**C/C++ 中##(两个井号)和#(一个井号)用法**

**##(两个井号)和#(一个井号)都是什么意思**

连接符 ##(两个井号)

不知道什么符 #(一个井号)

## 连接符号由两个井号组成，其功能是在带参数的宏定义中将两个子串(token)联接起来，从而形成一个新的子串。但它不可以是第一个或者最后一个子串。所谓的子串(token)就是指编译器能够识别的最小语法单元。具体的定义在编译原理里有详尽的解释

**#符是把传递过来的参数当成字符串进行替代。**

假设程序中已经定义了这样一个带参数的宏：

#define PRINT( n ) printf( "token" #n " = %d", token##n )

同时又定义了二个整形变量：

int token9 = 9;

现在在主程序中以下面的方式调用这个宏：

PRINT( 9 );

那么在编译时，上面的这句话被扩展为：

printf( "token" "9" " = %d", token9 );

注意到在这个例子中，PRINT(9);中的这个”9”被原封不动的当成了一个字符串，与”token”连接在了一起，从而成为了token9。而#n也被”9”所替代。

可想而知，上面程序运行的结果就是在屏幕上打印出

**token9=9**

还有点不明白?!

再来一个例子:

#define PRINT( n ) printf( "token" #n " = %d", game##n )

int token9 = 9;

int game9 = 99;

调用:

PRINT(9);

屏幕上打印出:

**token9 = 99**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**C++中##(两个井号)和#(一个井号)用法（转）**

C（和C++）中的宏（Macro）属于编译器预处理的范畴，属于编译期概念（而非运行期概念）。下面对常遇到的宏的使用问题做了简单总结。  
  
关 于#和##  
  
在C语言的宏中，#的功能是将其后面的宏参数进行字符串化操作（Stringfication），简单说就是在对它所引用的宏 变量通过替换后在其左右各加上一个双引号。比如下面代码中的宏：  
  
#define WARN\_IF(EXP) \  
  
do{ if (EXP) \  
  
fprintf(stderr, "Warning: " #EXP "\n"); } \  
  
while(0)  
  
那么实际使用中会出现下面所示的替换过程：  
  
WARN\_IF (divider == 0);  
  
  
被替换为  
  
  
do {  
  
if (divider == 0)  
  
fprintf(stderr, "Warning" "divider == 0" "\n");  
  
} while(0);  
  
这样每次divider（除数）为0的时候便会在标 准错误流上输出一个提示信息。  
  
而##被称为连接符（concatenator），用来将两个Token连接为一个Token。注意这里连 接的对象是Token就行，而不一定是宏的变量。比如你要做一个菜单项命令名和函数指针组成的结构体的数组，并且希望在函数名和菜单项命令名之间有直观 的、名字上的关系。那么下面的代码就非常实用：  
  
struct command  
  
{  
  
char \* name;  
  
void (\*function) (void);  
  
};  
  
  
#define COMMAND(NAME) { NAME, NAME ## \_command }  
  
  
// 然后你就用一些预先定义好的命令来方便的初始化一个command结构的数组了：  
  
  
struct command commands[] = {  
  
COMMAND(quit),  
  
COMMAND(help),  
  
...  
  
}  
  
COMMAND宏在这里充当一个代码生成器的作用，这样可以在一定程度上减少代码密度，间接地也可 以减少不留心所造成的错误。我们还可以n个##符号连接 n+1个Token，这个特性也是#符号所不具备的。比如：  
  
#define LINK\_MULTIPLE(a,b,c,d) a##\_##b##\_##c##\_##d  
  
  
typedef struct \_record\_type LINK\_MULTIPLE(name,company,position,salary);  
  
// 这里这个语句将展开为：  
  
// typedef struct \_record\_type name\_company\_position\_salary;  
  
关于...的使用  
  
...在C宏中称为Variadic Macro，也就是变参宏。比如：  
  
#define myprintf(templt,...) fprintf(stderr,templt,\_\_VA\_ARGS\_\_)  
  
  
// 或者  
  
  
#define myprintf(templt,args...) fprintf(stderr,templt,args)  
  
第一个宏中由于没有对变 参起名，我们用默认的宏\_\_VA\_ARGS\_\_来替代它。第二个宏中，我们显式地命名变参为args，那么我们在宏定义中就可以用args来代指变参了。 同C语言的stdcall一样，变参必须作为参数表的最有一项出现。当上面的宏中我们只能提供第一个参数templt时，C标准要求我们必须写成：  
  
myprintf(templt,);  
  
的 形式。这时的替换过程为：  
  
myprintf("Error!\n",);  
  
  
替换为：  
  
  
  
fprintf(stderr,"Error!\n",);  
  
这是一个语法错误，不能正常编译。这个问题一般有 两个解决方法。首先，GNU CPP提供的解决方法允许上面的宏调用写成：  
  
myprintf(templt);  
  
而它将会 被通过替换变成：  
  
fprintf(stderr,"Error!\n",);  
  
很明显，这里仍然会产生编译错误（非本例的 某些情况下不会产生编译错误）。除了这种方式外，c99和GNU CPP都支持下面的宏定义方式：  
  
#define myprintf(templt, ...) fprintf(stderr,templt, ##\_\_VAR\_ARGS\_\_)  
  
这 时，##这个连接符号充当的作用就是当\_\_VAR\_ARGS\_\_为空的时候，消除前面的那个逗号。那么此时的翻译过程如下：  
  
myprintf(templt);  
  
  
被转化为：  
  
  
fprintf(stderr,templt);  
  
这样如果templt合法，将不会产生 编译错误。 这里列出了一些宏使用中容易出错的地方，以及合适的使用方式。  
  
错误的嵌套－Misnesting  
  
宏的定义不一定要有完整的、配对的括号，但是为了避免出错并且提高可读性，最好避免这样使用。  
  
由 操作符优先级引起的问题－Operator Precedence Problem  
  
由于宏只是简单的替换，宏的参数如果是复合结构，那么 通过替换之后可能由于各个参数之间的操作符优先级高于单个参数内部各部分之间相互作用的操作符优先级，如果我们不用括号保护各个宏参数，可能会产生预想不 到的情形。比如：  
  
#define ceil\_div(x, y) (x + y - 1) / y  
  
那么  
  
a = ceil\_div( b & c, sizeof(int) );  
  
将被转化为：  
  
a = ( b & c + sizeof(int) - 1) / sizeof(int);  
  
// 由于+/-的优先级高于&的优先级，那么上面式子等同于：  
  
a = ( b & (c + sizeof(int) - 1)) / sizeof(int);  
  
这显然不是调用者的初衷。为了避免这种情况发生，应当多写几个括号：  
  
#define ceil\_div(x, y) (((x) + (y) - 1) / (y))  
  
消除多余的分号－Semicolon Swallowing  
  
通常情况下，为了使函数模样的宏在表面上看起来像一个通常的C语言调用一样，通常情况下我们在宏的后面加上一个分 号，比如下面的带参宏：  
  
MY\_MACRO(x);  
  
但是如果是下面的情况：  
  
#define MY\_MACRO(x) { \  
  
/\* line 1 \*/ \  
  
/\* line 2 \*/ \  
  
/\* line 3 \*/ }  
  
  
  
//...  
  
  
if (condition())  
  
MY\_MACRO(a);  
  
else  
  
{...}  
  
这样会由于多出的那个分号产生编译错误。为了避免这种情况出现同时保持MY\_MACRO(x);的这种写法，我们 需要把宏定义为这种形式：  
  
#define MY\_MACRO(x) do {  
  
/\* line 1 \*/ \  
  
/\* line 2 \*/ \  
  
/\* line 3 \*/ } while(0)  
  
这样只要保证总是使用分号，就不会有任何问题。  
  
Duplication of Side Effects  
  
这里的Side Effect是指宏在展开的时候对其参数可能进行多次Evaluation（也就是取值），但是如果这个宏参数是一个函数，那么就有可能被调用多次从而达 到不一致的结果，甚至会发生更严重的错误。比如：  
  
#define min(X,Y) ((X) > (Y) ? (Y) : (X))  
  
  
//...  
  
  
  
c = min(a,foo(b));  
  
这 时foo()函数就被调用了两次。为了解决这个潜在的问题，我们应当这样写min(X,Y)这个宏：  
  
#define min(X,Y) ({ \  
  
typeof (X) x\_ = (X); \  
  
typeof (Y) y\_ = (Y); \  
  
(x\_ < y\_) ? x\_ : y\_; })  
  
({...})的作用是将内部的几条语句中最后一条的值返回，它也允许在内部声明变量（因为它通过大括号组成了一个局部 Scope）。  
  
==  
  
#define display(name) printf(""#name"")   
int main() {   
display(name);   
}   
运行结果是name,为什么不是"#name"呢？   
---------------------------------------------------------------   
  
#在这里是字符串化的意思   
printf(""#name"") 相当于   
printf("" "name" "")   
---------------------------------------------------------------   
  
The number-sign or "stringizing" operator (#) converts macro parameters (after expansion) to string constants  
---------------------------------------------------------------   
  
printf("" ＃name "") <1>   
相当于printf("" "name" "") <2>   
而<2>中的第2，3个“中间时空格 等价于("空＋name＋空')   
---------------------------------------------------------------   
  
## 连接符与# 符   
  
##连接符号由两个井号组成，其功能是在带参数的宏定义中将两个子串(token)联接起来， 从而形成一个新的子串。但它不可以是第一个或者最后一个子串。所谓的子串 (token)就是指编译器能够识别的最小语法单元。具体的定义在编译原理里有详尽的解释，但不知道也无所谓。同时值得注意的是#符是把传递过来的参数当 成字符串进行替代。下面来看看它们是怎样工作的。这是MSDN上的一个例子。   
  
假设程序中已经定义了这样一个带参数的宏：   
#define paster( n ) printf( "token" #n " = %d", token##n )   
  
同时又定义了一个整形变 量：   
int token9 = 9;   
  
现在在主程序中以下面的方式调用这个宏：   
paster( 9 );   
  
那 么在编译时，上面的这句话被扩展为：   
printf( "token" "9" " = %d", token9 );   
  
注意到 在这个例子中，paster(9);中的这个”9”被原封不动的当成了一个字符串，与”token”连接在了一起，从而成为了token9。而#n也 被”9”所替代。   
  
可想而知，上面程序运行的结果就是在屏幕上打印出token9=9   
---------------------------------------------------------------   
  
#define display(name) printf(""#name"")   
int main() {   
display(name);   
}   
====================================   
特殊性就在 于它是个宏，宏里面处理#号就如LS所说！   
处理后就是一个附加的字符串！   
  
但printf(""#name"") ；就不行了！   
---------------------------------------------------------------   
  
#define display(name) printf(""#name"")   
  
该定义 字符串化name，   
得 到结果其实就是 printf("name")   
(前后的空字符串拿掉)   
  
这样输出来的自然是 name   
  
从另 外一个角度讲，   
#是一个连接符号，   
参与运算了， 自然不会输出了 ...