软件编码规范说明书

一、排版

1.程序块要采用缩进风格编写，缩进的空格数为 4 个。

说明：对于由开发工具自动生成的代码可以有不一致。

2.较长的语句（>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。

3.循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分，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。

4.若函数或过程中的参数较长，则要进行适当的划分。

5.不允许把多个短语句写在一行中，即一行只写一条语句。

6.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 {}。

7.对齐只使用空格键，不使用 TAB键

说明： 以免用不同的编辑器阅读程序时， 因TAB键所设置的空格数目不同而造成程序布局不整齐，不要使用BC作为编辑器合版本，因为BC会自动将8个空格变为一个 TAB 键，因此使用 BC 合入的版本大多会将缩进变乱。

8.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。

9.程序块的分界符应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及 if、for、do、while、switch、case语句中的程序都要采用如上的缩进方式。

二、注释

1.一般情况下，源程序有效注释量必须在 20 ％以上

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。

2.说明性文件（如头文件.h文件、.inc文件、.def文件、编译说明文件.cfg等）头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明。

3.源文件头部应进行注释，列出：版权说明、版本号、生成日期、作者、模块目的 / 功能、主要函数及其功能、修改日志等。

4.函数头部应进行注释，列出：函数的目的 / 功能、输入参数、输出参数、返回值、调用关系（函数、表）等。

5.注释的内容要清楚、明了，含义准确，防止注释二义性。

说明：错误的注释不但无益反而有害。

6.注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

7.数据结构声明 ( 包括数组、结构、类、枚举等 ) ，如果其命名不是充分自注释的，必须加以注释。对数据结构的注释应放在其上方相邻位置，不可放在下面；对结构中的每个域的注释放在此域的右方

8.全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明

9.注释与所描述内容进行同样的缩排

说明：可使程序排版整齐，并方便注释的阅读与理解。

10.将注释与其上面的代码用空行隔开

11.注释格式尽量统一，建议使用 “/\* …… \*/ ”

三、标识符命名

1.标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

说明：较短的单词可通过去掉 “元音”形成缩写；较长的单词可取单词的头几个字母形成缩写；一些单词有大家公认的缩写。

2.命名中若使用特殊约定或缩写，则要有注释说明

说明：应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写，进行必要的注释说明。

3.自己特有的命名风格，要自始至终保持一致，不可来回变化

说明：个人的命名风格，在符合所在项目组或产品组的命名规则的前提下，才可使用。（即命名规则中没有规定到的地方才可有个人命名风格）。

4.在同一软件产品内，应规划好接口部分标识符（变量、结构、函数及常量）的命名，防止编译、链接时产生冲突。

说明： 对接口部分的标识符应该有更严格限制，防止冲突。 如可规定接口部分的变量与常量之前加上 “模块 ”标识等。

5.除了编译开关 / 头文件等特殊应用，应避免使用 \_EXAMPLE\_TEST\_ 之类以下划线开始和结尾的定义。

四、可读性

1.注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级

说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。

2.避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替。

3.源程序中关系较为紧密的代码应尽可能相邻

说明：便于程序阅读和查找。

4.不要使用难懂的技巧性很高的语句，除非很有必要时

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。

五、函数、过程

1.对所调用函数的错误返回码要仔细、全面地处理

2.明确函数功能，精确（而不是近似）地实现函数设计

3.防止将函数的参数作为工作变量

说明：将函数的参数作为工作变量，有可能错误地改变参数内容，所以很危险。对必须改变的参数，最好先用局部变量代之，最后再将该局部变量的内容赋给该参数。

4.函数的规模尽量限制在 200 行以内

说明：不包括注释和空格行。

5.一个函数仅完成一件功能

6.为简单功能编写函数

说明：虽然为仅用一两行就可完成的功能去编函数好象没有必要，但用函数可使功能明确化，增加程序可读性，亦可方便维护、测试。

7.检查函数所有参数输入的有效性

8.函数名应准确描述函数的功能

六、可测性

1.在进行集成测试 / 系统联调之前，要构造好测试环境、测试项目及测试用例，同时仔细分析并优化测试用例，以提高测试效率

说明：好的测试用例应尽可能模拟出程序所遇到的边界值、各种复杂环境及一些极端情况等。

2.在编写代码之前，应预先设计好程序调试与测试的方法和手段，并设计好各种调测开关及相应测试代码如打印函数等

说明：程序的调试与测试是软件生存周期中很重要的一个阶段，如何对软件进行较全面、高率的测试并尽可能地找出软件中的错误就成为很关键的问题。因此在编写源代码之前，除了要有一套比较完善的测试计划外，还应设计出一系列代码测试手段，为单元测试、集成测试及系统联调提供方便。

七、程序效率

1.编程时要经常注意代码的效率

说明：代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。

2.在保证软件系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率

说明：不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。

3.通过对系统数据结构的划分与组织的改进，以及对程序算法的优化来提高空间效率

说明：这种方式是解决软件空间效率的根本办法。

4.循环体内工作量最小化

说明：应仔细考虑循环体内的语句是否可以放在循环体之外，使循环体内工作量最小，从而提高程序的时间效率。

5.仔细分析有关算法，并进行优化

6.编程时，要随时留心代码效率；优化代码时，要考虑周全

7.在保证程序质量的前提下， 通过压缩代码量、 去掉不必要代码以及减少不必要的局部和全局变量，

说明：这种方式对提高空间效率可起到一定作用，但往往不能解决根本问题。

八、质量保证

1.在软件设计过程中构筑软件质量

2.代码质量保证优先原则

（1）正确性，指程序要实现设计要求的功能。

（2）稳定性、安全性，指程序稳定、可靠、安全。

（3）可测试性，指程序要具有良好的可测试性。

（4）规范 / 可读性，指程序书写风格、命名规则等要符合规范。

（5）全局效率，指软件系统的整体效率。

（6）局部效率，指某个模块 / 子模块 / 函数的本身效率。

（7）个人表达方式 / 个人方便性，指个人编程习惯。

3.防止内存操作越界

说明：内存操作主要是指对数组、指针、内存地址等的操作。内存操作越界是软件系统主要错误之一，后果往往非常严重，所以当我们进行这些操作时一定要仔细小心。

4.认真处理程序所能遇到的各种出错情况

5.精心构造算法，并对其性能、效率进行测试

九、代码编辑、编译、审查

1.打开编译器的所有告警开关对程序进行编译

2.通过代码走读及审查方式对代码进行检查

说明：代码走读主要是对程序的编程风格如注释、命名等以及编程时易出错的内容进行检查，可由开发人员自己或开发人员交叉的方式进行；代码审查主要是对程序实现的功能及程序的稳定性、安全性、可靠性等进行检查及评审，可通过自审、交叉审核或指定部门抽查等方式进行。

3.测试部测试产品之前，应对代码进行抽查及评审

4.编写代码时要注意随时保存，并定期备份，防止由于断电、硬盘损坏等原因造成代码丢失

5.使用软件工具进行代码审查

十、代码测试、维护

1.单元测试要求至少达到语句覆盖

2.清理、整理或优化后的代码要经过审查及测试

3.代码版本升级要经过严格测试

4.使用工具软件对代码版本进行维护