[**C语言 ## \_\_VA\_ARGS\_\_ 宏**](http://www.cnblogs.com/alexshi/archive/2012/03/09/2388453.html)

在GNU C中，宏可以接受可变数目的参数，就象函数一样，例如:

#define pr\_debug(fmt,arg...) \

printk(KERN\_DEBUG fmt, ##arg)

用可变参数宏(variadic macros)传递可变参数表   
你可能很熟悉在函数中使用可变参数表，如:

void printf(const char\* format, ...);

直到最近，可变参数表还是只能应用在真正的函数中，不能使用在宏中。

C99编译器标准终于改变了这种局面，它允许你可以定义可变参数宏(variadic macros)，这样你就可以使用拥有可以变化的参数表的宏。可变参数宏就像下面这个样子:

#define debug(...) printf(\_\_VA\_ARGS\_\_)

缺省号代表一个可以变化的参数表。使用保留名 \_\_VA\_ARGS\_\_ 把参数传递给宏。当宏的调用展开时，实际的参数就传递给 printf()了。例如:

Debug("Y = %d\n", y);

而处理器会把宏的调用替换成:

printf("Y = %d\n", y);

因为debug()是一个可变参数宏，你能在每一次调用中传递不同数目的参数:

debug("test");&nbsp; // 一个参数

可变参数宏不被ANSI/ISO C++ 所正式支持。因此，你应当检查你的编译器，看它是否支持这项技术。

用GCC和C99的可变参数宏， 更方便地打印调试信息

gcc的预处理提供的可变参数宏定义真是好用:

#ifdef DEBUG

#define dbgprint(format,args...) \

fprintf(stderr, format, ##args)

#else

#define dbgprint(format,args...)

#endif

如此定义之后，代码中就可以用dbgprint了，例如dbgprint("%s", \_\_FILE\_\_);

下面是C99的方法:

#define dgbmsg(fmt,...) printf(fmt,\_\_VA\_ARGS\_\_)

新的C99规范支持了可变参数的宏   
具体使用如下:

以下内容为程序代码:

#include <stdarg.h>

#include <stdio.h>

#define LOGSTRINGS(fm, ...) printf(fm,\_\_VA\_ARGS\_\_)

int main()

{

LOGSTRINGS("hello, %d ", 10);

return 0;

}

但现在似乎只有gcc才支持。   
可变参数的宏里的'##'操作说明带有可变参数的宏(Macros with a Variable Number of Arguments)   
在1999年版本的ISO C 标准中，宏可以象函数一样，定义时可以带有可变参数。宏的语法和函数的语法类似。下面有个例子:

#define debug(format, ...) fprintf (stderr, format, \_\_VA\_ARGS\_\_)

这里，'...'指可变参数。这类宏在被调用时，它(这里指'...')被表示成零个或多个符号，包括里面的逗号，一直到到右括弧结束为止。当被调用时，在宏体(macro body)中，那些符号序列集合将代替里面的\_\_VA\_ARGS\_\_标识符。更多的信息可以参考CPP手册。   
GCC始终支持复杂的宏，它使用一种不同的语法从而可以使你可以给可变参数一个名字，如同其它参数一样。例如下面的例子:

#define debug(format, args...) fprintf (stderr, format, args)

这和上面举的那个ISO C定义的宏例子是完全一样的，但是这么写可读性更强并且更容易进行描述。   
GNU CPP还有两种更复杂的宏扩展，支持上面两种格式的定义格式。   
在标准C里，你不能省略可变参数，但是你却可以给它传递一个空的参数。例如，下面的宏调用在ISO C里是非法的，因为字符串后面没有逗号:   
debug ("A message")   
GNU CPP在这种情况下可以让你完全的忽略可变参数。在上面的例子中，编译器仍然会有问题(complain)，因为宏展开后，里面的字符串后面会有个多余的逗号。

为了解决这个问题，CPP使用一个特殊的'##'操作。书写格式为:

#define debug(format, ...) fprintf (stderr, format, ## \_\_VA\_ARGS\_\_)

这里，如果可变参数被忽略或为空，'##'操作将使预处理器(preprocessor)去除掉它前面的那个逗号。如果你在宏调用时，确实提供了一些可变参数，GNU CPP也会工作正常，它会把这些可变参数放到逗号的后面。象其它的pasted macro参数一样，这些参数不是宏的扩展。   
##还可以起到替换作用   
如:

#define FUN(IName) IName##\_ptr

这里将会把IName变成实际数据.

怎样写参数个数可变的宏   
一种流行的技巧是用一个单独的用括弧括起来的的 ``参数" 定义和调用宏, 参数在 宏扩展的时候成为类似 printf() 那样的函数的整个参数列表。

#define DEBUG(args) (printf("DEBUG: "), printf args)

if (n != 0) DEBUG(("n is %d\n", n));

明显的缺陷是调用者必须记住使用一对额外的括弧。   
gcc 有一个扩展可以让函数式的宏接受可变个数的参数。 但这不是标准。另一种 可能的解决方案是根据参数个数使用多个宏 (DEBUG1, DEBUG2, 等等), 或者用逗号玩个这样的花招:

#define DEBUG(args) (printf("DEBUG: "), printf(args))

#define \_ ,

DEBUG("i = %d" \_ i);

C99 引入了对参数个数可变的函数式宏的正式支持。在宏 ``原型" 的末尾加上符号 ... (就像在参数可变的函数定义中), 宏定义中的伪宏 \_\_VA\_ARGS\_\_ 就会在调用是 替换成可变参数。   
最后, 你总是可以使用真实的函数, 接受明确定义的可变参数

如果你需要替换宏, 使用一个 函数和一个非函数式宏, 如 #define printf myprintf