# 嵌入式C语言开发规范 V1.2.0版本

## 一、开发环境

主要为WIN10及以上版本，所用到的集成开发环境有：

|  |  |  |
| --- | --- | --- |
| **平台** | **集成开发环境** | **调试环境** |
| STM32 HAL库/  标准库 | MDK V5.35  (AC5 / --gnu / C99mode / USE Micro Lib)  STM32CubeMX 6.6.1 | 调试下载ST-Linkv2/Jlink-v8  程序下载  STM32CubeProgrammer  内存访问  STM32 ST-LINK Utility  固件烧录  STM32TrustedPackageCreator  变量监测  STM32CubeMonitor |
| Arduino | Arduino2.0.7 |  |

## 二、参考规范

进行开发时参考C99标准，主要用到的C99特性包括：变长数组VLA、restrict关键字、inline关键字、内联函数、\_Bool类型、可移植类型stdint.h和inttypes.h。

## 三、基本排版格式

1. 需要以4个空格为单位的缩进;
2. 不适用Tab键进行缩进；
3. UTF-8编码格式；
4. **所有注释，用英文编写，并且格式：/\*xxx\*/不允许使用//**

**中文注释在不同版本IDE当中会出现编码错误，导致出现乱码**

## 四、文件夹结构

对于STM32标准库开发和Arduino开发，文件夹结构如下：

|  |  |  |
| --- | --- | --- |
| **母文件夹** | **子文件夹** | **作用** |
| Doc | 无 | 用来存放程序说明的文件，一般以Readme.txt命名  内容包括项目简介、代码层次、文件夹和文件介绍、  开发环境介绍（IDE名称及版本、编译器名称及版本、硬件平台信息、内存空间组织：堆栈尺寸及起始地址、固件库版本、调试设置、公共宏）  使用的引脚及宏定义、使用的外设作用和参数设置  **以英文书写** |
| Lib | 无 | 固件库文件 |
| Listing | 无 | 编译器编译时候产生的C/汇编/链接文件 |
| Output | 无 | 存放编译产生的调试信息、hex文件、预览信息、封装库灯 |
| Project | 无 | 用来存放工程 |
| User | Common | 通用文件：包括数据类型重定义文件、数据类型转换文件 |
| Hardware | 驱动文件：主要是外设操作函数的封装，驱动模块 |
| Function | 功能文件：主要是对驱动文件的封装，使用模块实现某种功能 |
| Task | 应用文件：主要是调用功能文件中功能函数，实现应用 |
| Config | 设置文件：包括 |

对于STM32 HAL库开发，自定义文件夹结构如下：

|  |  |  |
| --- | --- | --- |
| **母文件夹** | **子文件夹** | **作用** |
| Doc | 无 | 用来存放程序说明的文件，一般以Readme.txt命名  内容包括项目简介、代码层次、文件夹和文件介绍、  开发环境介绍（IDE名称及版本、编译器名称及版本、硬件平台信息、内存空间组织：堆栈尺寸及起始地址、固件库版本、调试设置、公共宏）  使用的引脚及宏定义、使用的外设作用和参数设置  **以英文书写** |
| User | Common | 通用文件：包括数据类型重定义文件、数据类型转换文件 |
| Hardware | 驱动文件：主要是外设操作函数的封装，驱动模块 |
| Function | 功能文件：主要是对驱动文件的封装，使用模块实现某种功能 |
| Task | 应用文件：主要是调用功能文件中功能函数，实现应用 |

其余文件夹由STM32CubeMX自动生成，无需管理

## 五、文档与注释

1. 对项目工程要有工程说明，工程说明放在Doc文件夹下，工程说明包括如下内容：文件夹文件说明、编程环境说明、修改版本记录；
2. 关于注释长度没有具体限制，只要能提供帮助，就尽可能地注释；
3. 注释应该解释代码为什么要这么做，而不是如何去做(代码本身已经表明了如何去做)；
4. 所有代码，空闲时中文注释与英文注释各一份，忙碌时：

**用英文编写，并且格式：/\*xxx\*/不允许使用//**

**中文注释在不同版本IDE当中会出现编码错误，导致出现乱码**

## 六、文件结构

每个 C++/C 程序通常分为两个文件。一个文件用于保存程序的声明（declaration），称为头文件。另一个文件用于保存程序的实现（implementation），称为定义（definition）文件。 C++/C 程序的头文件以“.h”为后缀，C 程序的定义文件以“.c”为后缀。

### 6.1 文件信息声明

文件信息声明位于头文件和定义文件的开头，主要内容有：

（1） 版权信息；

（2） 文件名称，项目代码，摘要，参考文献；

（3） 当前版本号，作者/修改者，完成日期；

（4） 版本历史信息；

（5） 主要函数描述。

文件信息声明模板如下：

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File Name :

\* Description :

\* Version :

\* Author :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

### 6.2 头文件

早期的编程语言如 Basic、Fortran 没有头文件的概念，C++/C 语言的初学者虽然会用使

用头文件，但常常不明其理。这里对头文件的作用略作解释：

（1） 通过头文件来调用库功能。在很多场合，源代码不便（或不准）向用户公布，

只要向用户提供头文件和二进制的库即可。用户只需要按照头文件中的接口声明来调用库功能，而不必关心接口怎么实现的。编译器会从库中提取相应的代码；

（2） 头文件能加强类型安全检查。如果某个接口被实现或被使用时，其方式与头文

件中的声明不一致，编译器就会指出错误，这一简单的规则能大大减轻程序员调试、改错的负担。

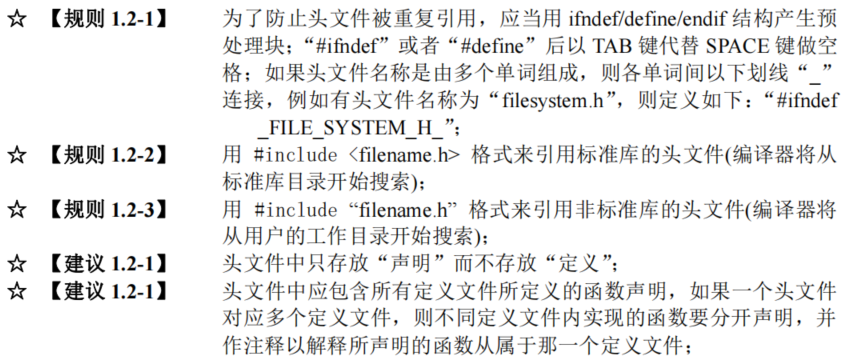
头文件由三部分内容组成：

(1) 头文件开头处的文件信息声明；

(2) 预处理块；

(3) 函数和类结构声明等。

假设头文件名称为 filesystem.h，头文件的结构参见如下示例



头文件模板如下：

// 文件信息声明

// 文件名字（大写）\_H

#ifndef \_File\_NAME\_H

#define \_File\_NAME\_H

//增加对C++项目引用的支持

//当在一个C++项目引用这个头文件时，编译器能知道要按照C语言对此文件进行编译

#ifdef \_\_cplusplus

Extern "C" {

#endif

// 引用的标准库头文件

/\* Includes ------------------------------------------------------------------\*/

#include <math.h>

…

// 引用的非标准库头文件

#include “myheader.h”

…

// 公共宏定义

/\* Common macro definitions---------------------------------------------------\*/

// 数据结构定义

/\* Data structure declaration-------------------------------------------------\*/

typedef struct Student

{

…

}Stu;

// 外部变量声明

/\* Extern Variable------------------------------------------------------------\*/

//函数声明

/\* Function declaration-------------------------------------------------------\*/

void Function1(…);

…

#ifdef \_\_cplusplus

}

#endif

#endif

**注意，以上文件//后内容不用写**

**//文件信息声明 按上文的文件信息声明模板编写**

### 6.3 源文件/定义文件

定义文件有三部分内容：

(1) 定义文件开头处的文件信息声明；

(2) 对一些头文件的引用；

(3) 程序的实现体（包括数据和代码）。

假设定义文件的名称为 filesystem.c，定义文件的结构参见示例

**注意，以下文件//后内容不用写**

**//文件信息声明 按上文的文件信息声明模板编写**

源文件模板如下：

// 文件信息声明

// 引用的头文件

/\* Includes ------------------------------------------------------------------\*/

#include “filesystem.h”

…

// 引用的外部函数声明

/\* External function declaration----------------------------------------------\*/

// 宏定义

/\* Private macro definitions--------------------------------------------------\*/

// 全局变量定义

/\* Global variable------------------------------------------------------------\*/

// 内部函数定义

/\* Static function definition-------------------------------------------------\*/

Static void static\_Function1(…)

{

…

}

// 函数定义

/\* Function definition--------------------------------------------------------\*/

void Function1(…)

{

…

}

### 3.4 目录结构

如果一个软件的头文件数目比较多（如超过十个），通常应将头文件和定义文件分别保

存于不同的目录，以便于维护。

例如可将头文件保存于 include 目录，将定义文件保存于 source 目录（可以是多级目录）。

如果某些头文件是私有的，它不会被用户的程序直接引用，则没有必要公开其“声明”。

为了加强信息隐藏，这些私有的头文件可以和定义文件存放于同一个目录。

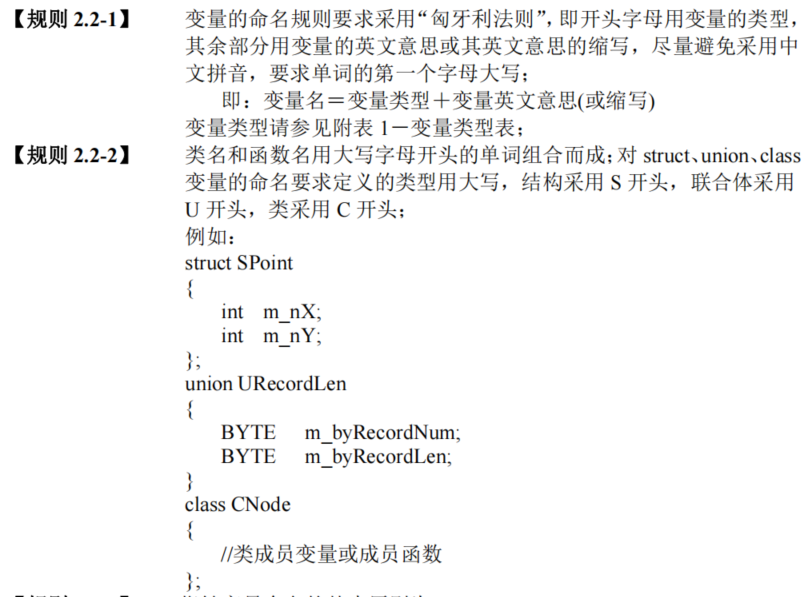
## 七、文件命名

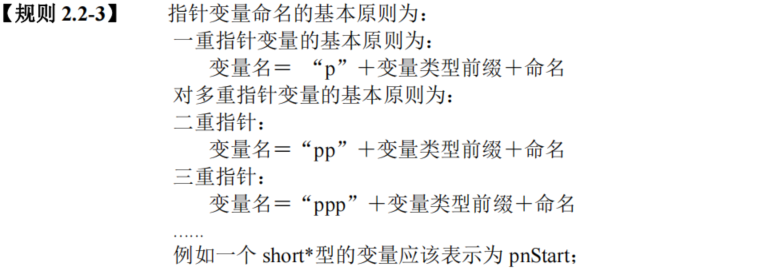
以“大文件夹\_子文件夹\_含义名”进行命名：

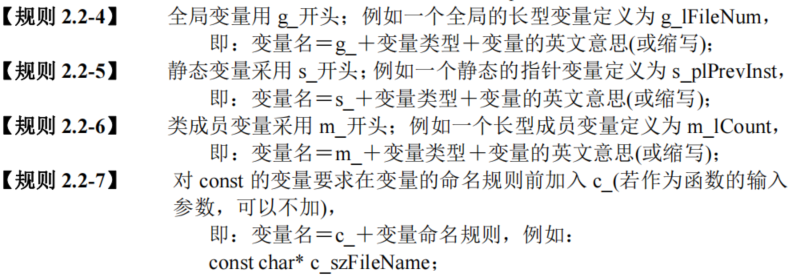
例如在Hardware文件夹下的ADC文件夹中ADC操作相关的源文件，命名为hardware\_adc\_operate.c

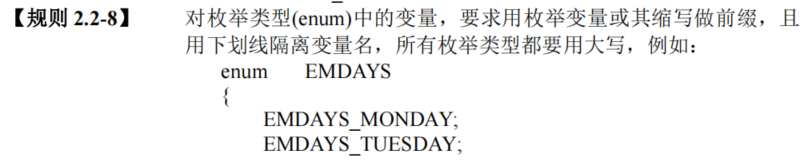
## 八、变量命名标准

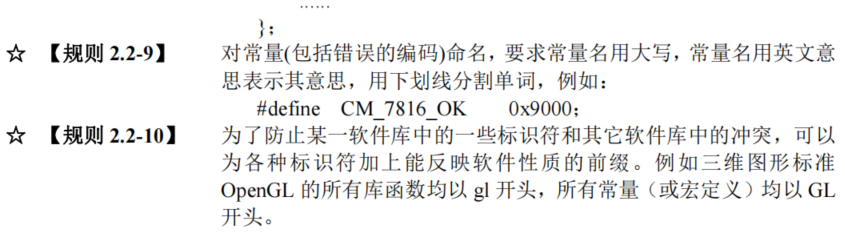
采用匈牙利命名法，具体规则如下：











## 九、类型前缀缩写

|  |
| --- |
| a 数组（Array） |
| b 布尔值（Boolean） |
| by 字节（Byte） |
| c 有符号字符（Char） |
| w 字（Word） |
| dw 双字（Double Word） |
| h Handle（句柄） |
| i 整形（Int） |
| f 浮点数（float） |
| p 指针（Pointer） |
| s 字符串（String） |

## 十、基本数据类型

为保证程序在不同系统间的可移植性，引入C99标准库中可移植类型stdint.h和inttypes.h。同时根据单片机特点（绝大部分单片机字长为32位）和实际需要，弃用字长为64位的数据类型。同时为了兼容ST单片机的旧类型，重新定义数据类型如下，并将重新定义的数据类型重新写入自定义的numtype.h文件夹：

/\* ========================= Basic data types==============================\*/

/\* Signed \*/

typedef signed char int8\_t;

typedef short int int16\_t;

typedef int int32\_t;

/\* Unsigned \*/

typedef unsigned char uint8\_t;

typedef unsigned short int uint16\_t;

typedef unsigned int uint32\_t;

/\* Float \*/

typedef float float32\_t;

typedef double float64\_t;

/\* Signed pointer \*/

typedef signed char \* p\_int8\_t;

typedef short int\* p\_int16\_t;

typedef int\* p\_int32\_t;

/\* Unsigned pointer \*/

typedef unsigned char\* p\_uint8\_t;

typedef unsigned short int\* p\_uint16\_t;

typedef unsigned int\* p\_uint32\_t;

/\* Void pointer \*/

typedef void\* p\_void;

/\* Float pointer \*/

typedef float\* p\_float32\_t;

typedef double\* p\_float64\_t;

/\* =========================== Compound data type ============================\*/

/\* Function return type \*/

typedef enum

{

Operation\_Success = 1,

Operation\_Fail = 0,

Operation\_Wait = 2

} t\_FuncRet;

/\* Boolean type definition \*/

typedef enum

{

FALSE = 0,

TRUE = 1

} bool;

以上数据变量命名方式遵循类型前缀缩写规定。

## 十一、宏定义

常量宏采用全部大写以及用 \_ 分隔符，尽量少用或者不用函数宏。

示例：

#define CM\_7816\_OK 0x9000；

## 十二、函数

函数接口的两个要素是参数和返回值。C 语言中，函数的参数和返回值的传递方式有两

种：值传递（pass by value）和指针传递（pass by pointer）。

### 12.1 函数注释

模板如下：

/\*\*

 \* @description:

 \* @param {数据类型}

 \* @return {数据类型}

 \* @author: leeqingshui

 \*/

### 12.2 参数规则

1. 参数的书写要完整，不要贪图省事只写参数的类型而省略参数名字，如果函数没有参数，则用 void 填充；例如：

void SetValue(int nWidth, int nHeight); // 良好的风格

void SetValue(int, int); // 不良的风格

float GetValue(void); // 良好的风格

float GetValue(); // 不良的风格

1. 参数命名要恰当，顺序要合理。

例如，编写字符串拷贝函数 StringCopy，它有两个参数，如果把参数名字起为 str1 和 str2，例如： void StringCopy(char \*str1, char \*str2); 那么我们很难搞清楚究竟是把 str1 拷贝到 str2 中，还是刚好倒过来， 可以把参数名字起得更有意义，如叫 strSource 和 strDestination。这 样从名字上就可以看出应该把 strSource 拷贝到 strDestination。还有一个问题，这两个参数那一个该在前那一个该在后？参数的顺序要遵循程序员的习惯。

一般地，应将目的参数放在前面，源参数放在后面。

如果将函数声明为：

void StringCopy(char \*strSource, char \*strDestination);

别人在使用时可能会不假思索地写成如下形式：

char str[20];

StringCopy(str, “Hello World”); // 参数顺序颠倒

1. 如果参数是指针，且仅作输入用，则应在类型前加 const，以防止该指针在函数体内被意外修改。例如：

void StringCopy(char \*strDestination，const char \*strSource);

1. 避免函数有太多的参数，参数个数尽量控制在 5 个以内。如果参数太多，在使用时容易将参数类型或顺序搞错；
2. 尽量不要使用类型和数目不确定的参数；C 标准库函数 printf 是采用不确定参数的典型代表，其原型为：int printf(const chat \*format[, argument]…); 这种风格的函数在编译时丧失了严格的类型安全检查。

### 12.3 返回值规则

1. 不要省略返回值的类型；
2. 不可返回局部指针；

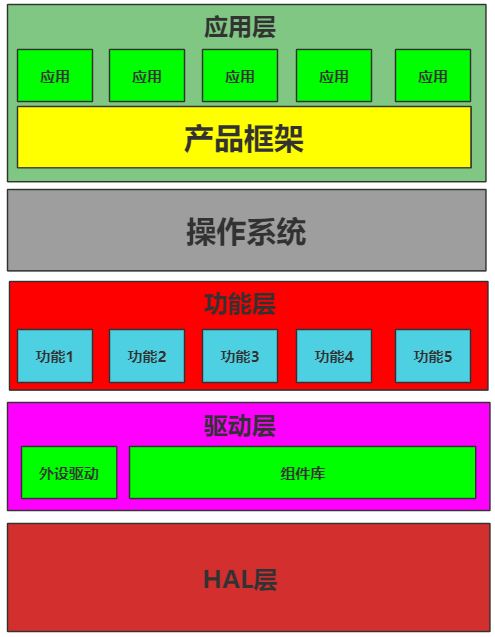
### 12.4 函数实现

1. 模块化设计：函数的功能要单一，不要设计多用途的函数；函数体的规模要小，尽量控制在 150 行代码之内。
2. 尽量避免函数带有“记忆”功能（全局变量、静态局部变量）。相同的输入应当产生相同的输出带有“记忆”功能的函数，其行为可能是不可预测的，因为它的行为可能取决于某种“记忆状态”。这样的函数既不易理解又不利于测试和维护。在 C语言中，函数的 static 局部变量是函数的“记忆”存储器。建议尽量少用 static 局部变量，除非必需。
3. 在函数体的“入口处”，对参数的有效性进行检查；很多程序错误是由非法参数引起的，我们应该充分理解并正确使用“断言”（assert）来防止此类错误。同时，不仅要检查输入参数的有效性，还要检查通过其它途径进入函数体内的变量的有效性，例如全局变量、文件句柄等。
4. 用于出错处理的返回值一定要清楚，让使用者不容易忽视或误解错误情况。
5. 避免函数嵌套过深，新增函数代码嵌套不超过4层。
6. 设计高扇入，合理扇出（小于7）的函数：

扇出是指一个函数直接调用（控制）其它函数的数目，而扇入是指有多少上级函数调用它。 扇出过大，表明函数过分复杂，控制和协调下级函数过多；扇出过小，表明函数的调用层次过多，不利于程序阅读和函数结构分析，并且运行时会对系统资源（如堆栈空间）造成压力。通常函数比较合理的扇出（调度函数除外）通常是3~5。

## 十三、分层设计

为了便于系统移植，整体程序应采用分层设计：



## STM32Readme文件模板

/\*============================Project description===============================\*/

Function brief :

/\*==============================code layers====================================\*/

1. Folder organization :

2. code layer:

/\*=====================Development Environment Description======================\*/

1. Development Board:

2. CPU internal resources :

(1) CPU model:

(2) Main frequency:

(3) Internal Flash capacity:

(4) Internal SRAM capacity:

3. Memory space allocation:

IROM1 : START - SIZE -

IRAM1 : START - SIZE -

STACK SIZE :

HEAP SIZE :

4. Software Development Related:

(1) IDE :

(2) Firmware Package Name and Version :

(3) Compiler :

(4) Common macro definition :

5. Debug Settings : ST-LINK/V2

(1) SWD Req: Selected:

(2) Connect & Reset Options :

(3) Cache Options :

(4) Trace Options :

(5) Flash Download :

/\*=========================GPIO pin use with peripherals==========================\*/

/\*======================= peripherals Parameter =================================\*/

/\*===================================Key points===============================\*/