

# 嵌入式C语言编码规范

# 规范内容

## 一、引言

## 二、规范

1、文件内部构成

2、命名规范

3、标识符和常量

4、类型和类型转换

5、初始化、声明和定义

6、控制语句和表达式

7、函数

8、指针和数组

9、结构与联合

10、预处理指令

# 一、引言

嵌入式系统在各行各业都得到了广泛应用，C语言的使用也越来越体现出广泛性，因此嵌入式软件的安全可靠性变得尤为重要。制定本规范的目的与意义在于：

- 1、树立良好的编程习惯和编程思路，摒弃那些可能存在风险的编程行为。保证编写出安全健壮的代码，进而保证嵌入式产品的安全性、可靠性。
- 2、使编写的代码更加容易阅读、容易理解而且容易维护。
- 3、良好的编程风格是提高程序可靠性非常重要的手段，也是大型项目多人合作开发的技术基础。
- 4、遵循良好的共通的编码规范，也是提高编码能力，保证软件工程这个阶段质量的一个重要手段。同时也是衡量一个组织软件开发能力的一个重要指标。

# 二、规范

## 1、文件内部构成

用于存储源代码的C 程序文件可以分为两类：源文件和头文件。源文件和头文件中包含的内容是不同的。

**源文件主要包括以下内容：**

- 只在本文件内部使用的（对外部隐藏的）类型；
- 只在本文件内部使用的（对外部隐藏的）常量；
- 只在本文件内部使用的（对外部隐藏的）宏定义；
- 全局变量和文件级（static）变量的定义；
- 函数原型声明和函数定义；
- 包含文件部分，文件头的说明，函数头的说明。

**头文件中包含如下内容：**

- 提供给外部参照的类型；
- 提供给外部参照常量；
- 提供给外部参照宏定义；
- 提供给外部参照（全局）函数原型声明；
- 提供给外部参照全局变量的外部声明；
- 包含文件部分，文件头的说明。但头文件中不要定义变量。

# 文件头说明实例

```
>/*****  
>* File Name : DP_DrawE.c  
>* Model Name : MF7878/R/J  
>* Module Name : Draw Engine/Display  
>* uCom : Mitsubishi M16C/80 series  
>*  
>* Create Date : 1999/10/01  
>* Author/Corporation : WhoAmI/NAS  
>*  
>* Abstract Description : Place some description here.  
>*  
>*-----Revision History-----  
>* No Version Date Revised By Item Description  
>* 1 V0.95 00.05.18 WhoAmI[NAS] abcdefghijklm WhatUDo  
>*  
>*****/  
>Source/ Header File Header Section各部分内容的含义说明，请参考下面内容。  
>1) 文件名信息；  
>2) 适用的产品型号（Model）名称：可以是多个型号；  
>3) 所属的模块（Module）名称：当模块很大时，可以考虑在大模块内增加子模块的标示；  
>4) 适用的处理器（ $\mu$ Com）型号：可以是多个型号；  
>5) 预先包含头文件：只有在头文件的描述中使用，注明包含本文件之前应该首先包含的头文件；  
>6) 文件创建日期；  
>7) 文件创建者/公司名称；  
>8) 概要描述：概要的描述文件的功能、构成等信息，如果存在特殊的考虑，也请注明；  
>9) 修改履历，其中请标明：  
> 1) 修改序号（No.）；  
> 2) 修改对应版本号（Version）；  
> 3) 修改日期（Date）；  
> 4) 修改人（Revised By）；  
> 5) 修改项（Item）；  
> 6) 修改描述（Description）：请注明修改的原因和对策，如果存在特殊的考虑，也请注明；  
>注意点：  
>修改履历的纪录一般在V0.80之后开始。但是，如果目前ver0.80的完成度和质量普遍很低，因此在此具体的项目中可以考虑提高开始纪录修改履历的版本号。
```

## 2 命名规范

**规范2.1 关于文件标识符命名规则，请遵照以下规范：**

文件标识符分为两部分，即文件名前缀和后缀。格式如下：

× × × …… × × . × × ×

- 1) 文件名前缀表示该文件的内容或作用，可以由项目组成员统一约定。最好不要超过8个字符；文件名前缀的最前面要使用范围限定符——模块名（文件名）缩写；
- 2) 文件名后缀表示该文件的类型，该部分最多为3个字符：
  - 1) 源文件：.c；
  - 2) 头文件：.h；
  - 3) 其它类型文件：如.tbl文件等，使用之前进行统一规定。
- 3) 前缀和后缀这两部分字符应仅使用字母、数字和下划线。文件标识的长度不能超过32个字符，以便于识别；
- 4) 本规范建议文件名全部使用大写。

## 2 命名规范

### 规范2.2 关于模块标识符命名规则：

- 1、模块名就是范围限定符，各种全局标识符（文件名、全局函数名、全局变量名等）的命名，必须使用范围限定符作为前缀。
- 2、模块名必须进行适当的缩写。例如Stand By 模块，省略缩写为STBY；
- 3、模块名要求全部为大写。

### 规范2.3 关于C标识符命名规则，请按照「标识符前缀」 + 「含义标识」规范进行命名。

「标识符前缀」由以下元素构成，各部分内容需要遵守相应定义：

范围限定符前缀 + 作用域前缀 + 数据类型前缀 + 含义标识  
「标识符前缀」

范围限定符前缀的形式为：模块名 + 下划线，即模块名\_

## 2 命名规范

作用域前缀：

NO	标识符类型	作用域前缀
1	Global Variable	W
2	File Static Variable	n
3	Function Static Variable	fn
4	Auto Variable	a
5	Global Function	w
6	Static Function	n

数据类型前缀：1、参照Microsoft 的匈牙利标记法；2、对于用户自定义的类型（enum、struct、union），变量类型不做区分，统一使用“st”。

## 2 命名规范

No.	Data Type	Prefix	Example
1	bit	bt	bit btVariable;
2	boolean	b	boolean bVariable;
3	char	c	char cVariable;
4	int	i	int i;
5	short[int]	s	short[int] sVariable;
6	long[int]	l	long[int] lVariable;
7	unsigned[int]	u	unsigned[int] uiVariable;
8	double	d	double dVariable;
9	float	f	float fVariable;
10	pointer	p	void *pvVariable;
11	void	v/vd	void *pvVariable;
12	array of	a	char acVariable[TABLE_MAX];

## 2 命名规范

含义标识的命名：

- 1、各单词的开头用大写字母，其余用小写字母，省略用语除外；
- 2、禁止在数据命名的一部分插入数字（数学公式例外）；
- 3、进行命名的时候，在充分把握数据对象（变量、函数等）的内容含义的基础上，进行能明确显示其内容的命名(见名知意)。

1) 变量含义标识符构成：目标词+动词（过去分词）+[状语]+[目的地]；

2) 函数含义标识符构成：动词（一般现时）+目标词+[状语]+[目的地]；

Example：

变量：DataGotFromCD      从CD中取得的数据

      DataDeletedFromCD    从CD中删除数据

函数：GetDataFromCD

      DeleteDataFromCD

# 2 命名规范

## 标识符命名举例

NO	分类	举例说明
1	变量定义具有全局作用域	<code>INT8U <u>AMM_wucDDMDDataEnable</u>;</code>
2	变量定义具有局部作用域	<code>INT8U <b>aucBeepType</b>;</code>
3	函数定义具有局部作用域	<code><u>static void nvdAPIAmmKeyOutUp</u>(void)</code>
4	结构体定义具有局部作用域	<code>typedef struct { INT8U ucSize; INT8U aucReceiverID[RECEIVERID_SIZE]; INT8U *pucReceiverBuff; }SDARS_DispReceiverID_st,*pSDARS_DispReceiverID_st; <i><u>pSDARS_DispReceiverID_st apstDispReceiverI</u></i></code>

## 3 标识符和常量

**规范3.1 标识符的内部有效字符和外部有效字符不能多于31。**

便于编译器识别，代码清晰易读，并保证可移植性。

**规范3.2 具有内部作用域的标识符，不应与具有外部作用域的标识符重名，这会隐藏了外部标识符（同一文件）。**

在嵌套的范围中使用相同名称的标识符会使得代码非常混乱，例如：

```
INT16U i; /* 该变量具有外部作用域 */
{
    INT16U i; /* 定义了一个具有内部作用域的变量*/
    i = 3; /* NG :外部作用域变量被隐藏，容易引起代码混乱 */
}
```

**规范3.3 静态变量和全局变量的标识符不能重名（不同文件）。**

## 3 标识符和常量

[例][file1.c]

```
static INT8U nucVar1; /* NG:多个 file 中进行了定义 */  
void nvdfunc1(void)  
{  
...  
}
```

[file2.c]

```
INT8U nucVar1; /* NG:多个 file 中进行了定义 */  
INT8U nucVar2;  
void nvdfunc2(void)  
{  
...  
}
```

## 3 标识符和常量

**规范3.4 不应使用八进制常量(零除外)和八进制转义序列。**

任何以0零开始的整型常量都被看做是八进制的，所以这是危险的。如在书写固定长度的常量时，例如下面为3个数字位做数组初始化时，将产生非预期的结果。

(052是八进制的即十进制的42)：

```
code[1] = 109+100; /* equivalent to decimal 209 */
```

```
code[2] = 100+052; /* equivalent to decimal 142 */
```

## 4 类型和类型转换

*规范4.1 应该使用标明了大小和符号的typedef代替基本数据类型。不应使用基本数值类型char、int、short、long、float和double，而应使用typedef进行类型的定义。*

typedef 类型定义见下页附表。

# 4 类型和类型转换

typedef 类型定义附表：

No.	基本数据类型	typedef 定义
1	typedef unsigned char	BOOLEAN;
2	typedef unsigned char	boolean;
3	typedef unsigned char	bit;
4	typedef unsigned char	INT8U;
5	typedef signed char	INT8S;
6	typedef unsigned short int	INT16U;
7	typedef signed short int	INT16S;
8	typedef unsigned long int	INT32U;
9	typedef signed long int	INT32S;
10	typedef float	FP32;

## 4 类型和类型转换

*规范4.2 对于long型的整形常量应该追加后缀u(或U),UL(或ul)。提高代码的可读性。*

例：

```
INT32U aulVar32x;
```

```
aulVar32x = aulVar32x + 3; /* NG*/
```

```
aulVar32x = aulVar32x + 3u; /* OK*/
```

注意点：对char和short类型没有规定。

## 4 类型和类型转换

**规范4.3 使用不同类型的变量进行赋值/运算的时候、一定要使用强制转换运算符。防止由于隐式类型转换，引起信息丢失。**

[例1] NG：数据被截断

```
extern INT16U nuifunc2( void );
void nvdfunc1( )
{
    INT8U aucVar;
    aucVar = nuifunc2( ); /* NG：信息的一部丢失了*/
}
```

[例2] NG：浮点运算应该注意的情况

```
INT16U auiVar1, auiVer2;
FP32 afVar3;
auiVar1 = 1U;
auiVer2 = 3U;
afVar3 = auiVar1 / auiVer2; /* NG：运算结果=0.0 */
afVar3 = (FP32)auiVar1 / (FP32)auiVer2; /* OK：运算结果=0.333 */
```

## 4 类型和类型转换

**规范4.4 指针和整数之间的转换、以及指针和浮点数之间的转换要避免。如果指针所指向的类型带有const 或volatile 限定符，那么移除限定符的强制转换是不允许的。**

下面的两个例子：例1，意图不明。例2，破坏数据的完整性。

例1：NG，指针和整数间进行了转换

```
INT16U auiVar;  
INT8U *apucVar;  
apucVar = ( INT8U * ) auiVar; /* NG:整数和指针间进行转换*/
```

例2：NG，对volatile 限定符进行转换

```
volatile INT16U *wpuiVar;  
INT16U *apuiVar;  
apuiVar = (INT16U * ) wpuiVar; /* NG:对volatile 指向的指针进行  
转换*/  
  
if( apuiVar == 0U )  
{  
    ...  
}
```

## 5 初始化声明和定义

**规范5.1 自动变量在使用前都应被赋值。为避免错误严禁读取未经初始化的变量。**

本规范的意图是，使所有变量在其被读之前已经写过了，除了声明中的初始化。自动存储类型变量通常不是自动初始化的。具有静态存储类型的变量，缺省地被自动赋予零值。

[例] NG: 变量未初始化进行了使用

```
void nvdfunc1( void )
{
    INT16U auiVar1 = 10;
    INT16U auiVar2;
    auiVar2 += auiVar1; /* NG */
}
```

**规范5.2 枚举列表中不能显式用于除首元素之外的元素,除非所有的元素都是显式初始化的。**

如果枚举列表的成员没有显式地初始化，那么C将为其分配一个从0开始的整数序列。首元素为0，后续元素依次加1，这样确保所用初始化值一定要足够小，这样列表中的后续值就不会超出该枚举常量所用的int存储量。避免自动与手动分配的混合时易产生的错误。

# 5 初始化声明和定义

规范5.2 例：

```
enum colour { red = 3, blue, green, yellow = 5 } ;  
/* NG : green 和 yellow 代表了相同的值,可能是非预期的 */  
enum colour { red = 3, blue = 4, green = 5, yellow = 5 } ;  
/* OK : green 和 yellow 代表了相同的值,预期的 */
```

**规范5.3 局部变量的大小不要超过64 bytes。以避免stack溢出。如果局部变量确实过大，应该定义成static全局变量或从堆中分配。**

# 5 初始化声明和定义

## 规范5.4 数组和结构的非零初始化应该使用大括号。

数组结构和联合的初始化列表，要以一对大括号括起来。本规范更进一步地要求使用附加的大括号来指示嵌套的结构。它迫使程序员显式地考虑和描述复杂数据类型元素，另外也可以避免产生错误，提高程序的可读性。比如多维数组的初始化次序：

例：

```
INT16U y[3][2] = { 1, 2, 3, 4, 5, 6 }; /* NG: 未有效使用{ }*/
```

```
INT16U y[3][2] = { { 1, 2 }, { 3, 4 }, { 5, 6 } }; /* OK */
```

```
struct stag
```

```
{
```

```
    INT16U uiMary1[1];
```

```
    INT16U uiMary2[2];
```

```
};
```

```
struct stag astVar = { {2}, {2,3} };
```

# 5 初始化声明和定义

**规范5.5 函数应当具有原型声明，原型声明在函数的定义范围和调用范围内都是可见的。**

原型的使用使得编译器能够检查函数定义和调用的完整性。如果没有原型就不会迫使编译器检查出函数调用当中的一定错误（比如函数体具有不同的参数数目，调用和定义之间参数类型的不匹配）。如下例：

```
[head.h]
extern void nvdfunc1( void );
extern FP32 nvdfunc3( INT16U auiArg );
[file_ok.c]
#include "head.h"
void nvdfunc2( void );
void nvdfunc2( void )
{
    ...
    afVar = nvdfunc3( auiVar ); /* OK : 原型声明可见 */
    ...
}
```

# 5 初始化声明和定义

## **规范5.6 函数不应该声明为具有块作用域。**

在块作用域中声明函数会引起混淆,并可能导致未定义的行为。

例：

```
void nvdfunc1( void )
{
    extern INT32U nvdfunc2( void ); /* NG : 函数内进行了声明 */
    static INT32U nvdfunc3( void ); /* NG : 函数内进行了声明 */
    ...
    nvdfunc3();
}
```

# 5 初始化声明和定义

**规范5.7** 如果对象的访问只是在单一的文件（函数）中，那么对象应该在文件（函数）范围内使用static进行声明。良好的编程实践是尽量避免使用全局标识符。

使用static存储类标识符将确保标识符只是在声明它的文件中是可见的。避免了和其他文件或库中的相同标识符发生混淆的可能性。

例: 只在内部使用的变量或者函数，进行static定义

```
static INT8U nucVar; /* OK */
static void nvdfunc1( void )
{
    nucVar = 1u;
    return nucVar;
}
static void nvdfunc2( void ); /* OK : static定义 */
static void nvdfunc1( void )
{
    nvdfunc2 ( ) ;
}
```

## 5 初始化声明和定义

**规范5.8** 对外提供的变量或函数应该在唯一的一个头文件中进行声明，不要在源文件内部直接声明，更不允许在不同文件中进行多次定义。

例：extern INT16U wuiVar1; [head.h] /\* OK \*/

[file1.c]

INT16U wuiVar1;

INT16U wuiVar2;

[file2.c]

INT16U wuiVar1; /\* NG \*/

**规范5.9** 当一个数组声明为具有外部链接,它的大小应该显式声明或者通过初始化进行隐式定义。

例:

INT8U array1[10]; /\* OK \*/

extern INT8U array2[]; /\* NG : 未指定数组大小 \*/

INT8U array2[] = { 0, 10, 15 }; /\* OK: \*/

**规范5.10** 所有的对象和函数标识符在使用前必须声明。

## 6 控制语句和表达式

**规范6.1 不要过分依赖C表达式中缺省的运算符优先级。**

使用相当复杂的C运算符优先级，很容易引起错误，使用括号可以避免这样的错误，并且可以使得代码更为清晰，可读。

例：下面表达式的计算结果为：

```
INT8U aucVar, aucVarLow = 1, aucVarHi = 2;
```

```
aucVar = aucVarHi << 4 + aucVarLow;
```

A. 33 B. 64 C. 17

**规范6.2 不能在具有副作用的表达式中使用sizeof运算符。**

当一个表达式使用了sizeof运算符，并期望计算表达式的值时，表达式是不会被计算的。sizeof只对表达式的类型有用。

例：

```
auiVar2 = sizeof (auiVar1= 1234); /* NG : auiVar1 并没有被赋值1234 */
```

**规范6.3 逻辑运算符&& 或||的右手操作数不能包含副作用。**

```
INT16U auiVar1, auiVar2, auiVar3;
```

```
if( (auiVar1 != 0U) && (auiVar2 == auiVar3++) ) /* NG : auiVar3++包含副作用*/
```

## 6 控制语句和表达式

**规范6.4 逻辑运算 (&&、|| 和!) 直接用到的操作数应该是布尔类型的，而布尔类型表达式也不能用做非逻辑运算。**

**规范6.5 位运算符不能用于基本类型(underlying type)是有符号的操作数上。位运算(~, <<, >>, &, ^ 和|)对有符号整数通常是无意义的。比如，如果右移运算把符号位移动到数据位上，或者左移运算把数据位移动到符号位上就会产生问题。**

**规范6.6 在一个表达式中，自增++和自减--运算符,不应同其他运算符混合在一起。**

不建议使用同其他算术运算符混合在一起的自增和自减运算符是因为

- 1) 它显著削弱了代码的可读性;
- 2) 在不同的变异环境下，会执行不同的运算次序，产生不同结果。

例：`INT16U auiVar, auiCnt, auiAry[10];`

```
    auiVar = auiAry[auiCnt] + auiCnt++; /* NG */
```

## 6 控制语句和表达式

**规范6.7 浮点表达式不能做相等或不等的判断。**

浮点型变量无论是float还是double类型的变量，都有精度限制。

**规范6.8 for 语句的三个表达式应该只关注循环控制。**

for 语句的三个表达式都给出时它们应该只用于如下目的：

第一个表达式初始化循环计数器；

第二个表达式包含对循环计数器和其他可选的循环控制变量的测试；

第三个表达式循环计数器的递增或递减。

例：for( auiCnt = 0U; auiCnt <=10U;

    auiVar = 2U \* auiVar + auiCnt \* 10U, auiCnt++) /\* NG\*/

**规范6.9 for 循环中用于计数的变量不应在循环体中修改。但是，如果变量不是用作计数器而是用于循环结束，那么可以修改。**

例：aucflag = 1;

    for ( i = 0; ( i < 5 ) && ( aucflag == 1 ); i++){

        aucflag = 0; /\* OK \*/

        i = i + 3; /\* NG\*/

    }

## 6 控制语句和表达式

**规范6.10** 组成*switch*、*while*、*do...while* 或*for* 结构体的语句应该是复合语句。即使该复合语句只包含一条语句也要扩在*{}*里。

例：NG，没有包括在大括号里

```
for ( i = 0 ; i < N_ELEMENTS ; ++i )  
    buffer[i] = 0;
```

**规范6.11** *if/else* 应该成对出现。所有的*f ... else if* 结构应该由*else* 子句结束。

**规范6.12** *switch* 语句中如果*case* 分支的内容不为空，那么必须以*break* 作为终了。

如果*case* 分支的内容不以*break* 终了的话，开发者的意图不是很明确，存在编程错误的可能性，代码的可读性降低。

**规范6.13** *switch* 语句的最后分支应该是*default* 分支。

## 7 函数

**规范7.1** 一定要显示声明函数的返回值类型，及所带的参数。如果没有要声明为void。

**规范7.2** 在函数的原型声明中应包含所有参数，不允许只包含参数类型的声明方式；没有参数时以“void”填充。

提高移植性，避免在某些编译器下编译不过的情况。例：

```
void nvdfunc( INT32U aulArg1, INT16U, INT32U aulArg4 ); /*NG*/  
void nvdfunc( INT32U aulSrc, INT16U auiWidth, INT32U aulList ); /* OK */  
void nvdfunc( INT32U, INT16U, INT32U ); /* NG*/
```

**规范7.3** 函数的声明和定义中使用的标识符(参数名)应该一致。

**规范7.4** 函数原型中的指针参数，如果不是用于修改所指向的对象，就应该声明为指向const的指针。

```
例：void nvdfunc ( INT16U *piParam1, const INT16U *piParam2,  
                  INT16U *piParam3 ){  
    *piParam1 = *piParam2 + *piParam3; /* NG:param3 没有声明为const */  
}
```

## 7 函数

**规范7.5** 函数调用时，只能使用加前缀&，或者使用括起来的参数列表，没有参数时，列表可以为空。

例：`extern INT16U nvdfunc2( void );`

```
void nvdfunc( void ){
```

```
    if (nvdfunc2) /* NG：请使用nvdfunc2() 或者&nvdfunc2 */{ }
```

```
}
```

**规范7.6** 不提倡使用递归函数调用。递归本身存在堆栈空间使用过度的危险，这能导致严重的错误。除非递归经过了非常严格的控制。

**规范7.7** 如果不需要修改输入参数的值，将变量值作为参数进行函数参数传递，而不是地址。提高代码性能和效率，防止误修改。

## 7 函数

**规范7.8** 注意控制参数的数量，一般来说不要超过5个，当参数过多时，应该考虑将参数定义为一个结构体，并且将结构体指针作为参数。

这样可以减少对堆栈的占用量。

**规范7.9** 函数传递的参数中结构体使用指针，可以减少参数本身处理的开销，减少堆栈的占用量。

这样也可以使代码更加规范。

**规范7.10** 函数的大小不要过长，一般定为200行以内（除去注释，空行，变量定义，调试开关等）。

限制函数的规模有助于函数功能的正确性和维护。

# 8 指针和数组

**规范8.1 指针在使用前一定要赋值，避免产生野指针。**

```
例：void nvdfunc( void ){  
    int i;  
    int *p;  
    i = 10;  
    *p = i; /*NG*/  
}
```

**规范8.2 分配内存后要立刻判断指针是否为NULL。**

如果对一个NULL指针进行间接访问，它的结果因编译器而异，有些机器它会访问内存位置零；有些机器将引发一个错误并终止程序。指针变量不能自动被初始化为NULL。

## 8 指针和数组

### **规范8.3 不要返回局部变量的地址。**

局部变量是在栈中分配的，函数返回后占用的内存会释放，继续使用这样的内存是危险的。因此，应该避免出现这样的危险。例：

```
extern void nvdfunc1( INT16U * );
INT16U *npuiVar;
void nvdfunc1(INT16U * apuiArg )
{
    npuiVar = apuiArg; /* NG */
}
```

### **规范8.4 字符数组的定义和初始化要考虑'\0'。**

例：`char hello[5] = "Hello"; /* NG */`

### **规范8.5 指针指向的内存被释放后，该指针要立刻赋值为NULL。**

防止指向内存的指针，在内存释放后，被错误的重新使用。

# 9 结构与联合

**规范9.1 对于所有结构与联合的类型，其内部的所有成员必须完整的指定。**

例：

```
struct stag{
    INT16U uiMvar;
    INT8U ucMvar[]; /* NG：未指定数组大小 */
}
```

**规范9.2 结构体的成员必须有名字，必须经过该名字进行结构体成员的访问。**

**规范9.3 在保证可读性的前提下，合理排列结构中成员的顺序（按照由小到大的顺序排列），可以提高内存利用率。**

# 10 预处理指令

**规范10.1** 文件中的`#include` 语句之前只能是其他预处理指令或注释。

**规范10.2** `#include` 指令中的头文件名字里不能出现非标准字符。

例：`#include "c:\temp\head.h" /* NG */`  
`#include "'head.h" /* NG */`

**规范10.3** 禁止头文件的重复包含，应采取防范措施。

通常的手段是为每个文件配置一个宏，当头文件第一次被包含时就定义这个宏，并在头文件被再次包含时使用它，以排除文件内容。

例：

一个名为`ahdr.h` 的文件可以组织如下

```
#ifndef AHDR_H
#define AHDR_H
...
#endif /* __AHDR_H*/
```

**规范10.4** 禁止在宏定义中出现`extern`、`static` 和`const` 这样的关键字。都可能导致非预期的行为，或者是非常难懂的代码。

# 10 预处理指令

**规范10.5 宏不能用于定义语句，或部分语句，除了do-while 结构。宏也不能重定义语言的语法。**

例：

```
#define CLOCK (XSTAL / 16) /* OK：定义表达式 */
#define PLUS2(X) ((X) + 2) /* OK：扩展表达式 */
#define READ_TIME_32()
do {
    ...
} while(0) /* example of do-while-zero */
#define INT32U long /* NG：使用宏定义了，应该使用typedef */
#define STARTIF if ( /* NG：定义部分语句 */
```

**规范10.6 增加调试语句应该放在调试开关内。**

调试代码不需要提交给客户，因此应该用宏封起来，避免与Release代码混在一起（定义调试开关的位置需要注意）。例：

```
# ifdef DEBUG_FUNC1 /* Modified by XXX on 05-10-10*/
    调试代码1；
    .....
# endif
```

# 附录：关于排版和注释

谢谢！

Neusoft