|  |  |  |  |
| --- | --- | --- | --- |
| **深圳国泰安教育技术股份有限公司** | **版本** | **密级** | **页数** |
| **V1.0** | **机密** | **共19页** |
| **文档编号:** **GTA\_CMMI\_CodeStandard\_C&C++** | | |

国泰安软件编码规范-C&C++语言

|  |  |  |  |
| --- | --- | --- | --- |
| **作者** | 柳晓煜 | **编制日期** | 2014-03-13 |
| **审核** |  | **审核日期** |  |
| **批准** |  | **批准日期** |  |

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| V1.0 | 柳晓煜 |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目 录

[1. 文档介绍 5](#_Toc385511898)

[1.1 文档目的 5](#_Toc385511899)

[1.2 读者对象 6](#_Toc385511900)

[1.3 参考文档 6](#_Toc385511901)

[1.4 术语与缩写解释 6](#_Toc385511902)

[2. 编程版式规范 7](#_Toc385511903)

[2.1 总体原则 7](#_Toc385511904)

[2.1 文件结构 8](#_Toc385511905)

[2.1.1 版权和版本声明 8](#_Toc385511906)

[2.1.2 函数头注释 8](#_Toc385511907)

[2.2 程序版式 9](#_Toc385511908)

[2.2.1 空行 9](#_Toc385511909)

[2.2.2 代码行 9](#_Toc385511910)

[2.2.3 代码行内空格 10](#_Toc385511911)

[2.2.4 对齐 11](#_Toc385511912)

[2.2.5 长行拆分 12](#_Toc385511913)

[2.2.6 修饰符的位置 12](#_Toc385511914)

[2.2.7 注释 13](#_Toc385511915)

[2.2.8 制表符与空格 13](#_Toc385511916)

[2.2.9 质量保证 13](#_Toc385511917)

[2.2.10 程序版式参照样例 15](#_Toc385511918)

[3. C/C++语言编程规范 16](#_Toc385511919)

[3.1 头文件 16](#_Toc385511920)

[3.2 命名规则 16](#_Toc385511921)

[3.2.1 通用命名规则 16](#_Toc385511922)

[3.2.2 变量 17](#_Toc385511923)

[3.3 if语句 17](#_Toc385511924)

[3.3.1 布尔变量与零值比较 17](#_Toc385511925)

[3.3.2 整形变量与零值比较 17](#_Toc385511926)

[3.3.3 浮点变量与零值比较 17](#_Toc385511927)

[3.2.2.4 指针变量与零值比较 18](#_Toc385511928)

[附录一《Google的C++编码规范》附件 19](#_Toc385511929)

# 文档介绍

为提高产品代码质量，指导广大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，重新对编程规范进行了梳理、优化、刷新，编写了本规范。

## 1.1 文档目的

* **提高可读性** “任何一个傻瓜都能写出计算机可以理解的代码，唯有写出人类容易理解的代码，才是优秀的程序员。”编码规范，帮助我们写出人类容易理解的代码，它为我们提供了最基本的模板，良好的编码风格，使代码具有一定的描述性，可以通过名字来获取一些需要IDE才能得到的提示，如可访问性、继承基类等。
* **统一全局，促进团队协作** 开发软件是一个团队活动，而不是个人的英雄主义。编码规范，要求团队成员遵守这一统一的全局决策，这样成员之间可以轻松地阅读对方的代码，所有成员正以一种清晰而一致的风格进行编码。而且，开发人员也可以集中精力关注他们真正应该关注的问题——自身代码的业务逻辑，与需求的契合度等局部问题**。**
* **有助于知识传递，加快工作交接**风格的相似性，能让开发人员更迅速，更容易理解一些陌生的代码，更快速地理解别人的代码。因为，他和你的代码风格是一样的，你没有必要对他的一些个性化风格进行揣测。这样的好处是开发人员可以很快的接手项目组其他成员的工作，快速完成工作交接。
* **减少名字增生，降低维护成本** 在没有规范的情况下，和容易为同一类型的实例起不同的名字。对于以后维护这些代码程序员来说会产生疑惑。
* 强调变量之间的关系，降低缺陷引入的机会 命名可以表示一定的逻辑关系，是开发人员在使用时保持警惕，从而一定程度上减少缺陷被引人的机会。
* **提高程序员的个人能力** 不可否认，每个程序员都应该养成良好的编码习惯，而编码规范无疑是教材之一。从一个程序员的代码本身能看出很多东西。所以，即便是为了自身发展，作为程序员也没有理由抵制这种规则的存在。你可能没有认识到，我们正默默地得益于编码规范。

## 1.2 读者对象

设计人员、开发人员。

## 1.3 参考文档

【1】Google的C++编码规范 作者：edisonpeng 版本： 发布日期：2009/3/25

【2】高质量C++C编程指南 作者：林锐 版本：V1.0 发布日期：2001/7/24

【3】华为c语言编程规范 作者： 版本： 发布日期：2011/5/9

## 1.4 术语与缩写解释

|  |  |
| --- | --- |
| **缩写、术语** | **解 释** |
|  |  |
|  |  |
|  |  |

# 2. 编程版式规范

## 2.1 总体原则

**1、清晰第一**

**清晰性是易于维护、易于重构的程序必需具备的特征**。

“程序必须为阅读它的人而编写，只是顺便用于机器执行。”——Harold Abelson 和 Gerald Jay Sussman “编写程序应该以人为本，计算机第二。”——Steve McConnell

本规范通过后文中的原则（如优秀的代码可以自我解释，不通过注释即可轻易读懂/头文件中适合放置接口的声明，不适合放置实现/除了常见的通用缩写以外，不使用单词缩写，不得使用汉语拼音）、规则（如防止局部变量与全局变量同名）等说明清晰的重要性。

一般情况下，代码的可阅读性高于性能，只有确定性能是瓶颈时，才应该主动优化。

**2、简洁为美**

**简洁就是易于理解并且易于实现**。代码越长越难以看懂，也就越容易在修改时引入错误。写的代码越多，意味着出错的地方越多，也就意味着代码的可靠性越低。因此，我们提倡大家通过编写简洁明了的代码来提升代码可靠性。

废弃的代码(没有被调用的函数和全局变量)要及时清除，重复代码应该尽可能提炼成函数。

本规范通过后文中的原则（如文件应当职责单一/一个函数仅完成一件功能）、规则（重复代码应该尽可能提炼成函数/避免函数过长，新增函数不超过50行）等说明简洁的重要性。

**3、选择合适的风格，与代码原有风格保持一致**

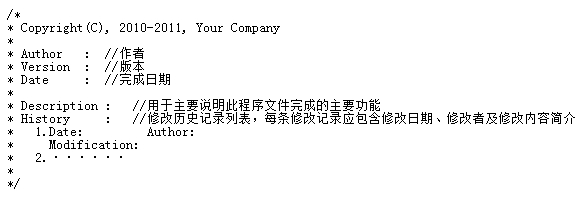
产品所有人共同分享同一种风格所带来的好处，远远超出为了统一而付出的代价。在公司已有编码规范的指导下，审慎地编排代码以使代码尽可能清晰，是一项非常重要的技能。如果重构/修改其他风格的代码时，比较明智的做法是根据现有代码的现有风格继续编写代码，或者使用格式转换工具进行转换成公司内部风格。

## 2.1 文件结构

2.1.1 版权和版本声明

版权和版本的声明位于文件的开头（参见示例1-1），主要内容有：

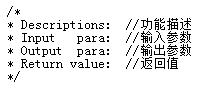
1. 版权信息。
2. 文件摘要。
3. 当前版本号，作者/修改者，完成日期。
4. 版本历史信息。



2.1.2 函数头注释

1. 函数功能描述，若分配空间，需要由调用者释放。
2. 输入输出参数，是否可以为NULL。
3. 返回值。

注：在类中除public接口外，其他函数可相应更换为简单注释：保留函数功能即可。



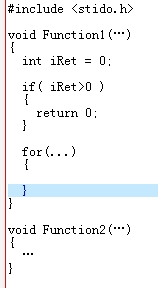
## 2.2 程序版式

版式虽然不会影响程序的功能，但会影响可读性。程序的版式追求清晰、美观，是程序风格的重要构成因素。

2.2.1 空行

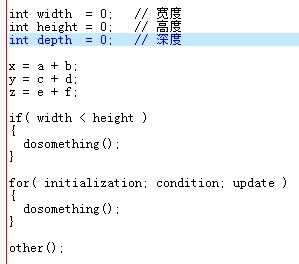
空行起着分隔程序段落的作用。空行得体将使程序的布局更加清晰。空行不会浪费内存。

* 【**规则2-1**】 在每个类声明之后、每个函数定义结束之后都要加空行。
* 【**规则2-2**】 在一个函数体内，逻揖上密切相关的语句之间不加空行，其它地方应加空行分隔。如：if()else()、while()、switch()、for()等前后加空行。



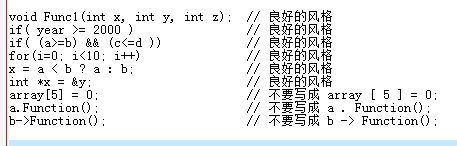
2.2.2 代码行

* 【**规则2-3**】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2-4**】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。
* 【**建议2-1**】尽可能在定义变量的同时初始化该变量（就近原则）



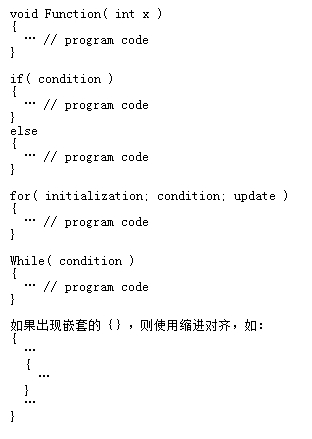
2.2.3 代码行内空格

* 【**规则2-5**】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。
* 【**规则2-6**】函数名之后不要留空格，紧跟左括号‘（’，以与关键字区别。
* 【**规则2-7**】‘**（**’向前紧跟，后留空格，‘**）**’向后紧跟，前留空格，‘**;**’，向前紧跟，紧跟处不留空格。
* 【**规则2-8**】‘，’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2-9**】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2-10**】一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。
* 【**规则2-11**】象“［］”、“.”、“->”这类操作符前后不加空格。
* 【**建议2-2**】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) )



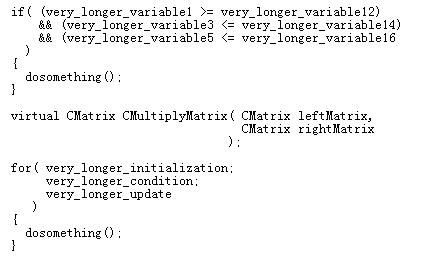
2.2.4 对齐

* 【**规则2-12**】程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐。
* 【**规则2-13**】{ }之内的代码块在‘{’右边数格处左对齐。



2.2.5 长行拆分

* 【建议**2-3**】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**2-14**】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。



2.2.6 修饰符的位置

* 【**规则2-15**】应当将修饰符 \* 和 ＆ 紧靠变量名

C:\Users\xiaoyu.liu\AppData\Roaming\Tencent\Users\17611253\QQ\WinTemp\RichOle\LNSMP3TPT[YT14[GGM$`6~K.jpg

2.2.7 注释

C 语言的注释符为“/\*…\*/”。C++语言中，程序块的注释常采用“/\*…\*/”，行注释一般采用“//…”。

注释通常用于：

（1）版本、版权声明；

（2）函数接口说明；

（3）重要的代码行或段落提示。

2.2.8 制表符与空格

* 【**规则2-16**】在一个函数内，行内缩进必须采用统一的格式：制表符或者空格。制表符占值大于等于2，小于等于4。**建议**：只使用空格，每次缩进2个空格。 使用空格进行缩进，不要在代码中使用tabs，设定编辑器将tab转为空格。

2.2.9 质量保证

* 【**规则2-17**】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. 简洁性，指程序易于理解并且易于实现。
3. 可维护性，指程序被修改的能力，包括纠错、改进、新需求或功能规格变化的适应能力。
4. 可靠性，指程序在给定时间间隔和环境条件下，按设计要求成功运行程序的概率。
5. 代码可测试性，指软件发现故障并隔离、定位故障的能力，以及在一定的时间和成本前提下，进行测试设计、测试执行的能力。
6. 代码性能高效，指是尽可能少地占用系统资源，包括内存和执行时间。

* 【**规则2-18**】禁止内存操作越界。

坚持下列措施可以避免内存越界：

1. 数组的大小要考虑最大情况，避免数组分配空间不够。
2. 避免使用危险函数sprintf /vsprintf/strcpy/strcat/gets操作字符串，使用相对安全的函数snprintf/strncpy/strncat/fgets代替。
3. 使用memcpy/memset时一定要确保长度不要越界
4. 字符串考虑最后的’\0’， 确保所有字符串是以’\0’结束
5. 指针加减操作时，考虑指针类型长度
6. 数组下标进行检查
7. 使用时sizeof或者strlen计算结构/字符串长度，避免手工计算。对于二进制流，禁止使用strlen计算长度，可用sizeof。

* 【**规则2-19**】禁止内存泄漏。

坚持下列措施可以避免内存泄漏：

1. 异常出口处检查内存、定时器/文件句柄/Socket/队列/信号量/GUI等资源是否全部释放
2. 删除结构指针时，必须从底层向上层顺序删除
3. 使用指针数组时，确保在释放数组时，数组中的每个元素指针是否已经提前被释放了
4. 避免重复分配内存
5. 小心使用有return、break语句的宏，确保前面资源已经释放
6. 检查队列中每个成员是否释放

* 【**规则2-20**】禁止引用已经释放的内存空间。

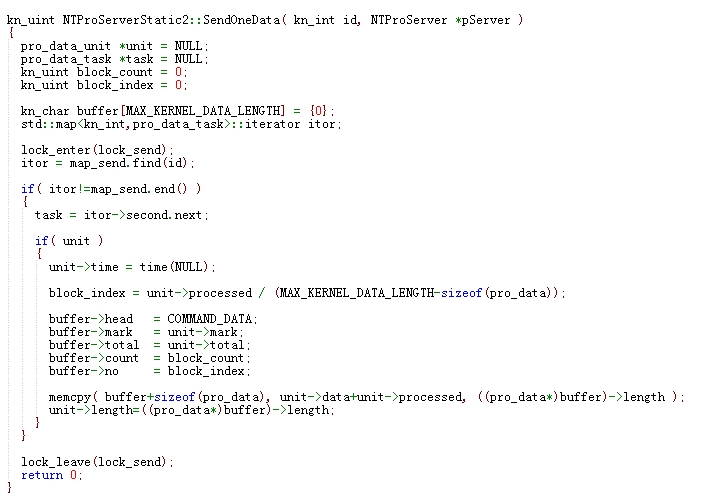
坚持下列措施可以避免引用已经释放的内存空间：

1. 内存释放后，把指针置为NULL；使用内存指针前进行非空判断。
2. 耦合度较强的模块互相调用时，一定要仔细考虑其调用关系，防止已经删除的对象被再次使用。
3. 避免操作已发送消息的内存。
4. 自动存储对象的地址不应赋值给其他的在第一个对象已经停止存在后仍然保持的对象（具有更大作用域的对象或者静态对象或者从一个函数返回的对象） 。

* 【**规则2-21**】必须判断函数返回值（非void），必须做异常处理。

处理顺序：首先为正常结果判断，然后为异常结果判断。

2.2.10 程序版式参照样例



# 3. C/C++语言编程规范

C++语言规则除以下外，遵循《Google的C++编码规范》，见附录附件

## 3.1 头文件

* 【**规则3-1**】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3-2**】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3-3**】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3-1**】头文件中只存放“声明”而不存放“定义”
* 【**规则3-4**】头文件应向稳定的方向包含：产品依赖于平台，平台依赖于标准库。一般来说，应当让不稳定的模块依赖稳定的模块。
* 【**规则3-5**】禁止头文件循环依赖。
* 【**规则3-6**】禁止包含用不到的头文件。
* 【**规则3-7**】禁止在extern "C"中包含头文件。

## 3.2 命名规则

3.2.1 通用命名规则

* 【**规则3-8**】标识符应当直观且可以拼读，可望文知意，不必进行“解码”。
* 【**规则3-9**】标识符的长度应当符合“min-length && max-information”原则。
* 【**规则3-10**】命名规则尽量与所采用的操作系统或开发工具的风格保持一致。
* 【**规则3-11**】 程序中不要出现仅靠大小写区分的相似的标识符。建议名称采用小写字母和下划线。

注：文件夹目录禁止使用除数字、字符下划线以外其他字符，特指空格。

3.2.2 变量

* 【**规则3-12**】 防止局部变量与全局变量同名。
* 【**规则3-13**】 通讯过程中使用的结构，必须注意字节序。
* 【**规则3-14**】 严禁使用未经初始化的变量作为右值，指针变量定义时初始化为NULL。
* 【**建议3-15**】 尽量减少没有必要的数据类型默认转换与强制转换。
* 【**规则3-16**】 静态变量加前缀s\_（表示static）。
* 【**规则3-17**】 如果不得已需要全局变量，则使全局变量加前缀g\_（表示global）。
* 【**规则3-18**】 类的数据成员加前缀m\_（表示member）。

## 3.3 if语句

3.3.1 布尔变量与零值比较

* 【**规则3-19**】 不可将布尔变量直接与TRUE、FALSE 或者1、0 进行比较。

应为：if (flag)或if (!flag)

3.3.2 整形变量与零值比较

* 【**规则3-20**】 应当将整型变量用“==”或“！=”直接与0 比较。

应为：if( 0==value )或if( 0!=value )

3.3.3 浮点变量与零值比较

* 【**规则3-21**】 不可将浮点变量用“==”或“！=”与任何数字比较。

应为：if( (-EPSINON <=x) && (EPSINON >=x) )。其中EPSINON 是允许的误差（即精度）。

3.2.2.4 指针变量与零值比较

* 【**规则3-22**】 应当将指针变量用“==”或“！=”与NULL 比较。

应为：if( NULL==p )或if( NULL!=p )。

# 附录一《Google的C++编码规范》附件

