ISIP嵌入式C语言编码规范

# 一、头文件

## 1.1 规则

* **【规则1-2-1】**为了防止头文件被重复引用，应当用ifndef/define/endif结构产生预处理块。
* **【规则1-2-2】**用 #include <filename.h> 格式来引用标准库的头文件（编译器将从标准库目录开始搜索）。
* **【规则1-2-3】**用 #include “filename.h” 格式来引用非标准库的头文件（编译器将从用户的工作目录开始搜索）。
* **【建议1-2-1】**头文件中只存放“声明”而不存放“定义”

**【建议1-2-2】**不提倡使用全局变量，尽量不要在头文件中出现象extern int value 这类声明，但有时为了提高代码运行效率，可以暴露部分全局变量。

## 1.2 头文件的作用

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

## 1.3 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。

在普通的小型嵌入式系统中，为了更快的找到声明和实现，我们一般将头和实现文件(.c,.cpp)放在同一目录下。对于大型的程序库，则需将头文件和实现文件分开存放。

# 二、程序的版式

## 2.1 空行

* **【规则2-1-1】**在每个类声明之后、每个函数定义结束之后都要加空行。参见示例2-1（a）
* **【规则2-1-2】**在一个函数体内逻辑上密切相关的语句之间不加空行，其它地方应加空行分隔。参见示例2-1（b）

|  |  |
| --- | --- |
| // 空行  void Function1(…)  {  …  }  // 空行  void Function2(…)  {  …  }  // 空行  void Function3(…)  {  …  } | // 空行  while (condition)  {  statement1;  // 空行  if (condition)  {  statement2;  }  else  {  statement3;  }  // 空行  statement4;  } |

示例2-1(a) 函数之间的空行 示例2-1(b) 函数内部的空行

## 2.2 代码行

* **【规则2-2-1】**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，并且方便于写注释。
* **【规则2-2-2】**if、for、while、do等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加{}。这样可以防止书写失误。

示例2-2（a）为风格良好的代码行，示例2-2（b）为风格不良的代码行。

|  |  |
| --- | --- |
| int width; // 宽度  int height; // 高度  int depth; // 深度 | int width, height, depth; // 宽度高度深度 |
| x = a + b;  y = c + d;  z = e + f; | X ＝ a + b; y = c + d; z = e + f; |
| if (width < height)  {  dosomething();  } | if (width < height) dosomething(); |
| for (initialization; condition; update)  {  dosomething();  }  // 空行  other(); | for (initialization; condition; update)  dosomething();  other(); |

示例2-2(a) **风格良好**的代码行 示例2-2(b) **风格不良**的代码行

* **【建议2-2-1】**尽可能在定义变量的同时初始化该变量（就近原则）

如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未被初始化的变量，可能会导致程序错误。本建议可以减少隐患。例如

int width = 10; // 定义并初绐化width

int height = 10;// 定义并初绐化height

int depth = 10; // 定义并初绐化depth

## 2.3 代码行内的空格

* **【规则2-3-1】**关键字之后要留空格。象const、virtual、inline、case 等关键字之后至少要留一个空格，否则无法辨析关键字。象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。
* **【规则2-3-2】**函数名之后不要留空格，紧跟左括号‘（’，以与关键字区别。
* **【规则2-3-3】**‘（’向后紧跟，‘）’、‘，’、‘;’向前紧跟，紧跟处不留空格。
* **【规则2-3-4】**‘，’之后要留空格，如Function(x, y, z)。如果‘;’不是一行的结束符号，其后要留空格，如for (initialization; condition; update)。
* **【规则2-3-5】**赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“\*”、“%”、“&&”、“||”、“<<”,“^”等二元操作符的前后应当加空格。
* **【规则2-3-6】**一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。
* **【规则2-3-7】**象“［］”、“.”、“->”这类操作符前后不加空格。
* **【建议2-3-1】**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，如for (i=0; i<10; i++)和if ((a<=b) && (c<=d))

|  |
| --- |
| void Func1(int x, int y, int z); // 良好的风格  void Func1 (int x,int y,int z); // 不良的风格 |
| if (year >= 2000) // 良好的风格  if(year>=2000) // 不良的风格  if ((a>=b) && (c<=d)) // 良好的风格  if(a>=b&&c<=d) // 不良的风格 |
| for (i=0; i<10; i++) // 良好的风格  for(i=0;i<10;i++) // 不良的风格  for (i = 0; I < 10; i ++) // 过多的空格 |
| x = a < b ? a : b; // 良好的风格  x=a<b?a:b; // 不好的风格 |
| int \*x = &y; // 良好的风格  int \* x = & y; // 不良的风格 |
| array[5] = 0; // 不要写成 array [ 5 ] = 0;  a.Function(); // 不要写成 a . Function();  b->Function(); // 不要写成 b -> Function(); |

示例2-3 代码行内的空格

## 2.4 对齐

对齐采用stroustrup格式，即大括号后半部分和语句块对齐，示例2-4。

|  |
| --- |
| void Function(int x) {  … // program code  } |
| if (condition) {  … // program code  }  else {  … // program code  } |
| for (initialization; condition; update) {  … // program code  } |
| while (condition) {  … // program code  } |
|  |

示例2-4 对齐

## 2.5 长行拆分

* **【规则2-5-1】**代码行最大长度宜控制在70至80个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。
* **【规则2-5-2】**长表达式要在低优先级操作符处拆分成新行，**操作符放在新行之首**（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

|  |
| --- |
| if ((very\_longer\_variable1 >= very\_longer\_variable12)  && (very\_longer\_variable3 <= very\_longer\_variable14)  && (very\_longer\_variable5 <= very\_longer\_variable16)) {  dosomething();  } |
| virtual CMatrix CMultiplyMatrix (CMatrix leftMatrix,  CMatrix rightMatrix); |
| for (very\_longer\_initialization;  very\_longer\_condition;  very\_longer\_update) {  dosomething();  } |

示例2-5 长行的拆分

## 2.6 Tab

Tab设置为缩进4格，不替换成空格。

# 三、命名

我们采用Linux下的命名规则，建议的命名规则如下。

(1)变量名必须有一定的意义，并且意义准确。例如 有一个变量用于保存图书的数目，可以命名为number\_of\_book或者num\_of\_book。不建议使用i，因为它没有意义。也不建议使用 number或book，因为意义不准确。

(2)不建议大小写混用。如定义一个计数变 量，int nCount；这在Windows中是一个很好的变量名，其中nCount的首字母n用来说明这个变量的类型是int。但在Linux下不建议大小写混合使用，一般标识符只由小写字母，数字和下划线构成。

(3)在失去意义的情况下，尽量使用较短的变量名。 例如有一个变量，用于暂时存储一个计数值，把变量命名为tmp\_count显然要比this\_is\_a\_temperary\_counter好。

(4)不采用匈牙利命名法表示变量的类型。如int nCount；n用于说明变量的类型，在Linux中不建议这样命名变量。

(5)函数名应该以动词开头，因为函数是一组具有特定功能的语句块。比如一个函数，它用于取得外部输入的数值，则可以命名为get\_input\_number。

(6)尽量避免使用全局变量。

# 四、程序的结构

## 4.1 模块化、面向对象

程序设计采用模块化设计，每个模块完成单独的一个功能模块，每个模块至少有一个.c文件和.h文件构成，.h文件是此模块的接口文件，声明别的模块能够使用的函数、数据类型、全局变量，由于.h文件起到接口的作用，其中的内容应该尽量的少，不被外部使用的函数、变量、类型都在.c文件中定义、使用。

划分模块功能时，采用面向对象的思想，先设计后实现，一个模块可以抽象为一个**类**，通过.h文件实现封装性，通过结构体的嵌入实现继承。

每个模块实现中，声明此模块的主要结构体类型，如：

typedef struct tcp\_comm\_t {

int connected\_socket\_id;

int connected\_count;

} tcp\_comm\_t;

一个类型有三类成员，1. 数据成员，以结构体的形式表达此类对象可以使用的数据信息；2. 类的成员函数，或者叫成员方法，是表达类对象可以执行的动作，完成对象的具体功能。 3. 类的事件。类的事件是类对象可以发出的事件，用函数指针的形式存在，使用类对象的其他类中可以定义类对象的事件处理函数，并挂载到类对象的事件上；当类对象发生此事件时，将回调此事件对应的事件处理函数。比如按键对象有按键按下on\_key\_press、按键松开on\_key\_release事件，如果一个App模块使用了按键模块，就可以自己定义按键按下处理函数app\_\_key\_press\_callback并挂载到按键对象的on\_key\_press事件，这样当按键模块扫描到按键被按下时就会回调App对象的app\_\_key\_press\_callback函数。类的事件可以由类对象在运行时根据条件产生和处理，比如使用类对象内部的定时扫描、硬件中断等，也可以由类对象的成员函数被调用时达到了事件触发条件而产生。

## 4.2 访问权限限定规则

由于C语言结构体没有对成员访问做限定，所以在编写代码时程序员应该主动确定访问限制的约定，我们规定如下：

私有成员以下划线开头。

公开成员不以下划线开头。

并且公开成员和私有成员定义尽量分开，推荐先写公开成员，后写私有成员。

## 4.3 唯一变量模式

如果系统中只需要存在一个此类的实例对象，则可以在.c中定义此类型的全局变量，如 tcp\_comm\_t g\_tcp\_comm;如果别的模块需要此对象的信息，可以将此对象通过extern声明到对应的.h文件中。

## 4.4 成员函数命名规则

类的函数（方法）命名以类名起始，以**两个**下划线作为类名和成员的分割，如：

int tcp\_comm\_\_init(int socket);

int tcp\_comm\_\_rev\_msg(uint8\_t \*buffer, int length);

int tcp\_comm\_\_send\_msg(uint8\_t \*msg, int msglength);

如果此类可以实例化多个对象，则在定义其成员函数时，第一个参数必须为要操作的类的对象指针，如：

int tcp\_comm\_\_dispatch\_msg(tcp\_comm\_t \*tcpcomm, uint8\_t \*msg, int msglength);

## 4.5 私有成员函数

在.c文件中只被此模块所使用的函数，声明时需声明为static，即本地化，防止污染全局命名空间。

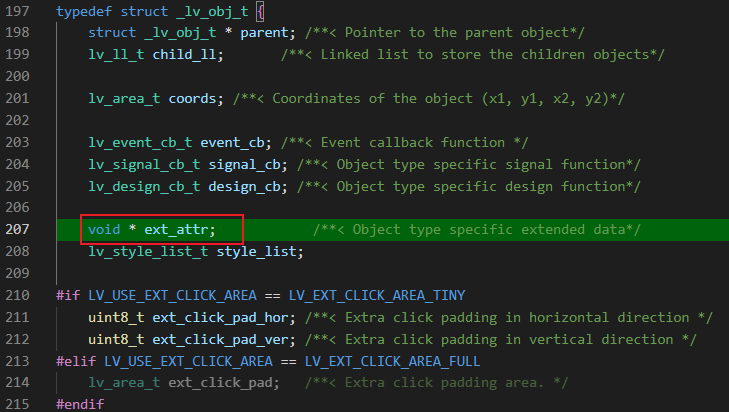
## 4.6 继承的实现

用C语言描述继承比较困难，如果确实需要继承，则推荐用单一类作为根类，如所有类都从object\_t开始继承，在父类中保留一个指向子类具体属性的一个属性。以LVGL库为例，在LVGL中所有的对象都是lv\_obj\_t，lv\_obj\_t有比较多的共用的方法，如下：

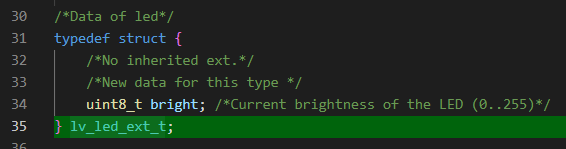
（https://docs.lvgl.io/master/overview/object.html）



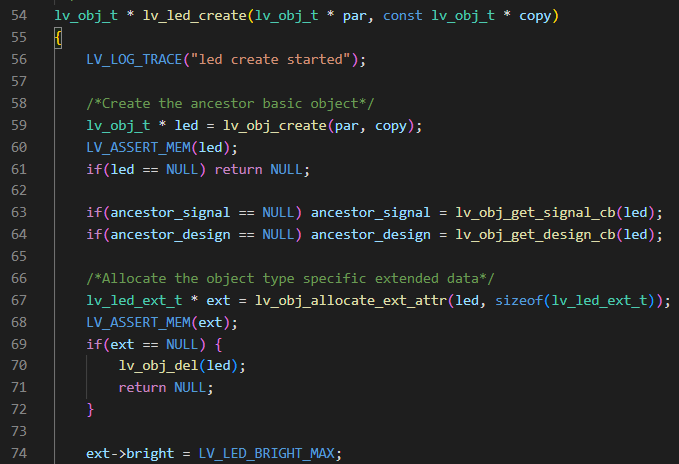
共用的设置大小、尺寸等操作都封装在object类中，而继承衍生的子组件可以有自己的属性和方法，子组件额外的属性都可以存储在object的ext\_attr属性，ext\_attr只是一个指针，可以指向任何数据结构，因此每种子组件都可以拥有自己特殊的属性。如



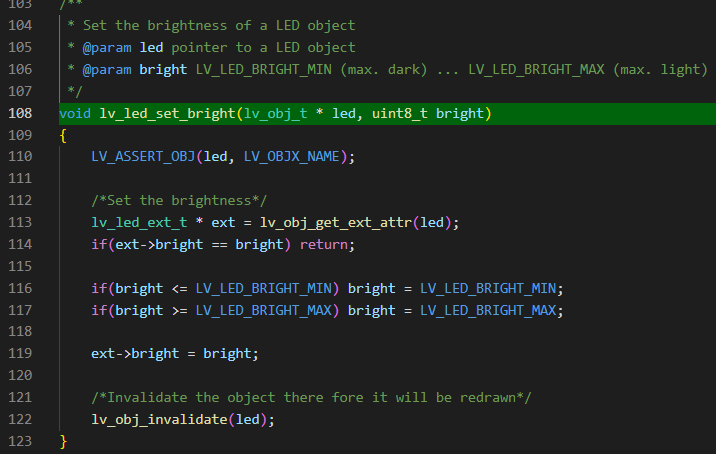
以LED控件为例，LED控件额外的属性如下：



在LED控件构造函数中，会为ext\_attr分配内存，并设置对应的额外属性，如下图所示：



LED组件也增加了特有是方法，设置LED的亮度，其实就是操作额外属性bright，如下图所示：



可以看到，每个子类都可以增加自己所需要的属性和方法。

在我们具体的代码设计时可以参考类似的设计方案。