案请进入计算机考试论坛 2009年全国计算机等级考试报名信息汇总 2009年NCRE考试有新变化 2009年全国计算机等级考试大纲 2009年上半年全国计算机二级考试试题及答案 2009年上半年全国计算机等级考试试题答案汇总 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com