1.0 简介

代码编写规则应该在建立在一个工程项目之前。该规则应该贯穿整个项目的始终以保证代码的一致性。采用标准的代码编写惯例，可大大简化项目的维护负担。

在C语言中可以有多种代码的编写方法（当然其它编程序语言亦如此），你可以尽可能采用一种好的风格，以达到以下目的：

可移植　(Portability)
连贯　(Consistency)
整洁　(Neatness)
易于维护　(Easy maintenance)
易于理解　(Easy understanding)
简洁　(Simplicity)

不管你采用那种风格，我所强调的就是：这种风格一定要贯穿你项目的始终。在以后的内容中我还要提到：即使在一个团队合作的大型项目中，这种风格也要贯穿始终。采用通用的代码编写风格可以减轻代码维护的工作量并降低维护费用；这种通用的代码风格还可以避免重写代码。本文将介绍我几年来一直采用的这种编码风格。

2.00基本原则

制定标准的基本目的就是加强代码的可维护性。也就是说代码必须易于阅读，易于理解，易于测试，易于移植。

所有的代码必须采用ANSI C标准

函数原型必须采用ANSI标准，因此，类型定义应包含在括号之内。

保持代码的简单清晰

不要在语言中使用晦涩难懂的表述，直接表明你的思想。

保持一致性

尽可能使用同样的规则。

保持标准的灵魂(Keep the *spirit* of the standards.)

Where you have a coding decision to make and there is no direct standard, then you should always keep within the spirit of the standards.

避免使用复杂语句

一个语句若有太多的决策点（Decision points）将会使代码难于理解，尤其是对于测试。

不要使用GOTO语句

更新原有代码

一旦修改已存在的代码，就要随时更新相关文档以遵守本文中所制定的规则，这将确保原有代码即时更新。

3.00 源代码文件

代码行的长度

我不喜欢超过80个字符宽度的C程序代码行，只是因为我们过去的显示器只允许显示80个字符的宽度。 一代码行的长度应考虑在一张8.5″ X 11″的打印纸上以每英寸17个字符的压缩打印模式打印的字符数量。在压缩模式下，你可以调整到132个字符并在页面左侧预留足够空间的装订线位置。允许每行132个字符是避免有注释的源代码折行，如果确实需要超过132个以上的字符才能使你的代码更清晰，那也未尝不可。我曾经写过超过300个字符宽度的初始化结构的代码（存于ROM中）。当然，你可能看不到完整表格的内容，但至少会整齐的多。

[TAB]键的应用

一定不要用[TAB]键（ASCII 0x09）。若使代码缩排一定要用空格键[SPACE]（ASCII0x20）。[TAB]键会在不同的电脑或打印机上显示会有所不同，避免这种现象出现，就是使用空格键代替[TAB]键。

缩行步长： 4个空格

代码缩行步长以4个空格（ASCII 字符0x20）组成。注意“case”语句实际上应缩行5个空格。

包含一个文件头

在每一个源代码文件的开始部分都应包含一个注释块，其内容应包括公司名称，地址，版权，程序员，文件的描述信息等，参见以下示例：

```
/*
*********************************************************************************
* Micrium, Inc.
* 949 Crestview Circle
* Weston, FL 33327
*
* (c) Copyright 2000, Micrium, Inc., Weston, FL
*
* All rights reserved. Micrium's source code is an unpublished work and the
* use of a copyright notice does not imply otherwise. This source code contains
* confidential, trade secret material of Micrium, Inc. Any attempt or participation
* in deciphering, decoding, reverse engineering or in any way altering the source
* code is strictly prohibited, unless the prior written consent of Micrium, Inc.
* is obtained.
*
* Filename :
* Programmer(s): Joe Programmer (JP)
* John Doe (JD)
* Created : YYYY/MM/DD
* Description :
*********************************************************************************
*/
```

一个完整的程序文件是一个包含可执行语句文件，而头文件则不是。这两种文件看起来很相像，如下所示。一个程序文件将包含部分或全部以下内容。

执行文件的布局:

File heading （文件头）

Revision history （版本更新纪录）

```
#include            (包含文件)
#define constants   (常量定义)
Macros              (宏定义)
Local data types    (局部数据类型定义)
Local variables     (局部变量)
Local tables        (局部表单)
Local function prototypes(局部函数原型)
Global functions    (全局函数)
Local functions     (局部函数)
```

头文件布局:

```
File heading        (文件头)
Revision history    (版本更新纪录)
#define constants   (常量定义)
Global macros       (全局宏定义)
Global data types   (全局数据类型定义)
Global variables    (全局变量)
Externals           (外部定义)
Global function prototypes(全局函数原型)
```

分隔各主要段落

每段应以一个以下示例的注释块开始:

```
/*
*********************************************************************************
* DATA TYPES
*********************************************************************************
*/
typedef unsigned char BOOLEAN;

/*
*********************************************************************************
* PROTOTYPES
*********************************************************************************
*/
BOOLEAN OSIsTaskRdy(void);
```

头文件必须保证避免在包含文件中重复定义一个数值。

推荐使用**#ifndef X.**代替**!defined(X)**

```
#if       !defined(module_H)
#define   module_H
          Body of the header file.
#endif /* End of module_H */
```

4.00 注释

Make every comment count.

保持代码与注释分隔清晰可见。

尽量少在语句间嵌入注释，永远不要将注释加在如下所示的代码之上。这会使代码难于理解，因为喧宾夺主的注释会转移对代码的阅读。

```
void ClkUpdateTime (void)
{                                          /* DO NOT comment like this! */
    /* Update the seconds */
    if (ClkSec >= CLK_MAX_SEC) {
        ClkSec = 0;
        /* Update the minutes */
        if (ClkMin >= CLK_MAX_MIN) {
            ClkMin = 0;
            /* Update the hours */
            if (ClkHour >= CLK_MAX_HOURS) {
                ClkHour = 0;
            } else {
                ClkHour++;
            }
        } else {
            ClkMin++;
        }
    } else {
        ClkSec++;
    }
}
```

不要将单句注释分隔成多行。

从来也不要这样做：

```
/* This type of comment can lead to confusion especially when describing a function like
ClkUpdateTime (). The function looks like actual code! */
```

使用注释块分隔代码段。

参见以下注释块。注意一个注释块的主题应以大写字符形式居中。

```
/*
*********************************************************************************
* VARIABLES
*********************************************************************************
*/
```

Use trailing comments as much as possible.

As much as possible, always start the trailing comment on the same column. If the code goes beyond the selected column, place the comment on the line just above while still starting at the same column. As much as possible, line up the terminating comment charaters. Using trailing comments allows the code to be visually separate from the code.

```
void ClkUpdateTime (void)
{
    if (ClkSec >= CLK_MAX_SEC) {                /* Update the seconds */
        ClkSec = 0;
        if (ClkMin >= CLK_MAX_MIN) {            /* Update the minutes */
            ClkMin = 0;
            if (ClkHour >= CLK_MAX_HOURS) {     /* Update the hours */
                ClkHour = 0;
            } else {
                ClkHour++;
            }
        } else {
            ClkMin++;
        }
    } else {
        ClkSec++;
    }
}
```

使用 #if 0 和 #endif 注释代码块。

注释不可以嵌套，使用**#if 0** 和 **#endif**来注释一段较大的代码

```
#if 0                           /* Comments out the following code */
#define   DISP_TBL_SIZE   5       /* Size of display buffer table */
#define   DISP_MAX_X     80       /* Max. number of characters in X axis */
#define   DISP_MAX_Y     25       /* Max. number of characters in Y axis */
#define   DISP_MASK    0x5F
#endif
```

5.00 命名规则

通用规则

```
#define   constants:
#define   macros:
typedefs:
enum tags:
    用下划线分隔所有大写的单词.
    例如: DISP_BUF_SIZE, MIN(), MAX(), etc.
局部变量(函数作用域):
    全部使用小写单词并以下划线分隔.
    使用标准变量名称(也就是: i, j, k 用于loop循环计数, p 用于指针变量等.)
一个文件作用域内的变量:
    以模块名称跟一个下划线为前缀,单词间以首字母大写分隔
    静态变量声明
    例如: Disp_Buf[], Comm_Ch, 等.
全局变量:
    以模块名称作为变量的前缀.
    如果是多个单字组成的名称每个单前缀字母要大写
    例如: DispMapTbl[], CommErrCtr, etc.
局部函数:
    以模块名称与下划线为前缀,单词间首字母大写.
    静态声明:
    例如: static void Comm_PutChar()
全局函数:
    以模块名称为前缀单词间首字母大写.
    例如: void CommInit()
```

使用首字母大写分隔名称中的单字(e.g. DispBuf[]).

原有代码中只含有小写字符的必须以下划线分隔(i.e. ‘_’, ASCII 字符0x2D).

使用规范化的缩略词/缩写词助记符。

建立一个标准规范的缩略词/缩写词和助记符詞典应用于整个项目。以下表格所列的是部分首字缩略詞/缩写词助记符字典的示例，另外在再做成一个相同内容的反向排序的缩略詞/缩写词助记符字典列表。

缩略词/缩写词和助记符使用“模块-对象-操作”的格式。

当创建一个全局的常量，变量和函数标示符的时候，先指定一个模块(或者子系统)的名称并跟随其对象和动作。参见以下示例：，

```
OSSemPost()
OSSemPend()
etc.
```

首字缩略/或缩写词助记符字典		
英文原词	中译文	首字略词或缩写词
Argument	自变量	Arg
Buffer	缓冲区	Buf
Clear	清除	Clr
Clock	时钟	Clk

Compare	比较	Cmp
Configuration	配置	Cfg
Context	上下文	Ctx
Delay	延迟	Dly
Device	设备	Dev
Disable	禁止	Dis
Display	显示	Disp
Enable	使能	En
Error	错误	Err
Function	函数	Fnct
Hexadecimal	16进制	Hex
High Priority Task	高优先级任务	HPT
I/O System	输入输出系统	IOS
Initialize	初始化	Init
Mailbox	信箱	Mbox
Manager	管理器	Mgr
Manual	手册	Man
Maximum	最大	Max
Message	消息	Msg
Minimum	最小	Min
Multiplex	复合	Mux
Operating System	操作系统	OS
Overflow	益出	Ovf
Parameter	参数	Param
Pointer	指针	Ptr
Previous	前	Prev
Priority	优先级	Prio
Read	读	Rd
Ready	待命	Rdy
Register	寄存器	Reg
Schedule	调度	Sched
Semaphore	信号	Sem
Stack	堆栈	Stk
Synchronize	同步	Sync
Timer	定时器	Tmr
Trigger	触发器	Trig
Write	写	Wr

坚持使用标准的缩略词/缩写词或词助记符

尽管你可以书写完整的单词，但也要坚持使用缩略词/缩写词或助记符。

例如：总是用Init代替Initialize。

6.00 数据类型

所有的数据类型必须使用大写字符形式声明

单词与单词之间的分隔使用下划线字符（ASCII 字符0x2D）

使用以下可移植的数据类型。

避免使用“char”以外所有标准C的数据类型，因为其数据长度不可移植。以下示例中INT??与FP??数据类型基于不同的处理器和不同的编译器应有不同的定义。

```
typedef unsigned char BOOLEAN;              /* Logical data type (TRUE or FALSE)    */
typedef unsigned char INT8U;                /* Unsigned 8 bit value                 */
typedef signed char INT8S;                  /* Signed 8 bit value                   */
typedef unsigned short INT16U;              /* Unsigned 16 bit value                */
typedef signed short INT16S;                /* Signed 16 bit value                  */
typedef signed short INT32U;                /* Unsigned 32 bit value                */
typedef signed short INT32S;                /* Signed 32 bit value                  */
typedef signed short INT64U;                /* Unsigned 64 bit value (if available) */
typedef signed short INT64S;                /* Signed 64 bit value (if available)   */
typedef float FP32;                         /* 32 bit, single prec. floating-point  */
typedef double FP64;                        /* 64 bit, double prec. floating-point  */
```

结构体和共用体必须定义类型.

所有的结构体和共用体必须定义成以下所示。其中数据类型必须全部使用大写字母。

```
typedef struct {
    char RxBuf[COMM_RX_SIZE];               /* Storage of characters received      */
    char *RxInPtr;                          /* Pointer to next free loc. in buffer */
    char *RxOutPtr;                         /* Pointer to next char. to extract    */
    INT16U RxCtr;                           /* Number of characters in Rx buffer   */
    char TxBuf[COMM_TX_SIZE];               /* Storage for characters to send      */
    char *TxInPtr;                          /* Pointer to next free loc. in Tx Buf */
    char *TxOutPtr;                         /* Pointer to next char to send        */
    INT16U TxCtr;                           /* Number of characters left to send   */
} COMM_BUF;
```

结构体队列（ Structure alignment）

每一个成员的数据类型缩进4个空格，同时结构体成员名称互相对齐；注释部分起始列也要垂直对齐。

数据定义的作用域

如果一个数据类型仅应用于一个文件中，他必须在该文件中声明；如果数据类型全局的，他必须在被包含的头文件中声明。

7.00 代码编排

每个语句行只做一件事

用以下两句

```
DispSegTblIx = 0;
DispDigMsk   = 0x80;
```

代替这两句：

```
DispSegTblIx = 0; DispDigMsk = 0x80;
```

用空行或注释将较大的代码块分隔开来

以下几个操作符不可使用空格符分隔：

-> 结构体指针操作符p->member

. 结构体成员操作符s.member

[] 数组下标a[i]

函数名后的括号不可以有空格符

```
DispInit();
```

函数参数的每个逗号后面至少要有一个空格符分隔。

例如： `DispStr(x, y, s);`

每个分号后面至少要有一个空格。

```
for (I = 0; i < 10; i++)
```

一元操作符与操作数之间不可以包含空格。

例如：

```
!value
~bits
++i
j--
(INT32U)x
*ptr
&x
sizeof(x)
```

二元和多元操作符与操作数之间至少要有一个空格。

```
c1 = c2;
x + y
i += 2;
n > 0 ? n : -n;
a < b
c >= 2
```

以下关键字后面要跟随一个空格符：

```
if (a > b)
while (x > 0)
for (i = 0; i < 10; i++)
} else {
switch (x)
return (y);
```

在赋值语句行中的等号(“=”)及数值要垂直对齐以保持整齐.

整型及浮点型数值的低位对齐。

```
DispSegTblIx =    0;
DispDigMsk   = 0x80;
DispScale    =    1.25;
```

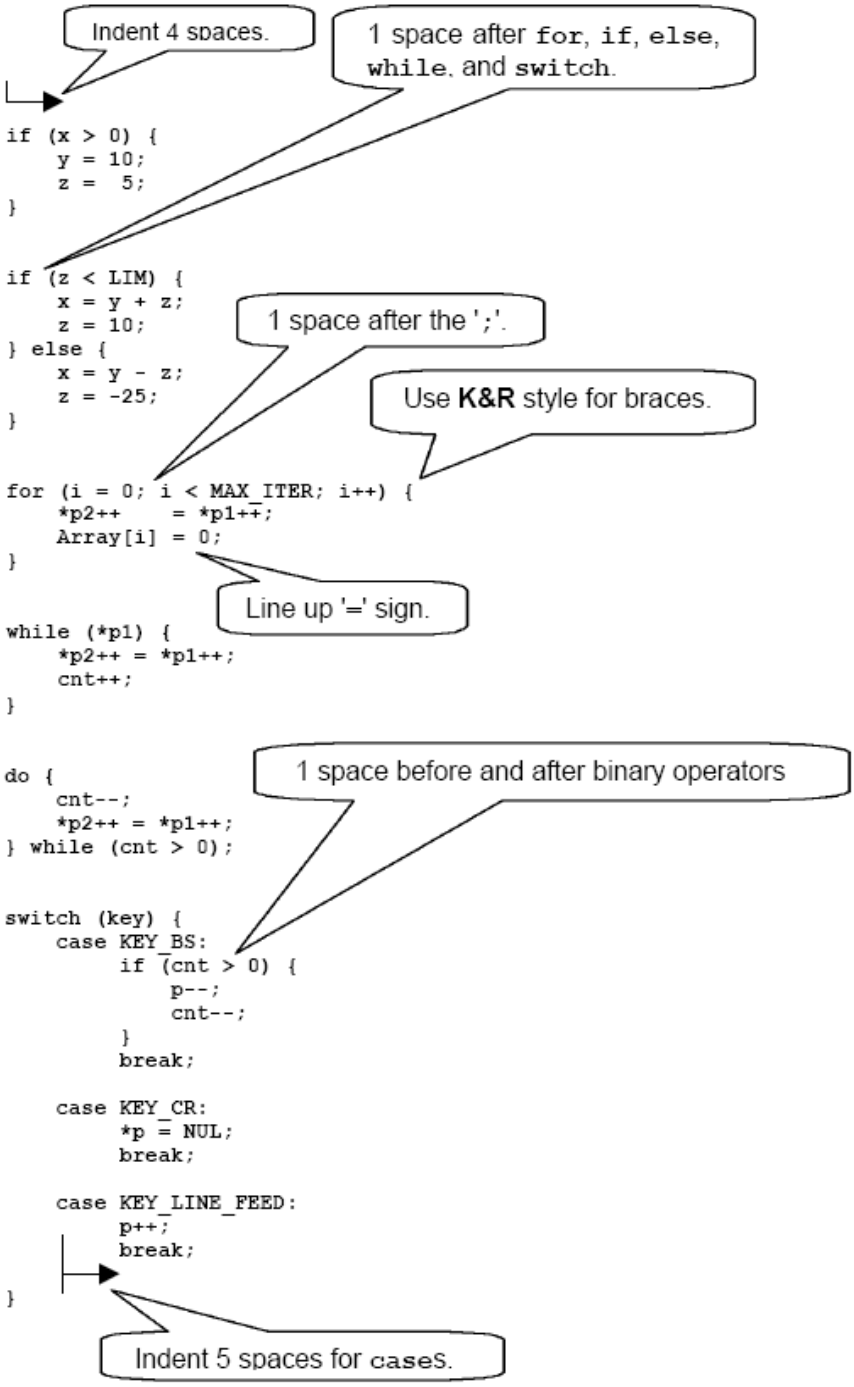
括号表达式开始符和结束符与括号内的字符间不要有空格：

```
x = (a + b) * c;
```

8.00 Constructs

The following construct style should be use.

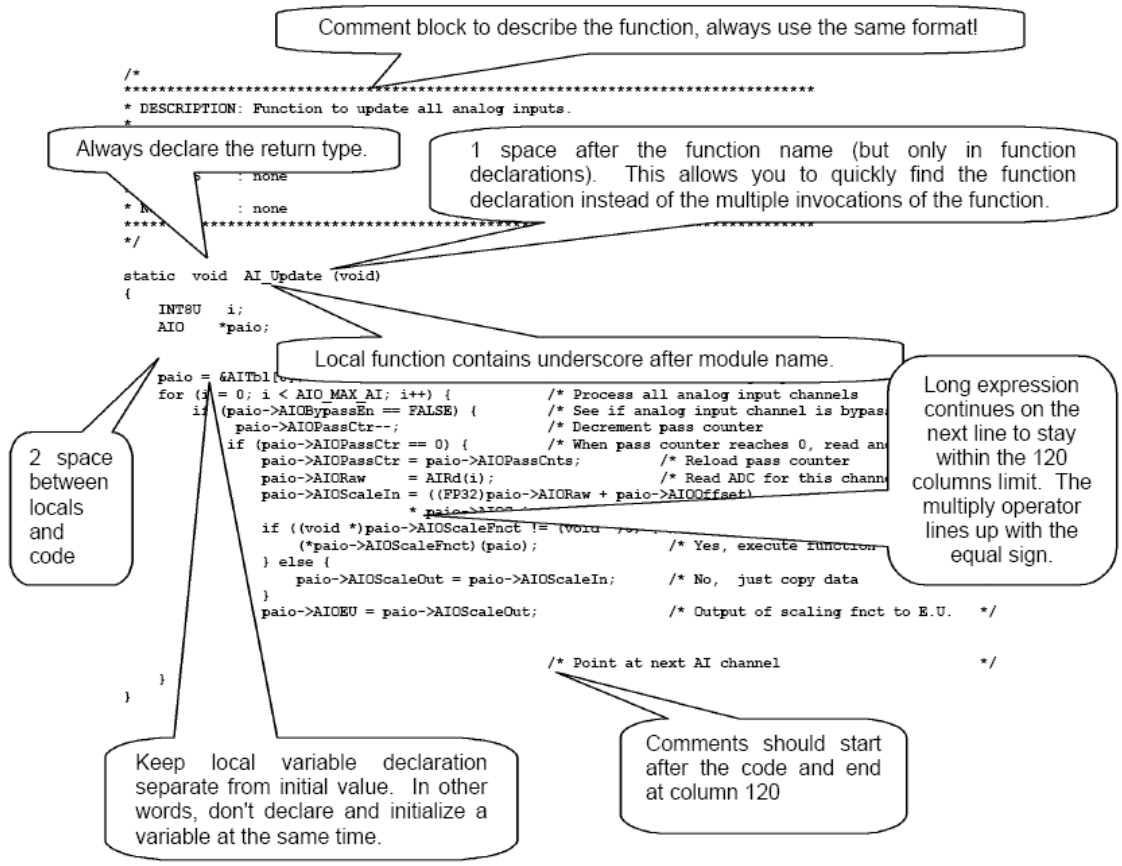
Indentation is 4 spaces.
TABs MUST not be used.
Always use braces, even for null statements.
Use K&R style for braces.




```
if (x > 0) {
    y = 10;
    z =  5;
}

if (z < LIM) {
    x = y + z;
    z = 10;
} else {
    x = y - z;
    z = -25;
}

for (i = 0; i < MAX_ITER; i++) {
    *p2++    = *p1++;
    Array[i] = 0;
}

while (*p1) {
    *p2++ = *p1++;
    cnt++;
}

do {
    cnt--;
    *p2++ = *p1++;
} while (cnt > 0);

switch (key) {
    case KEY_BS:
         if (cnt > 0) {
             p--;
             cnt--;
         }
         break;

    case KEY_CR:
         *p = NUL;
         break;

    case KEY_LINE_FEED:
         p++;
         break;
}
```

8.00 Functions

The format of a function should be as shown below.




```
/*
*********************************************************************************
* DESCRIPTION: Function to update all analog inputs.
*
* A                : none
*
* R                : none
*********************************************************************************
*/

static  void  AI_Update (void)
{
    INT8U   i;
    AIO    *paio;

    paio = &AITbl[0];
    for (i = 0; i < AIO_MAX_AI; i++) {          /* Process all analog input channels
        if (paio->AIOBypassEn == FALSE) {       /* See if analog input channel is bypas
            paio->AIOPassCtr--;                 /* Decrement pass counter
            if (paio->AIOPassCtr == 0) {        /* When pass counter reaches 0, read an
                paio->AIOPassCtr = paio->AIOPassCnts;       /* Reload pass counter
                paio->AIORaw     = AIRd(i);                 /* Read ADC for this chann
                paio->AIOScaleIn = ((FP32)paio->AIORaw + paio->AIOOffset)
                                 * paio->AIOG
                if ((void *)paio->AIOScaleFnct != (void *)0) {
                    (*paio->AIOScaleFnct)(paio);            /* Yes, execute function
                } else {
                    paio->AIOScaleOut = paio->AIOScaleIn;   /* No,  just copy data
                }
                paio->AIOEU = paio->AIOScaleOut;            /* Output of scaling fnct to E.U.   */
            }

            paio++;                                         /* Point at next AI channel         */
        }
    }
}
```