嵌入式C语言代码规范

510实验室

目录

[前言 4](#_Toc103593792)

[参考文献 4](#_Toc103593793)

[1. 总体原则 5](#_Toc103593794)

[1.1. 清晰第一 5](#_Toc103593795)

[1.2. 简洁为美 5](#_Toc103593796)

[1.3. 风格统一 5](#_Toc103593797)

[2. 排版风格 5](#_Toc103593798)

[2.1. 缩进 5](#_Toc103593799)

[2.2. 空行 5](#_Toc103593800)

[2.3. 代码行 6](#_Toc103593801)

[2.4. 空格 6](#_Toc103593802)

[2.5. 成对书写 6](#_Toc103593803)

[2.6. 对齐 7](#_Toc103593804)

[2.7. 注释 7](#_Toc103593805)

[3. 命名规范 7](#_Toc103593806)

[3.1. 文件名 7](#_Toc103593807)

[3.2. 变量名 8](#_Toc103593808)

[3.3. 宏定义/枚举 8](#_Toc103593809)

[3.4. 函数名 8](#_Toc103593810)

[4. 注释规范 8](#_Toc103593811)

[4.1. 注释符号 8](#_Toc103593812)

[4.2. 文件注释 9](#_Toc103593813)

[4.3. 类/结构体注释 9](#_Toc103593814)

[4.4. 枚举注释 9](#_Toc103593815)

# 前言

“代码是写给人看的，不是写给机器看的，只是顺便计算机可以执行而已”

作为一个软件开发人员，应养成良好的编程习惯，随着编码越来越多，内容也会变得越来越多，规范化自己的编程有助于为了在程序代码量很大的时候，便于自己阅读，也便于别人阅读（团队合作），特别是作为一个合格的开发者，更需要规范自身写的程序代码，形成一种良好的习惯。

在C语言中不遵守编译器的规定，编译器在编译时就会报错，这个规定叫作规则。但是有一种规定，它是一种人为的、约定成俗的，即使不按照那种规定也不会出错，这种规定就叫作规范。

# 参考文献

《代码整洁之道》

《华为C编程规范》

《MISRA C编码规范标准》

《计算机程序的构造和解释》

# 1. 总体原则

## 1.1. 清晰第一

清晰性是易于维护、易于重构的程序必需具备的特征，一般情况下，代码的可阅读性高于性能，只有确定性能是瓶颈时，才应该主动优化。

## 1.2. 简洁为美

代码越简单越好，代码一行只做一件事，一个函数只完成一个功能等，代码越长越难以看懂，也就越容易在修改时引入错误。

## 1.3. 风格统一

因为软件开发不可避免要维护他人代码或者和他人合作开发，此时就需要尽可能和与原有代码风格保持一致或者和合作者达成共识的代码风格，否则每个人都使用自己的代码风格，在阅读时变得十分吃力。

# 2. 排版风格

代码规范化中的排版基本上有七大风格，主要体现在缩进、空行、代码行、空格、成对书写、对齐、注释七方面的书写规范上。

## 2.1. 缩进

程序块采用缩进风格编写，缩进可以使程序更有层次感，每次缩进一般为 4 个空格（部分要求为一个制表符Tab）。

原则是：如果地位相等，则不需要缩进；如果属于某一个代码的内部代码就需要缩进。

## 2.2. 空行

空行起着分隔程序段落的作用。空行得体将使程序的布局更加清晰。空行不会浪费内存，虽然在文档中比较多，但是值得。

1、尽可能在定义变量的同时初始化该变量，即遵循就近原则；如果变量的引用和定义相隔比较远，那么变量的初始化就很容易被忘记。若引用了未被初始化的变量，就会导致程序出错。

2、每个函数定义结束之后都要加空行。

3、相对独立的程序块之间、变量说明之后必须加空行。比如上面几行代码完成的是一个功能，下面几行代码完成的是另一个功能，那么它们中间就要加空行。这样看起来更清晰。

## 2.3. 代码行

1、一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且便于写注释。

2、if 、 for 、 do 、 while 、 case 、 switch 、 default 等语句独占一行，执行语句不得紧跟其后（不论执行语句有多少行，就算只有一行也要加{}，并且遵循对齐的原则，这样可以防止书写失误）。

3、一条语句不能过长，如不能拆分需要分行写， 80/132是VTY常见的行宽值，我们使用80为语句长度。

## 2.4. 空格

1、关键字之后要留空格。像 const、case 等关键字之后至少要留一个空格，否则无法辨析关键字。像 if、for、while 等关键字之后应留一个空格再跟左括号(，以突出关键字。

2、函数名之后不要留空格，应紧跟左括号(，以与关键字区别。

3、(向后紧跟；， )、,、; 这三个向前紧跟，紧跟处不留空格。

4、,之后要留空格，如果;不是一行的结束符号，其后要留空格。

5、赋值运算符、关系运算符、算术运算符、逻辑运算符、位运算符，如 =、==、!=、+=、-=、\*=、/=、%=、>>=、<<=、&=、^=、|=、<、<=、>、>=、+、－、\*、/、%、&、|、&&、||、<<、>>、^ 等双目运算符的前后应当加空格（运算符“%”是求余运算符，与 printf 中 %d 的“%”不同，所以 %d 中的“%”前后不用加空格）。

6、单目运算符 !、~、++、--、－、\*、& 等前后不需要加空格。

7、数组符号[ ]、结构体成员运算符.、指向结构体成员运算符->，这类操作符前后不加空格。

8、对于表达式比较长的 for 语句和 if 语句，为了紧凑起见，可以适当地去掉一些空格。但 for 和 if 后面紧跟的空格不可以删，其后面的语句可以根据语句的长度适当地去掉一些空格。例如：for (i=0; i<10; i++)，即 for 和分号后面保留空格，=和<前后的空格去掉。

## 2.5. 成对书写

成对的符号一定要成对书写，如 ()、{}。不要写完左括号然后写内容最后再补右括号，这样很容易漏掉右括号，尤其是写嵌套程序的时候。

## 2.6. 对齐

对齐主要是{}方面的。

1、{ 和 }分别都要独占一行。互为一对的{和}要位于同一列，并且与引用它们的语句左对齐。

2、{ }之内的代码要向内一个缩进，且同一地位的要左对齐，地位不同的继续缩进。

## 2.7. 注释

C语言中一行注释一般采用//…，多行注释必须采用 /\*…\*/，或者统一采用/\*…\*/。注释通常用于重要的代码行或段落提示。在一般情况下，源程序有效注释量必须在 20% 以上。虽然注释有助于理解代码，但注意不可过多地使用注释，注释太多会让人眼花缭乱。

1、注释用使用对代码的功能做解释，并不是说明是怎么做的。

2、对于一些巧妙地、用特殊方式实现功能的代码，可以使用注释说明这样做的目的或好处等。

3、代码十分明确的，一目了然的，则不必增加注释，否则就是多余的注释，如 int minValue = 5; // 定义最小值为5。

4、边写代码边注释，修改代码的同时要修改相应的注释，以保证注释与代码的一致性，不再有用的注释要删除。

5、每一条宏定义的右边必须要有注释，说明其作用。

6、相关函数和结构体等需要注释，具体注释方式请参考文档中的“注释规范”内容。

# 3. 命名规范

标识符的命名规则历来是一个敏感话题，典型的命名风格如unix风格、windows风格等，从来无法达成共识。实际上，各种风格都有其优势也有其劣势，而且往往和个人的审美观有关。对标识符定义主要是为了让团队的代码看起来尽可能统一，有利于代码的后续阅读和修改。

命名完全体现了程序的可阅读性和可理解性，在一定程度上是不需要写注释也能看懂代码。

## 3.1. 文件名

因为不同系统对文件名大小写处理会有所不同（如Windows系统不区分大小写，但是Linux系统则区分）。

与库函数区分，比如，自定义的串口可以加上my或者串口使用的功能修饰如：mqtt\_usart，zigbee\_usart等。

## 3.2. 变量名

变量命名可以是：属性+类型+描述。

1、使用名词或者形容词 + 名词方式命名变量（描述）。

2、禁止使用单字节命名变量，但允许定义 i、j、k等作为局部循环变量。

如：int error\_number;或 int err\_num; 。

## 3.3. 宏定义/枚举

1、对于数值或者字符串等等常量或者枚举的定义，全部采用全大写字母，单词之间加下划线“\_”的方式命名。

如：#define FIRMWARE\_VERSION (0x0102)

2、除了头文件或编译开关等特殊标识符定义，宏定义不能使用下划线“\_”开头和结尾（一般来说，下划线开头或结尾的宏都是一些内部的定义）。

## 3.4. 函数名

1、函数名通常以函数要执行的动作进行命名，所以一般采用动词或者动词+名词的结构。

2、可以对函数名进行大小写处理。

如：printEmployeePaychecks();或者Print\_Employee\_Paychecks();

# 4. 注释规范

注释原则是有助于对程序的阅读理解以及提供二次开发所需文档，注释的方式有很多。遵循原则为，说明性文件、函数接口必须充分注释说明。全局变量需要说明功能及取值范围，需要自行处理资料函数需要加上使用警告信息。

## 4.1. 注释符号

C语言中一行注释一般采用//…，多行注释必须采用 /\*…\*/，或者统一采用/\*…\*/。

## 4.2. 文件注释

文件注释通常放在整个文件开头，如说明文件名、作者、日期、描述或版本等诸多信息。

/\*\*

\* @file 文件名

\* @brief 简介

\* @details 细节

\* @mainpage 工程概览

\* @author 作者

\* @email 邮箱

\* @version 版本号

\* @date 年-月-日

\* @license 版权

\*/

或者可以使用更少的信息，只体现函数作用，相关变量信息等。

## 4.3. 类/结构体注释

类或者结构体定义的注释方式非常简单，使用@brief后面填写类的概述，换行填写类的详细信息。

/\*\*

\* @brief ZigBee通讯控制状态及通讯信息结构体

\*/

typedef struct

{

FunctionalState Recvstatus; /\* ZIGBEE接收状态 \*/

ZIGBEE\_StatusEnum Status; /\* ZIGBEE通讯状态 \*/

uint8\_t ZIGBEE\_SendBuffer[300]; /\* ZIGBEE发送数据 \*/

ZIGBEE\_DataRecvTypedef ZIGBEE\_Recv; /\* ZIGBEE接收数据 \*/

ZIGBEE\_ParamTypedef ZIGBEE\_Param; /\* ZIGBEE参数配置 \*/

} ZIGBEE\_InfoTypedef;

## 4.4. 枚举注释

/\*\*

\* @brief 使能/失能枚举定义

\*/

typedef enum

{

DISABLE = 0, /\* 禁止/失能 \*/

ENABLE = !DISABLE /\* 允许/使能 \*/

} FunctionalState;