框架编码规范

Framework Code specification based on C language development

整理:Yearnext

February 5, 2018

Contents

1 前言

技术人员设计程序的首要目的是用于技术人员沟通和交流，其次才是用于机器执行。程序的生命力在于用户使用，程序的成长在于后期的维护及根据用户需求更新和升级功能。如果你的程序只能由你来维护，当你离开这个程序时，你的程序也和你一起离开了，这将给公司和后来接手的技术人员带来巨大的痛苦和损失。

为提高产品代码质量，指导嵌入式软件开发人员编写出简洁、可维护、可靠、可测试、高效、可移植的代码，编写了本规范。

1.1 主要内容

一. 为形成统一编程规范，从编码形式角度出发，本规范对格式与排版、注释、标示符命名等方面进行了详细阐述。

二. 为编写出高质量嵌入式软件，从嵌入式软件安全及可靠性出发，本规范对由于C语言标准、C语言本身、C编译器及个人理解导致的潜在危险进行说明及规避。

本规范适用于嵌入式软件的开发，也对其他嵌入式软件开发起一定的指导作用。

1.2 规则术语介绍

原则：编程时必须坚持的指导思想。
规则：编程时需要遵循的约定，分为强制和建议（强制是必须遵守的，建议是一般情况下需要遵守，但没有强制性）。
说明：对原则/规则进行必要的解释。
实例：对此原则/规则从正、反两个方面给出例子。
材料：扩展、延伸的阅读材料。

1.3 规则形式

规则/原则<序号>(规则类型)：规则内容 [原始参考]

1. 序号：每条规则都有一个序号，序号是按照章节目录的形式，从数字1开始。
2. 规则类型：“强制”或者是“建议”。
3. 规则内容：此条规则的具体内容。
4. 原始参考：指示了产生本条款或本组条款的可应用的主要来源。

1.4 C语言术语介绍

声明(declaration)：指定了一个变量的标识符，用来描述变量的类型。声明，用于编译器(compiler)识别变量名所引用的实体。

定义(definition)：是对声明的实现或者实例化。连接器(linker)需要它（定义）来引用内存实体。

1.5 命名规则介绍

帕斯卡命名法：帕斯卡名法就是当变量名或函式名是由一个或多个单词连结在一起，而构成的唯一识别字时，每一个单词的首字母都采用大写字母。

小驼峰命名法：驼峰命名法就是当变量名或函式名是由一个或多个单词连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始，第二个单词的首字母大写或每一个单词的首字母都采用大写字母。

匈牙利命名法：匈牙利命名法通过在变量名前面加上相应的小写字母的符号标识作为前缀，标识出变量的作用域、类型等。这样做的好处在于能增加程序的可读性，便于对程序的理解和维护。匈牙利命名法关键是：标识符的名字以一个或者多个小写字母开头作为前缀；前缀之后的是首字母

大写的一个单词或多个单词组合，该单词要指明变量的用途（标识符名=属性+类型+对象描述）。

2 排版规范

2.1 缩进

规则 <2-1>（强制）：程序块要采用缩进风格编写，缩进的空格数为4。

[说明]

函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case语句下的情况处理语句也要遵从语句缩进要求（对于由开发工具自动生成的代码可以有不一致）。

2.2 空格空行

规则 <2-2-1>（强制）：进行双目运算、赋值时，操作符之前、之后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如”.”），后不应加空格。

[示例]

```
//! 反例
uint8_t func(void)
{
    struct
    {
        uint8_t a;
        uint8_t b;
    }var;

    var.a=0;
    var.b=1;

    var.a=var.a+var.b;

    return var.a&var.b?1:0;
}
```

```
17
18      //! 示范代码
19      uint8_t func(void)
20      {
21          struct
22          {
23              uint8_t a;
24              uint8_t b;
25          }var;
26
27          var.a = 0;
28          var.b = 1;
29
30          var.a = var.a + var.b;
31
32          return (var.a & var.b) ? (1) : (0);
33      }
```

规则 <2-2-2>（强制）：相对独立的程序块之间、变量声明/定义之后必须加空行。

[示例]

```
1       //! 反例
2       uint8_t func(void)
3       {
4           uint8_t a = 0;
5           uint8_t b = 1;
6           a++;b++;
7           return a*b;
8       }
9
10      //! 示范代码
11      uint8_t func(void)
12      {
13          uint8_t a = 0;
14          uint8_t b = 0;
```

```
15
16          a++;
17          b++;
18
19          return a*b;
20      }
```

规则<2-2-3>（强制）：较长的语句(大于80字符)要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

[示例]

```
1       //! 反例
2       if (a && b || a && c || c && d || d || e && f)
3       {
4           //! code
5       }
6
7       //! 示范代码
8       if (    a && b
9       || a && c
10      || c && d
11      || d
12      || e && f)
13      {
14          //! code
15      }
```

规则 <2-2-4>（强制）：不允许把多个短语句写在一行中，即一行只写一条语句。

[示例]

```
1       //! 反例
```

```
    uint8_t a, b, c; char str;

    //! 示范代码
    uint8_t a, b, c;
    char str;
```

2.3 对齐

规则 <2-3-1>（建议）：多行同类操作时操作符尽量保持对齐。

[示例]

```
    int a, aa, aaa;

    //! 反例
    a = 0;
    aa = 0;
    aaa = 0;

    //! 示范代码
    a   = 0;
    aa  = 0;
    aaa = 0;
```

规则 <2.3-2>（强制）：if、 for、 do、 while、 case、 switch、 default 等语句自占一行，且 if、 for、 do、 while等语句的执行语句部分无论多少都要加括号。

[示例]

```
    //! 反例
    if(a)
        a--;
    else
```

```
5           a++;
6
7       //! 示范代码
8       if (a)
9       {
10          a--;
11      }
12      else
13      {
14          a++;
15      }
```

规则 <2-3-3>（建议）：一行程序以小于80字符为宜，不要写的过长。

3　注释规范

规则 <2-4-1>（强制）：注释的内容要清楚、明了，含义准确，在代码的功能、意图层次上进行注释。

[说明]
注释的目的是让读者快速理解代码的意图。
注释不是为了名词解释(what)，而是说明用途(why)。

规则 <2.2-2>（强制）：注释应分为两个角度进行，首先是应用角度，主要是告诉使用者如何使用接口（即你提供的函数），其次是实现角度，主要是告诉后期升级、维护的技术人员实现的原理和细节。

[说明]
每一个产品都可以分为三个层次，产品本身是一个层次，这个层次之下的是你使用的更小的组件，这个层次之上的是你为别人提供的服务。

你这个产品的存在的价值就在于把最底层的小部件的使用细节隐藏，同时给最上层的用户提供方便、简洁的使用接口，满足用于需求。从这个角度来看软件的注释，你应该时刻想着你写的注释是给那一层次的人员看的，如果是用户，那么你应该注重描述如何使用，如果是后期维护者，那么你应该注重原理和实现细节。

规则 <2.2-3>（强制）：修改代码时，应维护代码周边的注释，使其代码和注释一致，不再使用的注释应删除。

[说明]
注释的目的在于帮助读者快速理解代码使用方法或者实现细节，若注释和代码不一致，会起到相反的作用。建议在修改代码前应该先修改注释。

3.1 头文件排版

规则 <2-4> (强制)：头文件排版内容依次为版权声明、文件信息、包含的头文件、宏定义、类型定义、声明变量、声明函数，且各个种类的内容间空一行。

3.2 源文件排版

规则 <2-5> (强制)：源文件排版内容依次为版权声明、文件信息、包含的头文件、宏定义、具有外部链接属性的全局变量定义、模块内部使用的static变量、具有内部链接的函数声明、函数实现代码。且各个种类的内容间空三行。

4　标识符命名规范

5 项目版本号命名规范

5.1 版本类型介绍

版本类型	版本名称	版本介绍
测试版本		
	Alphal	内部测试版
	Beta	外部测试版
	M 版	Milestone,每个开发阶段的终结点的里程碑版本
	Trail	试用版（含有某些限制，如时间、功能，注册后也有可能变为正式版）
	RC版	Release Candidate，意思是发布倒计时，该版本已经完成全部功能并清除大部分的BUG。到了这个阶段只会除BUG，不会对软件做任何大的更改
	RTM版	Release To Manufactur，意思是发布到生产商，这基本就是最终的版本
	GA版	Generally Available，最终版
正式版本		
	Enhance	增强版或者加强版
	Full version	完全版
	Release	发行版，有时间限制
	Upgrade	升级版
	Retail	零售版
	Plus	增强版，不过这种大部分是在程序界面及多媒体功能上增强

5.2 版本号管理策略

1. 项目初版本时，版本号可以为 0.1 或 0.1.0，也可以为 1.0 或 1.0.0。
2. 当项目在进行了局部修改或 bug 修正时，主版本号和子版本号都不变，修正版本号加1。

3. 当项目在原有的基础上增加了部分功能时，主版本号不变，子版本号加1，修正版本号复位为0，因而可以被忽略掉。

4. 当项目在进行了重大修改或局部修正累积较多，而导致项目整体发生全局变化时，主版本号加1。

5.3 版本信息格式介绍

工程名.分支名.主版本号.子版本号.修正版本号.版本类型.编译时间

[示例]

ysf.stm32f1xx.0.0.1.alpha.201609271352

格式名称	参数介绍
工程名	ysf
分支名	stm32f1xx
主版本号	0
子版本号	0
修正版本号	1
版本类型	alpha
编译时间	201609271352

6 嵌入式软件安全性相关规范

7　参考文献