**代码及注释风格**

2019

# C/C++

## 说明

从编程语言的发展上看，C语言基本上是比汇编语言高级一些，可以认为是最低级的高级语言。C是面向过程的编程语言，而且语法相对简单，也就造成了C语言代码的自由度非常高。C++作为几乎完美兼容C的高级语言，自由度也非常高。自由度高就意味着不同的人写的代码之间风格差异会很大，阅读不同风格代码的感受差异也非常大。因而在一个团队中，保持代码风格的一致性也就变得相当重要。

本文的目的在于规范团队中的代码风格，使得大家在代码风格上保持相对一致。这里所谓的代码风格主要包括命名规则以及注释方法，其它的方面也许会提到，以后也可以再补充。

## 命名规则

命名规则主要是说一个变量、函数、模块甚至是文件应该如何命名。最常见的命名规则有两种，一种是山形方法（微软一贯的命名方法，具体名称似乎是匈牙利命名法？），即除第一个单词外，其余单词的首字母大写，不使用下划线；另一种就是使用下划线分隔单词，而单词全部使用小写（谷歌采用的就是这种方法）。当然，这两种方法都是基本原则，特殊的地方需要特殊处理。比如微软要求变量的开头是类型简写。

我们主要采用山形方法命名规则，但具体情形有所变化。

表格 1 基本符号定义

|  |  |
| --- | --- |
| 符号 | 含义描述 |
| Word | 全小写的单词 |
| Word | 首字母大写的单词 |
| Words | 多个首字母大写的单词相连 |
| \_words | 多个小写单词相连，以下划线分隔 |
| \_WORDS | 多个大写单词相连，以下划线分隔 |

### 常量

常量一般指的是使用#define或者const定义的符号，其值在编译时即可确定。

使用#define定义的符号全部使用大写，单词之间使用下划线分隔。一般情况下单词不要简写。即：

**CONST\_NAME := \_WORDS**

例如：

#define PI 3.1416

#define BUFFER\_SIZE 1024

在C++中，这种使用#define定义的常量尽量用const替换。例如：

const float PI = 3.1416;

const int BUFFER\_SIZE = 1024;

常量的定义一般放在.c或.cpp文件的开头。如果需要放在头文件中，则使用#define定义。

### 全局变量

全局变量主要有两类：一类是全局可见，一类是加了static关键字（在函数或类外面定义）从而只有本文件可见。

全局变量以‘g\_’开头，然后是小写单词，然后是首字母大写的单词。即：

**GLOBAL\_VAR := g\_wordWords**

例如：

int g\_bufferSize = 1024;

struct TestStruct g\_testStruct;

Person g\_headMaster(“Peter Smith”);

注：尽量避免全局变量的使用，在C++中可以使用namespace限定全局变量的作用域。

静态全局变量的命名与全局变量相似，但以’s\_’开头。

**STATIC\_GLOBAL\_VAR := s\_wordWords**

### 局部变量

局部变量指的是在函数内部定义的变量，其作用域为整个函数或者代码块。其命名规则采用山形命名方法，即首个单词小写，其后单词首字母大写，避免使用下划线。即：

**LOCAL\_VAR := wordWords**

如：

bool isFound = false;

int bufferSize = 1024;

char buffer[1024];

vector<int> workHours;

在函数内部定义的静态变量命名规则与上述方法相同。除非必要，不使用函数内部静态变量。

### 结构体成员变量

结构体成员变量的命名方法与局部变量的命名方法相同。

**STRUCT\_MEMBER := LOCAL\_VAR**

### 类成员变量

类成员变量的命名方法与局部变量的命名方法相似：公有成员变量的命名方法不做任何改动；私有成员变量和保护成员变量加以’m\_’或’\_’前缀。

**PUBLIC\_CLASS\_MEMBER := LOCAL\_VAR**

**PROTECT\_CLASS\_MEMBER := m\_wordWords**

**PRIVATE\_CLASS\_MEMBER := PROTECT\_CLASS\_MEMBER**

类静态常量命名与常量的命名规则相同。

**CLASS\_CONST\_NAME := CONST\_NAME**

类静态变量的命名与普通成员变量命名规则相同。

**CLASS\_STATIC\_MEMBER := STATIC\_GLOBAL\_VAR**

### 结构体与类名

结构体与类的命名与局部变量命名规则相似，但第一个单词的首字母要大写。

**STRUCT\_NAME := Words**

**CLASS\_NAME := STRUCT\_NAME**

嵌套定义的结构体与类命名与普通结构体或类的命名规则相同，但如果只有一个单词可以全部小写。

### 命名空间

命名空间的命名规则与类名相同。

**NAMESPACE\_NAME := CLASS\_NAME**

### 全局函数

全局函数的命名规则如下：单词全部小写，并且以下划线进行分隔。

**GLOBAL\_FUNCTION\_NAME := \_words**

在C++中，尽量使用命名空间限定全局函数。

命名空间中的全局函数，其命名规则与类名相同。

**NAMESPACE\_FUNCTION\_NAME := CLASS\_NAME**

### 类成员函数

公有成员函数命名方法与类名相同；保护成员函数和私有成员函数的命名方法与局部变量相同。

**PUBLIC\_CLASS\_MEMBER\_FUNCTION := NAMESPACE\_FUNCTION\_NAME**

**PROTECT\_CLASS\_MEMBER\_FUNCTION := LOCAL\_VAR**

**PRIVATE\_CLASS\_MEMBER\_FUNCTION := PROTECT\_CLASS\_MEMBER\_FUNCTION**

### 文件

文件的命名规则与全局函数的命名规则相似，但分隔符可以使用’\_’或’-‘。

## 注释

注释非常重要，不仅对于别人重要，对自己也重要。

### 文件注释

文件注释指的是对一个源文件总体的注释，比如一个类的实现源文件。注释应该对文件的主要内容进行说明和描述。文件注释置于文件顶部。推荐格式如下：

|  |
| --- |
| /\* Description:  \* Author:  \* Date:  \*/ |

### 变量注释

对于常量和全局变量，用简练的语言说明其意义。

对于局部变量，对比较重要的进行注释。

对于成员变量，说明其意义，以及需要特别注意的地方。

### 函数注释

函数头注释应该包括：对函数的描述（前置条件、后置条件等）、参数列表的描述、返回值的描述。推荐格式如下：

|  |
| --- |
| /\* Description  \* params:  \* param1 [in] description  \* param2 [in/out] description  \* param3 [out] description  \* return:  \* value, range, and description  \*/ |

不要求对所有的函数进行函数头注释，但是在头文件里进行声明的函数应该写函数头注释，写在头文件中。

函数体注释：除了对其变量进行注释以外，在函数的实现体里插入少量注释说明代码段的作用，增加可读性。

对于成员函数，至少应该对所有的公有成员函数进行函数头注释。

### 结构体和类注释

对于结构体、类、枚举类型等，需要有一定的注释说明其含义或意义，以及与其它结构体或类的关系。