# 基于谷歌C++编程规范的编码规范文档

## 1. 命名规范

### 1.1 命名方式

* 类型名称使用大写字母开头的驼峰命名法，如 MyClassName。
* 变量和函数名使用小写字母开头的驼峰命名法，如 myVariableName 和 myFunctionName()。
* 常量使用全大写字母和下划线的方式，如 MY\_CONSTANT\_VALUE。
* 命名空间使用全小写字母和下划线的方式，如 my\_namespace。

### 1.2 命名长度

* 类型、函数和变量的命名应当具有描述性，同时也不应过于冗长。建议长度不超过 30 个字符。
* 常量名称可以比较长，但也不应该过度冗长。

### 1.3 命名约定

* 使用有意义的名称，名称应当反映出其所代表的含义。
* 使用可读性强的名称，例如避免使用单个字符的名称，除非是循环计数器或临时变量。
* 避免使用下划线作为标识符的开头，以免与实现细节相混淆。
* 避免使用双下划线，它们已被 C++ 标准保留给编译器使用。
* 避免使用缩写，除非它们是公认的缩写。
* 使用正确的拼写和语法，使用美式英语而不是英式英语。

## 2. 头文件

### 2.1 头文件保护

* 在头文件的开头和结尾都要使用预编译指令保护，以防止重复包含。例如：

|  |
| --- |
| #ifndef \_MY\_PROJECT\_INCLUDE\_PATH\_MY\_HEADER\_H\_  #define \_MY\_PROJECT\_INCLUDE\_PATH\_MY\_HEADER\_H\_  //头文件内容  #endif |

* 预编译指令的名称应当基于头文件的路径和名称，例如： MY\_PROJECT\_INCLUDE\_PATH\_MY\_HEADER\_H\_。

### 2.2 头文件顺序

* 按照以下顺序包含头文件：
* 本模块的头文件（.h）
* C++ 标准库头文件
* 第三方库头文件
* 每个部分按照字母顺序排列。

## 3. 格式化

### 3.1 缩进

* 使用两个空格作为缩进。不要使用制表符。
* 在条件语句、循环语句和函数定义中使用大括号来明确代码块的起始和结束。

### 3.2 行宽

* 建议每行代码的长度不超过 80 个字符。
* 如果某行代码超过了 80 个字符，可以使用换行符和缩进来将其分成多行。
* 运算符可以和下一行代码连在一起，不需要使用换行符。

### 3.3 空格

* 在二元运算符、逻辑运算符、关系运算符、赋值运算符和函数调用中，操作符和操作数 之间应当留有空格。例如：

|  |
| --- |
| int a = 1 + 2;  if (x == 0 && y != 0) {  ...  }  myFunction(arg1, arg2); |

* 在小括号和大括号之间留有一个空格。例如：

|  |
| --- |
| if (x == 0) {  ...  }  myFunction(arg1, arg2); |

* 在 ‘,’ 和操作数之间留有一个空格。例如：

|  |
| --- |
| myFunction(arg1, arg2); |

### 3.4 注释

* 使用 // 进行单行注释，使用 /\* ... \*/ 进行多行注释。
* 在函数或类的定义前应当包含注释，说明函数或类的功能和输入输出参数的含义。
* 对于复杂的算法或代码块，应当添加注释，说明代码的逻辑和目的。
* 避免使用无意义或过于明显的注释。

## 4. 函数

### 4.1 函数设计

* 函数应当尽量短小，只做一件事情，并且按照其名称所表示的含义进行命名。
* 函数应当避免使用全局变量或静态变量，以便于测试和重构。
* 函数参数的数量应当尽可能少，最好不要超过 3 个。

### 4.2 函数调用

* 函数调用应当遵循 "最小化依赖" 原则，即一个函数应当尽可能少的依赖于其他函数或 对象。
* 函数调用应当避免使用裸指针或引用传递，尽量使用智能指针或值传递。

## 5. 类

### 5.1 类设计

* 类应当尽可能地保持简单和可复用性。
* 类应当封装其数据成员，并提供对外的接口来访问和修改这些数据成员。
* 类应当提供适当的构造函数和析构函数，以便于类对象的正确创建和销毁。

### 5.2 继承和多态

* 类的继承应当尽量遵循 "Liskov 替换原则"，即子类对象可以替换父类对象使用。
* 继承关系应当尽量简单，最好不要超过两层。
* 多态应当遵循 "接口隔离原则"，即客户端不应当依赖于不需要的接口。
* 多态应当使用虚函数来实现，而非手动的类型检查和转换。

### 5.3 访问控制

* 类的数据成员应当使用 private 或 protected 进行封装，以便于类的内部和派生类 的访问控制。
* 类的成员函数应当使用 public 或 protected 进行访问控制，以便于类的外部和派生 类的访问控制。

## 6. 异常处理

* 异常应当只用于处理错误或异常情况，而非用作正常流程控制的手段。
* 异常应当尽可能的精细和具体，而非粗略和笼统。
* 异常处理应当遵循 "资源获取即初始化" 原则，即在异常抛出前应当确保已经释放了所 有已经获取的资源。

## 7. 内存管理

* 应当尽可能避免使用裸指针进行内存管理，而应当使用智能指针或容器等 RAII 技术。
* 在使用裸指针时，应当遵循 "内存所有权" 原则，即每个对象只有一个所有者，避免出 现多个对象共享同一个指针的情况。
* 在使用动态内存分配时，应当遵循 "异常安全" 原则，即在分配内存时如果发生异常， 应当确保已经释放了之前已经分配的内存。

## 8. 并发编程

* 并发编程应当尽可能避免共享资源，而应当使用消息传递或分离式的设计。
* 并发编程应当遵循 "锁粒度" 原则，即尽可能地减小锁的范围，避免对整个对象进行锁 定。
* 并发编程应当遵循 "死锁预防" 原则，即避免在持有一个锁的情况下再去请求另一个锁。

## 9. 测试和调试

* 对于每个函数和类，应当编写相应的单元测试，保证其正确性和稳定性。
* 对于整个系统，应当编写相应的集成测试和性能测试，保证其正确性和高效性。
* 在编写测试和调试代码时，应当遵循相同的编码规范和设计原则。