规范制定原则

1 方便代码的交流和维护。

2 不影响编码的效率，不与大众习惯冲突。

3 使代码更美观、阅读更方便。

4 使代码的逻辑更清晰、更易于理解。

5 本规范未说明的规则请遵循基础标准规范。

术语定义

Pascal 大小写

将标识符的首字母和后面连接的每个单词的首字母都大写。可以对三字符或更多字符的标识符使用Pascal 大小写。例如：CamelPascal

文件命名组织

1. 文件命名

1 文件名遵从Pascal命名法，无特殊情况，扩展名小写。

2 使用统一而又通用的文件扩展名： C# 类.cs

1. 命名空间

把所有系统框架提供的命名空间组织到一起，把第三方提供的命名空间放到系统命名空间的下面

3. 注意事项

1、一个文件应该只向在一个名称空间内定义类型。避免在一个文件中使用多个名称空间

2、避免在同一个文件中放置多个类

3、避免在一个文件内写多于500行的代码（机器自动生成的代码除外

4、一行不要超过80个字符

5、不要手动去修改任何机器生成的代码

代码规范化基本上有七大原则，体现在空行、空格、成对书写、缩进、对齐、代码行、注释七方面的书写规范上。

**1) 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，虽然打印含有空行的程序会多消耗一些纸张，但是值得。  
**规则一**：

定义变量后要空行。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。  
**规则二**：

每个函数定义结束之后都要加空行。  
 总规则：两个相对独立的程序块、变量说明之后必须要加空行。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) 空格**

**规则一**：

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。  
**规则二**：

函数名之后不要留空格，应紧跟左括号(，以与关键字区别。  
**规则三**：

(向后紧跟；)、,、;这三个向前紧跟；紧跟处不留空格。  
**规则四**：

”,”之后要留空格。如果;不是一行的结束符号，其后要留空格。  
**规则五**：

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。  
注意，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。  
**规则六**：

单目运算符 !、~、++、－－、－、\*、& 等前后不加空格。  
注意：

这里的“－”和规则五里面的“－”不同。这里的“－”是负号运算符，规则五里面的“－”是减法运算符。

这里的“\*”和规则五里面的“\*”也不同。

这里的“\*”是指针运算符，规则五里面的“\*”是乘法运算符。

这里的“&”和规则五里面的“&”也不同。这里的“&”是取地址运算符，规则五里面的“&”是按位与运算符。  
 总之，规则六中的是单目运算符，而规则五中的是双目运算符，它们是不一样的。  
**规则七**：

像数组符号[]、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。  
**规则八**：

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++)

for 和分号后面保留空格就可以了，=和<前后的空格可去掉。

**3) 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。

**4) 缩进**

缩进是通过键盘上的 Tab 键实现的，缩进可以使程序更有层次感。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。

**5) 对齐**

对齐主要是针对大括号{}说的：  
 **规则一**：{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。  
 **规则二**：{}之内的代码要向内缩进一个 Tab，且同一地位的要左对齐，地位不同的继续缩进。  
还有需要注意的是，很多编程软件是会“自动对齐”的，比如：

#include <stdio.h>  
int main(void)  
{  
    if (…)  
    return 0;  
}

写完 if 那一行后，按回车，此时光标在括号的右边，而此 if 下的大括号要写在与 if 左对齐的正下方，通常我们是按一下 Backspace 键使光标停在与 if 左对齐的正下方。但事实上我们不需要这样做，我们直接输入大括号即可，系统会自动对齐到与 if 左对齐的正下方。再接着看：

#include <stdio.h>  
int main(void)  
{  
    if (…)  
    {  
        while (…)  
    }  
    return 0;  
}

写完 while 那一行后，按回车，此时光标不是停在与 while 左对齐的正下方，同样，我们不需要按 Backspace，直接输入大括号即可，系统会自动对齐到与 while 左对齐的正下方的。  
 此外编程软件还有“对齐、缩进修正”功能。就是按 Ctrl+A 全选，然后按 Alt+F8，这时程序中所有成对的大括号都会自动对齐，未缩进的也会自动缩进。不管是在编程过程中，还是在编写结束完之后，都可以使用这个技巧。但如果完全按照规范写，那根本就不需要这个技巧，所以，这只是一个辅助功能。

**6) 代码行**

**规则一**：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。  
  
 **规则二**：if、else、for、while、do 等语句自占一行，执行语句不得紧跟其后。此外，非常重要的一点是，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。

**7) 注释**

C语言中一行注释一般采用//…，多行注释必须采用/\*…\*/。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。  
 **规则一**：注释是对代码的“提示”，而不是文档。程序中的注释不可喧宾夺主，注释太多会让人眼花缭乱。  
 **规则二**：如果代码本来就是清楚的，则不必加注释。

例如：

i++;  //i加1

这个就是多余的注释。  
 **规则三**：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。  
 **规则四**：当代码比较长，特别是有多重嵌套的时候，应当在段落的结束处加注释，这样便于阅读。  
 **规则五**：每一条宏定义的右边必须要有注释，说明其作用。