# #define 用法集锦

## Definition：

The #define Directive

You can use the #define directive to give a meaningful name to a constant in your program. The two forms of the syntax are:

Syntax

#define identifier token-stringopt

#define identifier[( identifieropt, … , identifieropt )] token-stringopt

## Usage：

### 1. 简单的 define 定义

#define MAXTIME 1000

一个简单的 MAXTIME 就定义好了，它代表 1000，如果在程序里面写

if(i<MAXTIME){.....}

编译器在处理这个代码之前会对 MAXTIME 进行处理替换为 1000。

这样的定义看起来类似于普通的常量定义 CONST，但也有着不同，因为 define 的定义更像是简单的文本替换，而不是作为一个量来使用，这个问题在下面反映的尤为突出。

### 2.define 的“函数定义”

define 可以像函数那样接受一些参数，如下

#define max(x,y) (x)>(y)?(x):(y);

这个定义就将返回两个数中较大的那个，看到了吗？因为这个“函数”没有类型检查，就好像一个函数模板似的，当然，它绝对没有模板那么安全就是了。可以作为一个简单的模板来使用而已。

但是这样做的话存在隐患，例子如下：

#define Add(a,b) a+b;

在一般使用的时候是没有问题的，但是如果遇到如：c * Add(a,b) * d 的时候就会出现问题，代数式的本意是 a+b 然后去和 c，d 相乘，但是因为使用了 define（它只是一个简单的替换），所以式子实际上变成了 c*a + b*d

另外举一个例子：

#define pin (int*);

pin a,b;

本意是 a 和 b 都是 int 型指针，但是实际上变成 int* a,b;

a 是 int 型指针，而 b 是 int 型变量。

这是应该使用 typedef 来代替 define，这样 a 和 b 就都是 int 型指针了。

所以我们在定义的时候，养成一个良好的习惯，建议所有的层次都要加括号。

### 3.宏的单行定义（少见用法）

#define A(x) T_##x

#define B（x) #@x

#define C（x) #x

我们假设：x=1，则有：

A(1)------〉T_1

B(1)------〉'1'

C(1)------〉"1"

（这里参考了 hustli 的文章）

## 3.define 的多行定义

define 可以替代多行的代码，例如 MFC 中的宏定义（非常的经典，虽然让人看了恶心）

```
#define MACRO(arg1, arg2) do { \
/* declarations */ \
stmt1; \
stmt2; \
/* ... */ \
} while(0) /* (no trailing ; ) */
```

关键是要在每一个换行的时候加上一个"\"

## 4.在大规模的开发过程中，特别是跨平台和系统的软件里，define 最重要的功能是条件编译。

就是：

```
#ifdef WINDOWS
.....
.....
#endif
#ifdef LINUX
.....
.....
#endif
```

可以在编译的时候通过#define 设置编译环境

## 5.如何定义宏、取消宏

```
//定义宏
#define [MacroName] [MacroValue]
//取消宏
#undef [MacroName]
//普通宏
#define PI (3.1415926)
```

带参数的宏

```
#define max(a,b) ((a)>(b)? (a),(b))
```

关键是十分容易产生错误，包括机器和人理解上的差异等等。

## 6.条件编译

#ifdef XXX...(#else) ... #endif

例如

```
#ifdef DV22_AUX_INPUT
#define AUX_MODE 3
#else
#define AUY_MODE 3
```

#endif
#ifndef XXX ... (#else) ... #endif

**7.头文件(.h)可以被头文件或 C 文件包含；**

重复包含（重复定义）

由于头文件包含可以嵌套，那么 C 文件就有可能包含多次同一个头文件，就可能出现重复定义的问题的。

通过条件编译开关来避免重复包含（重复定义）

例如

```
#ifndef __headerfileXXX__
#define __headerfileXXX__
…
//文件内容
…
#endif
```

# Instances：

**1、防止一个头文件被重复包含**

```
#ifndef COMDEF_H
#define COMDEF_H
 //头文件内容
#endif
```

**2、重新定义一些类型，防止由于各种平台和编译器的不同，而产生的类型字节数差异，方便移植。**

```
typedef  unsigned char      boolean;      /* Boolean value type. */
typedef  unsigned long int  uint32;       /* Unsigned 32 bit value */
typedef  unsigned short     uint16;       /* Unsigned 16 bit value */
typedef  unsigned char      uint8;        /* Unsigned 8  bit value */
typedef  signed long int    int32;        /* Signed 32 bit value */
typedef  signed short       int16;        /* Signed 16 bit value */
typedef  signed char        int8;         /* Signed 8  bit value */
//下面的不建议使用
typedef  unsigned char      byte;         /* Unsigned 8  bit value type. */
typedef  unsigned short     word;         /* Unsinged 16 bit value type. */
typedef  unsigned long      dword;        /* Unsigned 32 bit value type. */
typedef  unsigned char      uint1;        /* Unsigned 8  bit value type. */
typedef  unsigned short     uint2;        /* Unsigned 16 bit value type. */
typedef  unsigned long      uint4;        /* Unsigned 32 bit value type. */
typedef  signed char        int1;         /* Signed 8  bit value type. */
typedef  signed short       int2;         /* Signed 16 bit value type. */
typedef  long int           int4;         /* Signed 32 bit value type. */
typedef  signed long        sint31;       /* Signed 32 bit value */
typedef  signed short       sint15;       /* Signed 16 bit value */
```

```
typedef  signed char       sint7;       /* Signed 8  bit value */
```

**3、得到指定地址上的一个字节或字**

```
#define  MEM_B( x )  ( *( (byte *) (x) ) )
#define  MEM_W( x )  ( *( (word *) (x) ) )
```

**4、求最大值和最小值**

```
  #define  MAX( x, y ) ( ((x) > (y)) ? (x) : (y) )
  #define  MIN( x, y ) ( ((x) < (y)) ? (x) : (y) )
```

**5、得到一个 field 在结构体(struct)中的偏移量**

```
#define FPOS( type, field ) \
/*lint -e545 */ ( (dword) &(( type *) 0)-> field ) /*lint +e545 */
```

**6、得到一个结构体中 field 所占用的字节数**

```
#define FSIZ( type, field ) sizeof( ((type *) 0)->field )
```

**7、按照 LSB 格式把两个字节转化为一个 Word**

```
#define  FLIPW( ray ) ( (((word) (ray)[0]) * 256) + (ray)[1] )
```

**8、按照 LSB 格式把一个 Word 转化为两个字节**

```
#define  FLOPW( ray, val ) \
  (ray)[0] = ((val) / 256); \
  (ray)[1] = ((val) & 0xFF)
```

**9、得到一个变量的地址（word 宽度）**

```
#define  B_PTR( var )  ( (byte *) (void *) &(var) )
#define  W_PTR( var )  ( (word *) (void *) &(var) )
```

**10、得到一个字的高位和低位字节**

```
#define  WORD_LO(xxx)  ((byte) ((word)(xxx) & 255))
#define  WORD_HI(xxx)  ((byte) ((word)(xxx) >> 8))
```

**11、返回一个比 X 大的最接近的 8 的倍数**

```
#define RND8( x )       ((((x) + 7) / 8 ) * 8 )
```

**12、将一个字母转换为大写**

```
#define  UPCASE( c ) ( ((c) >= 'a' && (c) <= 'z') ? ((c) - 0x20) : (c) )
```

**13、判断字符是不是 10 进值的数字**

```
#define  DECCHK( c ) ((c) >= '0' && (c) <= '9')
```

**14、判断字符是不是 16 进值的数字**

```
#define  HEXCHK( c ) ( ((c) >= '0' && (c) <= '9') ||\
                      ((c) >= 'A' && (c) <= 'F') ||\
((c) >= 'a' && (c) <= 'f') )
```

**15、防止溢出的一个方法**

```
#define  INC_SAT( val )  (val = ((val)+1 > (val)) ? (val)+1 : (val))
```

**16、返回数组元素的个数**

```
#define  ARR_SIZE( a )  ( sizeof( (a) ) / sizeof( (a[0]) ) )
```

**17、返回一个无符号数 n 尾的值 MOD_BY_POWER_OF_TWO(X,n)=X%(2^n)**

```
#define MOD_BY_POWER_OF_TWO( val, mod_by ) \
         ( (dword)(val) & (dword)((mod_by)-1) )
```

**18、对于 IO 空间映射在存储空间的结构，输入输出处理**

```
  #define inp(port)         (*((volatile byte *) (port)))
  #define inpw(port)        (*((volatile word *) (port)))
```

```
#define inpdw(port)       (*((volatile dword *)(port)))
#define outp(port, val)   (*((volatile byte *) (port)) = ((byte) (val)))
#define outpw(port, val)  (*((volatile word *) (port)) = ((word) (val)))
#define outpdw(port, val) (*((volatile dword *) (port)) = ((dword) (val)))
```

## 19、使用一些宏跟踪调试

ANSI 标准说明了五个预定义的宏名。它们是：

__LINE__

__FILE__

__DATE__

__TIME__

__STDC__

C++中还定义了 __cplusplus

如果编译器不是标准的,则可能仅支持以上宏名中的几个,或根本不支持。记住编译程序也许还提供其它预定义的宏名。

__LINE__ 及 __FILE__ 宏指示，#line 指令可以改变它的值，简单的讲，编译时，它们包含程序的当前行数和文件名。

__DATE__ 宏指令含有形式为月/日/年的串,表示源文件被翻译到代码时的日期。

__TIME__ 宏指令包含程序编译的时间。时间用字符串表示，其形式为：分：秒

__STDC__ 宏指令的意义是编译时定义的。一般来讲，如果__STDC__已经定义，编译器将仅接受不包含任何非标准扩展的标准 C/C++代码。如果实现是标准的,则宏__STDC__含有十进制常量 1。如果它含有任何其它数,则实现是非标准的。

__cplusplus 与标准 c++一致的编译器把它定义为一个包含至少 6 为的数值。与标准 c++不一致的编译器将使用具有 5 位或更少的数值。

可以定义宏,例如:

当定义了_DEBUG,输出数据信息和所在文件所在行

```
#ifdef _DEBUG
#define DEBUGMSG(msg,date) printf(msg);printf(“%d%d%d”,date,_LINE_,_FILE_)
#else
#define DEBUGMSG(msg,date)
#endif
```

## 20、宏定义防止错误使用小括号包含。

例如：

有问题的定义：#define DUMP_WRITE(addr,nr) {memcpy(bufp,addr,nr); bufp += nr;}

应该使用的定义： #difne DO(a,b) do{a+b;a++;}while(0)

例如：

```
if(addr)
    DUMP_WRITE(addr,nr);
else
    do_somethong_else();
```

宏展开以后变成这样:

```
if(addr)
    {memcpy(bufp,addr,nr); bufp += nr;};
else
```

```
    do_something_else();
```
gcc 在碰到 else 前面的“;”时就认为 if 语句已经结束，因而后面的 else 不在 if 语句中。而采用 do{} while(0) 的定义，在任何情况下都没有问题。而改为 #difne DO(a,b) do{a+b;a++;}while(0) 的定义则在任何情况下都不会出错

## 21. define 中的特殊标识符

```
#define Conn(x,y) x##y
#define ToChar(x) #@x
#define ToString(x) #x

int a=Conn(12,34);
char b=ToChar(a);
char c[]=ToString(a);
```
结果是 a=1234,c='a',c='1234';
可以看出 ## 是简单的连接符，#@用来给参数加单引号，#用来给参数加双引号即转成字符串

```
#ifdef OS_GLOBALS.....................................1
#define OS_EXT.....................................2
#else.....................................3
#define OS_EXT extern.....................................4
#endif.....................................5
```

意思就是说,如果 OS_GLOBALS 被定义,则本文件中的 OS_EXT 被替换为空(被忽略), 如果 OS_GLOBALS 未被定义,则本文件中的 OS_EXT 被替换为 extern 关键字. 意义比较明确, 就是给出了本文件中哪些变量和函数,在何时需要使用 extern 引用 (取决于其他文件中有无定义 OS_GLOBALS.
为方便你理解, 举个例子.
现在有两文件 AAA.c 和 BBB.c

AAA.c 内容是

```
#define OS_GLOBALS
```

BBB.c 内容是

```
#ifdef OS_GLOBALS
#define OS_EXT
#else
#define OS_EXT extern
#endif
OS_EXT void function1(void);
```

如果 AAA.c 先于 BBB.c 编译时,OS_GLOBALS 被定义,在 BBB.c 里,实际上是

void function1(void); // 声明了自己的函数
如果 BBB.c 先于 AAA.c 编译,则 OS_GLOBALS 被定义,在 BBB.c 里,实际上是 extern void function1(void); // 声明了外部的一个函数.

另外,你想了解的比较重要的是,为什么 define 后面可以只有一个名字,这个语句的意思就是,将 XXXX 替换为空(从文本上忽略).比如
#define OS_EXT
意思就是说,在本文件中,凡 OS_EXT 文本串,在编译时都被替换成空白,被忽略,或者说删除.然而,这不影响它作为#ifdef 判断的有效性,OS_EXT 仍然是一个被 define 过的东西.就这个意思,应该理解了吧.

多重包含在绝大多数情况下出现在大型程序中，它往往需要使用很多头文件，因此要发现重复包含并不容易。要解决这个问题，我们可以使用条件编译。如果所有的头文件都像下面这样编写:

```
#ifndef  _HEADERNAME_H
#define  _HEADERNAME_H

...

#endif
```