c++编程规范

一.程序的版本

1.程序块要采用缩进风格编写，缩进的空格数为4个。

说明： 由开发工具自动生成的代码可能不一致，但如果开发工具可以配置，则应该统一配置缩进为4个空格。

2.缩进或者对齐只能使用空格键，不可使用TAB键。

说明： 使用TAB键需要设置TAB键的空格数目是4格。

3.相对独立的程序块之间、变量说明之后必须加空行。

说明：

以下情况应该是用空行分开：

1）函数之间应该用空行分开；

2）变量声明应尽可能靠近第一次使用处，避免一次性声明一组没有马上使用的变量；

3）用空行将代码按照逻辑片断划分；

4）每个类声明之后应该加入空格同其他代码分开。

4.较长的语句（>80字符）要分成多行书写。

说明：

以下情况应分多行书写：

1）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。

2）若函数或过程中的参数较长，则要进行适当的划分。

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。

eg:

```
perm_count_msg.head.len = NO7_TO_STAT_PERM_COUNT_LEN
                        + STAT_SIZE_PER_FRAM * sizeof( _UL );

act_task_table[frame_id * STAT_TASK_CHECK_NUMBER + index].occupied
          = stat_poi[index].occupied;
```

```
act_task_table[taskno].duration_true_or_false
          = SYS_get_sccp_statistic_state( stat_item );

report_or_not_flag = ((taskno < MAX_ACT_TASK_NUMBER)
                  && (n7stat_stat_item_valid (stat_item))
                 && (act_task_table[taskno].result_data != 0));

n7stat_str_compare((BYTE *) & stat_object,
               (BYTE *) & (act_task_table[taskno].stat_object),
              sizeof (_STAT_OBJECT));

n7stat_flash_act_duration( stat_item, frame_id *STAT_TASK_CHECK_NUMBER
                                    + index, stat_object );

if ((taskno < max_act_task_number)
&& (n7stat_stat_item_valid (stat_item)))
{
... // program code
}
for (i = 0, j = 0; (i < BufferKeyword[word_index].word_length)
                && (j < NewKeyword.word_length); i++, j++)

{
... // program code
}
for (i = 0, j = 0;
(i < first_word_length) && (j < second_word_length);
i++, j++)

{
... // program code
}
```

5.不允许把多个短语句写在一行中，即一行只写一条语句。

说明：

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

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.代码行之内应该留有适当的空格。

说明：

采用这种松散方式编写代码的目的是使代码更加清晰。代码行内应该适当的使用空格，具体如下：

1）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。象if、for、while 等关键字之后应留一个空格再跟左括号‘（’，以突出关键字。

2）函数名之后不要留空格，紧跟左括号‘(‘ ，以与关键字区别。

3）‘(‘向后紧跟，‘)’、‘，’、‘;’向前紧跟，紧跟处不留空格。

4）‘,’之后要留空格，如Function(x, y, z)。如果‘;’不是一行的结束符号，其 后也要留空格，如for (initialization; condition; update)。

5）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。

6）一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。

7）象“ [] ”、“.”、“->”这类操作符前后不加空格。

8.程序块的分界符（如C/C++语言的大括号‘{’和‘}’）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.源文件头部应进行注释，列出：生成日期、作者、模块目的/功能等。

示例：

下面这段源文件的头注释比较标准，可以不局限于此格式，但上述信息要包含在内。

/*********

FileName: test.cpp

Author: Version : Date:

Description: // 模块描述

Version: // 版本信息

Function List: // 主要函数及其功能

```
    1. -------
```

History: // 历史修改记录

```
 <author>  <time>   <version >   <desc>
 David    96/10/12     1.0     build this moudle
```

**********/

说明：

Description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。

History是修改历史记录列表，每条修改记录应包括修改日期、修改者及修改内容简述。

也可以采用javadoc 风格的文档注释，这里不再举例，下同。

2.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、返回值等。

3.注释应该和代码同时更新，不再有用的注释要删除。

4.注释的内容要清楚、明了，不能有二义性。

5.避免在注释中使用非常用的缩写或者术语。

6.注释的主要目的应该是解释为什么这么做，而不是正在做什么。如果从上下文不容易看出作者的目的，说明程序的可读性本身存在比较大的问题，应考虑对其重构。

7.避免非必要的注释。

eg:

ClassA *pA = new ClassA(); //创建新实

…

delete pA; //销毁对象

8.注释的版本

说明：

注释也需要与代码一样整齐排版

1）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。

2）注释与所描述内容进行同样的缩排。

3）将注释与其上面的代码用空行隔开。

4）变量、常量、宏的注释应放在其上方相邻位置或右方。

9.对于所有有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。

10.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可放在此域的右方。

eg: // sccp interface with sccp user primitive message name

enum SCCP_USER_PRIMITIVE

```
{
  N_UNITDATA_IND, /* sccp notify sccp user unit data come */
  N_NOTICE_IND,   /* sccp notify user the No.7 network can not */
                  /* transmission this message */
  N_UNITDATA_REQ, /* sccp user's unit data transmission request*/

};
```

11.对重要变量的定义需编写注释，特别是全局变量，更应有较详细的注释，包括对其功能、取值范围、以及存取时注意事项等的说明。

12.分支语句（条件分支、循环语句等）需编写注释。

说明：

这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有

13.注释不宜过多，也不能太少，源程序中有效注释量控制在20%~30%之间。

说明：

注释是对代码的“提示”，而不是文档，不可喧宾夺主，注释太多会让人眼花缭乱。

三.表示符命名

1.命名尽量使用英文单词，力求简单清楚，避免使用引起误解的词汇和模糊的缩写，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

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。

说明：

1）如在UNIX系统，可采用全小写加下划线的风格或大小写混排的方式，但不能使用大小写与下划线混排的方式。

2）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。

eg：Add_User不允许，add_user、AddUser、m_AddUser允许。

3.变量的命名可参考“匈牙利”标记法（Hungarian Notation）：TypePrefix+Name

C++程序不建议采用匈牙利命名法。因为C++本身就是强类型语言，不需要像C一样用匈牙利命名法来强调变量类型。有两个匈牙利命名法可以保留：m_xxxx 表示类的成员变量，g_xxx 表示全局变量。

4.常量、宏和模板名采用全大写的方式，每个单词间用下划线分隔。

5.枚举类型enum 常量应以大写字母开头或全部大写。

6.命名中若使用了特殊约定或缩写，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。

7.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

8.对于变量命名，禁止取单个字符（如i、j、k...），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但i、j、k作局部循环变量是允许的。

说明：

变量，尤其是局部变量，如果用单个字符表示，很容易敲错（如i写成j），而编译时又检查不出来，有可

9.除非必要，不要用数字或较奇怪的字符来定义标识符。

避免使用看上去相似的名称，如“l”、“1”和“I”看上去非常相似。

10.函数名以大写字母开头，采用谓-宾结构（动-名），且应反映函数执行什么操作以及返回什么内容。

说明：

函数在表达式中使用，通常用于`if` 子句，因此它们的意图应一目了然。

示例：

不好的命名：`if (CheckSize(x))`

没有帮助作用，因为它没有告诉我们`CheckSize`是在出错时返回`true` 还是在不出错时返回`true`。

好的命名：if (ValidSize(x))

则使函数的意图很明确。

11.类、结构、联合、枚举的命名须分别以C、S、U、E开头，其他部分遵从一般变量命名规范。

四.可读性

1.用括号明确表达式的操作顺序，避免使用默认优先级。

2.不要编写太复杂，多用途的符合表达式。

3.涉及物理状态或者含有物理意义的常量，避免直接使用数字，必须用有意义的枚举或常量来代替。

eg：

```
const int TRUNK_IDLE = 0;
const int TRUNK_BUSY = 1;
```

```
if (Trunk[index].trunk_state == TRUNK_IDLE)
{
  Trunk[index].trunk_state = TRUNK_BUSY;
  ...  // program code

}
```

4.禁止使用难以理解，容易产生歧义的语句。

五.变量、结构

1.尽量少用全局变量，尽量曲调没必要的公共变量。

说明：

公共变量是增大模块间耦合的原因之一，故应减少没必要的公共变量以降低模块间的耦合度。

2.变量，特别是指针变量，被创建之后应当及时把它们初始化，以防止把未被初始化的变量当成右值使用。

说明：在C/C++中引用未经赋值的指针，经常会引起系统崩溃。

3.仔细设计结构中元素的布局与排列顺序，使结构容易理解、节省占用空间，并减少引起误用现象。

4.留心具体语言及编译器处理不同数据类型的原则及有关细节。

说明：

如在C语言中，static局部变量将在内存“数据区”中生成，而非static局部变量将在“堆栈”中生成。这

5.尽量减少没有必要的数据类型默认转换与强制转换。

说明：

当进行数据类型强制转换时，其数据的意义、转换后的取值等都有可能发生变化，而这些细节若考虑不周

6.当声明用于分布式环境或不同CPU间通信环境的数据结构时，必须考虑机器的字节顺序、使用的位域及字节对齐等问题。

六.函数、过程

1.调用函数要检查所有可能的返回情况，不应该的返回情况要用ASSERT来确

认。

2.编写可重入函数时，应注意局部变量的使用（如编写C/C++语言的可重入函数时，应使用auto即缺省态局部变量或寄存器变量）。

3.调用公共接口函数时，调用者有保障调用参数符合要求的义务。作为一种防御性的编程风格，被调用函数也应该对传入参数做必要的安全检查。

4.函数的规模尽量限制在100行以内。（不包括空格行和注释）

5.一个函数仅完成一件功能。

说明：

多功能集于一身的函数，很可能使函数的理解、测试、维护等变得困难。

6.不能用ASSERT代替必要的安全处理代码，确保发布版的程序也能够合理地处理异常情况。

7. 尽量写类的构造、拷贝构造、析构和赋值函数，而不使用系统缺省的。

说明：

编译器以“位拷贝”的方式自动生成缺省的拷贝构造函数和赋值函数，倘若类中含有指针变量，那么这两个

8.对于不需要拷贝构造函数时，应显式地禁止它，避免编译器生成默认的拷贝构造函数。

9.谨慎使用与程序运行的环境相关的系统函数。

10.禁止编写依赖于其他函数内部实现的函数。

说明：

此条为函数独立性的基本要求。由于目前大部分高级语言都是结构化的，所以通过具体语言的语法要求与

11.检查函数所有参数与非参数的有效性。

说明：

1）函数的输入主要有两种：一种是参数输入；另一种是全局变量、数据文件的输入，即非参数输入。函数在使用输入之前，应进行必要的检查。

2）不应该的入口情况要用ASSERT来确认。

3）有时候不处理也是一种处理，但要明确哪些情况不处理。try...catch 是一种常用的不处理的处

理手段。

12.函数实现中不改变内容的参数要定义成const。

13.函数的返回值要清楚、明了，让使用者不容易忽视错误情况。

七.C++专用规范

1.在高警告级别下干净地编译。

使用编译器的最高警告级别。要求干净的（没有警告的）构建（build）并理解所有的警告。通过修改代

2.确保资源为对象所占有，使用显式的RAII和智能指针。

C++在语言层面强制的构造/析构恰好与资源获取/释放这对函数相对应，在处理需要调用成对的获取/释

最好用智能指针来保存动态分配的资源，而不要用原始指针。

3.主动使用const，避免使用宏。

应该尽可能的使用常量而不用变量，另外在定义数值的时候，应该把const做为默认的选项。它是安全的

宏无视作用域，无视类型系统，无视所有其它的语言特性和规则，并从#define处开始将该符号劫持。只

4.合理使用组合(composition)和继承(inheritance)。

继承是C++中耦合度最强的关系之一。软件工程的一条重要原则是尽量减少耦合，在组合和继承都能均可

1）在不影响调用代码的同时也更灵活。

2）编译期绝缘性好，编译时间也能缩短。

3）代码不可预测程度降低（有些类不适合作为基类）。

5.尽可能局部地声明变量。

尽可能局部地声明每个变量，这通常是在程序具备了足够的数据来初始化变量之后，并紧接着首次使用该

例外：

1）有时将变量从循环内提出到循环外是有益的。

2）由于常量不增加状态，因此本条对常量不适用。

6.通过值，（智能）指针，或引用适当地取得参数。

对仅用于输入的参数来说：

1）始终给仅用于输入的指针或引用参数加上const限定符。

2）最好是通过原始类型（例如：char，float）和可以通过值来复制并且复制成本低的值对象（例如：Point，complex）来取得参数。

3）对其它自定义类型的输入，最好是通过const引用来取得。

4）如果函数需要参数的复本，那么可以考虑用传递值来代替传递引用。从概念上说，这等价于取得一个const引用再做一次复制，它可以帮助编译器更好地优化掉临时对象。

对输出或输入/输出参数来说：

1）如果参数是可选的（因此调用方可以传递空指针来表示"不可用"或"不关心"的值），或者函数要保存指针的一个复本或操控参数的所有权，那么最好是通过（智能）指针传递。

2）如果参数是必需的，而且函数无需保存指向该参数的指针或无需操控参数的所有权，那么最好是通过引用传递。这表明该参数是必需的，并让调用方来负责提供一个有效的对象。

7.不要在头文件中定义具有链接属性的实体。

重复导致膨胀：

具有链接属性的实体，包括名字空间层级的变量或函数，需要占用内存。把此类实体定义在头文件中会导致编译错误或内存浪费。应该把具有链接属性的实体放在实现文件中。

下面这些具有外部链接属性的实体可以放在头文件中：

1）内联函数：虽然它们具有外部链接属性，但是链接器会保证不拒绝链接多个复本。除此之外，它们的行为和普通的函数完全一样。

2）函数模板（Function templates）：与内联函数相似，除了重复的复本是可接受的之外（最好是完全一样的），模板实例化的行为与普通的函数一样。而一个好的编译系统会消除无用的复本。

3）类模板的静态数据成员：这对链接器来说可能有点粗暴，不过只需在头文件中定义它们，则可让编译器和链接器来处理剩余的事情。

8.尽量用异常来报告错误。

与错误码相比，要尽量用异常来报告错误。对一些无法使用异常的错误，或者一些不属于错误的情况，可以用状态码（status code，例如：返回码，errno）来报告。如果不可能或不需要从错误中恢复，那么可以使用其它方法，比如正常或非正常地终止程序。

在C++中，和用错误码来报告错误相比，用异常来报告错误具有许多明显的优势，所有这些都使得编出来的代码更健壮：

1）程序员不能无视异常：错误码的最糟糕的缺点就是在默认情况下它们会被忽略；即使是给予错误码微不足道的关注，都必须显式地编写代码，以接受错误并做出反应。程序员因为偶然（或因为懒惰）而忘记关注错误码是很平常的事。这使得代码复查变得更困难。程序员不能无视异常；

要忽略异常，必须显式地捕获它（即使只是用catch(...)），然后不对之进行处理。

2）异常会自动传递：默认情况下错误码不会跨作用域传递；为了把一个低层的错误码通知高层的调用函数，程序员必须在中间层的代码中显式地手工编写代码以传递该错误。异常会自动地跨作用域传递，直到被处理为止。（“试图使每个函数都成为防火墙并不是一种好办法。”[Stroustrup94, §16.8]）

3）异常处理从主控制流中去除了错误处理及恢复：错误码的检测及处理，一旦要写的话，就必须夹杂在主控制流中（并使之变得难以理解）。这使得主控制流以及错误处理的代码都更难以理解和维护。异常处理很自然地把错误检测及恢复移到醒目的catch代码块中，即它使错误处理既醒目，又易于使用，而不是纠缠在主控制流中。

附录A 程序效率

1.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：

1）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）不能一味地为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。

2.局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。

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。

4.循环体内工作量最小化。

说明：

1）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。

2）在多重循环中，应将最忙的循环放在最内层，这样可以减少CPU切入循环层的次数，从而提高效率。

5.对模块中函数的划分及组织方式进行分析、优化，改进模块中函数的组织结构，提高程序效率。

说明：

软件系统的效率主要与算法、处理任务方式、系统功能及函数结构有很大关系，仅在代码上下功夫一般不

6.避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中。

说明：

目的是减少判断次数。循环体中的判断语句是否可以移到循环体外，要视程序的具体情况而言，一般情况

7.在逻辑清楚且不影响可读性的情况下，代码越少越好。

8.尽量使用标准库函数，不要"发明"已经存在的库函数。

9.要尽量重用已有的代码，直接调用已有的API 。

附录B 质量保证

1.只引用属于自己的存贮空间。

说明：

若模块封装得较好，那么一般不会发生非法引用他人的空间的情况。

2.防止引用已经释放的内存空间。

说明：

在实际编程过程中，稍不留心就会出现在一个模块中释放了某个内存块（如C语言指针），而另一模块在

3.过程/函数中动态分配的资源（包括内存、文件等），在过程/函数退出之前要释放。

说明：

防止内存泄露。

4.充分理解new/delete，malloc/free 等指针相关的函数的意义，对指针操作时需小心翼翼

5.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

6.要时刻注意易混淆的操作符。当编完程序后，应从头至尾检查一遍这些操作符，以防止拼写错误。

说明：

形式相近的操作符最容易引起误用，如C/C++的“=”与“==”、“|”与“||”、“&”与“&&”等，若拼写错了

7.条件表达式要把常量写在前面。

说明：

习惯写`if (MAX_COUNT == nIndex)` 就不会发生`if (nIndex = MAX_COUNT)`的错误

8.有可能的话，if语句尽量加上else分支，对没有else分支的语句要小心对待；switch语句必须有default分支。

9.尽量少用goto语句。

说明：

1）goto语句会破坏程序的结构性，所以除非确实需要，最好不使用goto语句。

2）使用goto 语句时，不能往回跳。

3）尽量不要用多于一个的goto语句标记。

10.不使用与硬件、操作系统、或编译器相关的语句，而使用建议的标准语句，以提高软件的可移植性和可重用性。

11.时刻注意表达式是否会上溢、下溢。

12.使用第三方提供的软件开发工具包或控件时，要注意以下几点：

1）充分了解应用接口、使用环境及使用时注意事项。

2）不能过分相信其正确性。

3）除非必要，不要使用不熟悉的第三方工具包与控件。

说明：

使用工具包与控件，可加快程序开发速度，节省时间，但使用之前一定对它有较充分的了解，同时第三方

13.资源文件（多语言版本支持），如果资源是对语言敏感的，应让该资源与源代码文件脱离，具体方法有下面几种：使用单独的资源文件、DLL文件或其它单独的描述文件（如数据库格式）。

14.打开编译器的所有告警开关对程序进行编译，并且要确认、处理所有的编译告警。

15.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

16.如果可能，单元测试要覆盖98%以上的代码，尽可能早地发现和解决问题。

说明：1

）尽早发现问题可以避免问题的扩大化，减少运维成本。

2）开发人员要树立"不要依赖测试人员"的观念，且不要抱侥幸心理，会出问题的地方总是会出问题的。