**C++编码规范**

**C++编码规范总则**

本规范总则的示例都以C++语言为背景，采用以下的术语描述：

**规则：**编码时强制必须遵守的原则。

**建议：**编码时必须加以考虑的原则。

**说明：**对比规则或建议进行必要的解释。

**示例：**对此规则或建议从正、反两个方面给出例子。

**1 排版**

**规则1-1：**程序块要采用缩进风格编写，缩进的空格数为1个。

说明：对于由开发工具自动生成的代码可以有不一致。

**规则1-2：**相对独立的程序块之间、变量说明之后必须加空行。

示例：如下例子不符合规范。

if (!vaild\_ni(ni))

{

...// program code

}

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

应书写如下：

if (!vaild\_ni(ni))

{

...// program code

}

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

**规则1-3：**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，是排版整齐，语句可读。

示例：

act\_task\_table[frame\_id \*STAT\_TASK\_CHECK\_NUMBER + index].occupied

= stat\_poi[index].occupied;

if ((taskno < max\_act\_task\_number)

&& (n7stat\_stat\_item\_valid(stat\_item)))

{

...// program code

}

**规则1-4：**不允许把多个语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

rect.length = 0; rect.width = 0;

应书写如下：

rect.length = 0;

rect.width = 0;

**规则1-5：**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{}。

示例：如下例子不符合规范。

if (pUserCR == NULL) return;

应书写如下：

if (pUserCR == NULL)

{

return;

}

**2 注释（注释均采用中文。）**

**规则2-1：**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。

**规则2-2：**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：生成日期、作者姓名、内容、功能、与其他文件的关系等，头文件的注释中还应有函数功能简要说明。

示例：下面这段头文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

文件：

作者：

日期：

描述： //用于详细说明此程序文件完成的主要功能，与其他模块

//或函数的接口，输出值、取值范围、含义及参数间的控

//制、顺序、独立或依赖等关系

其他： //其它内容的说明

函数列表： //主要函数列表，每条记录应包括函数名及功能简要说明

1.……

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**规则2-3：**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：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。

示例：下面这段头文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

文件：test.cpp

作者：包小米

日期：2019-6-14

描述： //模块描述

其他： //其他内容说明

函数列表： //主要函数及其功能

……

历史修改记录：

<作者> <日期> <描述>

包小米 2019-6-14 建立文件

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

说明：描述一项描述本文件的内容、功能、内部各部分之间的关系及本文件与其它文件关系等。

**规则2-4：**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。

示例：下面这段函数的注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名： //函数名称

描述： //函数功能、性能等的描述

调用的函数： //被本函数调用的函数清单

调用本函数的函数： //调用本函数的函数清单

被访问的表： //被访问的表

被修改的表： //被修改的表

输入参数： //输入参数说明，包括每个参数的作

//用、取值说明及参数间关系。

输出参数： //对输出参数的说明

函数返回值： //函数返回值的说明

其他： //其他说明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**规则2-5：**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。

示例：可按如下形式说明枚举/数据/联合结构。

/\*sccp接口使用的私有消息枚举\*/

enum SCCP\_USER\_PRIMITIVE

{

N\_UNITDATA\_IND, /\* sccp notify sccp user unit data come \*/ N\_NOTICE\_IND, /\* sccp notify user the network can not \*/

/\* transmission this message \*/

N\_UNITDATA\_REQ, /\* sccp user's unit data transmission request\*/

}

**规则2-6：**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。

示例：

/\* SCCP传输时的错误代码\*/ // 变量作用、含义

/\* 0 - 成功 1 - GT表单错误 \*/

/\* 2 - GT错误 Others - 待用 \*/ // 变量取值范围

/\* 只有本模块的函数SCCPTranslate()可以修改 \*/

/\* 其他模块只有通过调用函数GetGTTransErrorCode()来访问它 \*/ // 使用方法

BYTE g\_GTTranErrorCode;

**建议2-1：**将注释与其上面的代码用空行隔开。

/\*注释1 某某\*/

program code one

/\*注释2 某某\*/

program code two

应书写如下：

/\*注释1 某某\*/

program code one

/\*注释2 某某\*/

program code two

**建议2-2：**避免在一行代码或表达中间插入注释，否则容易使代码可理解性变差。

说明：除非必要，不应在代码或表达中间插入注释，否则容易使代码可理解性变差。

**建议2-3**：通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的。

说明：清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

**建议2-4：**在代码的功能、意图层次上进行注释，提供有用、额外的信息。

说明：注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

示例：如下注释意义不大。

/\* 如果receive\_flag是真 \*/

if(receive\_flag)

而如下的注释则给出了额外有用的信息。

/\* 如果mtp收到了链路信息\*/

if(receive\_flag)

**建议2-5：**在程序块的结束行右方加注释标记，以表明某程序块的结束。

说明：当代码段较长，特别是多重嵌套时，这样做可以使代码更清晰，更便于阅读。

示例：参见如下例子。

if(...)

{

program code

while (index < MAX\_INDEX)

{

program code

}/\*while (index < MAX\_INDEX)在此处结束\*/ //指明该条while语句结束

}/\*if(...)在此处结束\*/ //指明是哪条if语句结束

**3标识符命名**

**规则3-1：**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。

示例：如下单词的缩写能够被大家基本认可。

temp 可缩写为 tmp ;

flag 可缩写为 flg ;

statistic 可缩写为stat;

increment 可缩写为 inc;

message 可缩写为 msg ;

**规则3-2：**命名规范必须与所使用的系统风格保持一致，并在同一项目中统一，比如采用UNIX的 全小写加下划线的风格或大小写混排的方式，不要使用大小写与下划线混排的方式，用作特殊 标识如标识成员变量或全局变量的m\_和g\_，其后加上大小写混排的方式是允许的。

示例：

Add\_User 不允许，add\_user、AddUser、m\_AddUser 允许。

**建议3-1**：除非必要，不要用数字或较奇怪的字符来定义标识符。

示例：如下命名，使人产生疑惑。

#define \_EXAMPLE\_0\_TEST\_

#define \_EXAMPLE\_1\_TEST\_

void set\_sls00(BYTE sls);

应改为有意义的单词命名

#define \_EXAMPLE\_UNIT\_TEST\_

#define \_EXAMPLE\_ASSERT\_TEST\_

void set\_udt\_msg\_sls(BYTE sls);

**建议3-2：**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。

**建议3-3：**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。

**4可读性**

**规则4-1：**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。

示例：

下列语句中的表达式

word = (high << 8) | low    (1)

if ((a | b) && (a & c))     (2)

if ((a | b) < (c & d))      (3)

如果书写为

high << 8 | low

a | b && a & c

a | b < c & d

由于

high << 8 | low = ( high << 8) | low,

a | b && a & c = (a | b) && (a & c)，

(1)(2)不会出错，但语句不易理解；

a | b < c & d = a | （b < c）  & d，(3)造成了判断条件出错。

**规则4-2：**避免使用不易理解的数字，用有意义的标识来代替。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。

示例：如下的程序可读性差。

if(Trunk[index].trunk\_state == 0)

{

Trunk[index].trunk\_state = 1;

...  // program code

}

应书写如下。

#define TRUNK\_IDLE 0

#define TRUNK\_BUSY 1

if(Trunk[index].trunk\_state == 0)

{

Trunk[index].trunk\_state = 1;

...  // program code

}