C++语言编码规范

一.命名规则：

1.名称

（1）类的名称：

a.类的名称通常是名词。类的名称要告诉我们，这个类是什么。

b.类的名自不需要告诉我们，它从哪个类继承而来的。

c. 有时候加个后缀是很有用的。

2）方法和函数的名称：

a. 方法和函数通常都要执行某种行为，因此，名称要能清楚的说明它做什么， 这样也可以很容

易的区别函数和数据。

b. 函数名总以动词开头，后面跟随其它名称。这样看起来更自然些。

c. 可以加一些必要的后缀：

d. 前缀也同样有用：

（3）含有度量单位的名称：（适用于C/C++）

如果一个变量用于表示时间，重量或其它度量单位，应把度量单位添加到名

称中，以便开发人员更早一步发现问题。

（4）缩写名称不要全部大写（适用于C/C++）

无论是什么缩写名称，我们总以一个大写字母开头，后面跟随的字母全部用小写。

2.类的命名：（适用于C++）

（1）用大写字母作为单词的分隔，每个单词的首字母大写，其它字母均小写。

（2）名字的第一个字母应大写

（3）不含有下划线 ('\_')。

3.类库（或程序库）命名（适用于C/C++）

  (1)使用命名空间防止名字冲突。

  (2)如果编译器没有实现命名空间，需要用前缀来避名名字冲突，不过前缀不要过长（2个字母比较好）。

4.方法和函数的命名（适用于C++）

使用与类名相同的规则

5.类属性的命名（适用于C++）

（1）属性（通常是非公有数据成员）名字以字母'm'开头。

（2）在'm(m\_)'后面，使用与类名相同的规则。

（3）'m(m\_)' 总是位于其它修饰符（如表示指针的 'p'）的前面。

6.方法和函数参数的命名（适用于C++）

（1）第一个字母必须小写

（2）第一个字母后面的单词使用与类名相同的规则

7.局部变量的命名（适用于C/C++）

(1) 所有字母都用小写

（2）使用下划线 '\_' 作为单词的分隔。

8.指针变量的命名前缀（适用于C/C++）

（1）指针变量多数情况应在前面加'p'

(2) 星号 '\*' 应靠近类型，而不是变量名。

9.引用变量和返回饮用函数的命名前缀（适用于C++）

（1）引用必须用 'r'作前缀修饰。

10. 全局变量的命名前缀（适用于C/C++）

全局变量总是以 'g(g\_)' 作为前缀。

11. 全局常量的命名（适用于C/C++）

全局常量全部大写，并以下划线 '\_' 分隔单词。

12. 静态变量的命名前缀（适用于C++）

静态变量以 's' 作为前缀。

13. 自定义类型（typedef）的命名（适用于C/C++）

（1）类型定义名称指的是用typedef定义的名称。

（2）类型定义名称使用与类名相同的规则，并使用Type作为后缀。

14. 宏定义的命名（适用于C/C++）

所有单词的字母都用大写，并使用下划线 '\_' 分隔.

15. C 函数的命名（适用于C/C++）

（1）C++项目中，应尽量少用C函数。

（2）C函数使用GNU规范，所有字母都使用小写，并用下划线 '\_' 作为单词的分隔。

（3）特别的，为了赚容C/C++，在必要的时候，在C++中应以下面的格式定义C函数：

   extern “C” int some\_bloody\_function();

或在C/C++中推荐使用下面的格式：

  #ifdef\_\_cplusplus\_\_

   extern “C”{

   #endif

   int

  some\_bloody\_function()

   {

   }

   #ifdef\_\_cplusplus\_\_

   }

   #endif

16. 枚举的命名（适用于C/C++）

所有字母都大写，并用下划线 '\_' 作为单词分隔。

二.排版规则：

1.布局和模板：

（1）类的布局模板：

请使用下面的模板来创建一个新的类:

/\*\*

\* 用一行来描述类

\*

\*#include "XX.h" <BR>

\*-llib

\*

\* 类的详细说明

\*

\* @seesomething

\*/

#ifndef SORUTION\_PROJECT\_CLASSNAME\_H

#define SORUTION\_PROJECT\_CLASSNAME\_H

// 在这里包含系统头文件

//

// 在这里包含项目头文件

//

// 在这里包含局部头文件

//

// 在这里放置前置引用

//

class XX

{

public:

// 类的生命周期控制函数，如构造和析构，以及状态机

/\*\*

\*Default constructor.

\*/

XX(void);

/\*\*

\*Copy constructor.

\*

\*@param from The value to copy to this object.

\*/

XX(const XX& from);

/\*\*

\*Destructor.

\*/

virtual ~XX(void);

// 在这里放置类的运算操作符

/\*\*

\*Assignment operator.

\*

\*@param from THe value to assign to this object.

\*

\*@return A reference to this object.

\*/

XX& operator=(XX&from);

// 在这里放置类的操作

// 在这里放置属性存取

// 在这里放置类的状态查询

protected:

private:

};

// 内联方法定义

//

// 外部引用

//

#endif // SORUTION\_PROJECT\_CLASSNAME\_H

a. 定义的顺序是: public, protected, private

b. 要清楚public/protected/private都应该放置哪些东西

(2) 源文件格式（适用于C++）

#include "XX.h" // class implemented

/////////////// PUBLIC///////////////////////

//================= 构造函数 ====================

XX::XX()

{

}// XX

XX::XX(const XX&)

{

}// XX

XX::~XX()

{

}// ~XX

//=============== 操作符=========================

XX&

XX::operator=(XX&);

{

return \*this;

}// =

//==============类的操作 =====================

//==============属性存取 =====================

//==============状态查询 =====================

///////////// PROTECTED //////////////////

///////////// PRIVATE //////////////////

1. 源文件格式：

#include "XX.h" // class implemented

/////////////// PUBLIC///////////////////////

//================= 构造函数 ====================

XX::XX()

{

}// XX

XX::XX(const XX&)

{

}// XX

XX::~XX()

{

}// ~XX

//=============== 操作符=========================

XX&

XX::operator=(XX&);

{

return \*this;

}// =

//==============类的操作 =====================

//==============属性存取 =====================

//==============状态查询 =====================

///////////// PROTECTED //////////////////

///////////// PRIVATE //////////////////

保护头文件不被重复包含：（适用于C/C++）

2. 应使用宏定义来保护头文件不被重复包含；
3. 如果使用命名空间的时候，要把命名空间加到文件名前面。

（4）方法和函数布局：（适用于C/C++）

对于有较多参数的函数的写法：

如果参数较多，一行写不下，我们应该分成几行来写，并且每个参数都另起一行对齐 。

1. 缩进、制表符以及空格：（适用于C/C++）

（1）缩进的时候，每一层缩进3，4，或8个空格。（推荐使用4个空格）

（2）不要使用TAB，用空格，大多数编辑器可以用空格代替TAB。TAB应固定4个空格，因为大多数编辑器都是这么设置的。

（3）虽然没有规定缩进的层次，但是4至5层是合适的。如果缩进的层次太多，你可能需要考虑是否进行代码重构了。

1. 尽量保证一行不超过78个字（适用于C/C++）

有许多编辑器屏幕只有78个字母宽

1. 保证一行只写一条语句：（适用于C/C++）
2. 花括号的规则：
3. 位置：在关键字的下一行单独放置括号并且与关键字对齐
4. 使用：所有的 if, while 和 do 语句，要么用单行格式，要么使用花括号格式
5. 注释：在花括号结束的位置加上注释是一个好习惯。假如前后花括号距离很远，注释就能帮你理解它是如何对应的。
6. 大小： 一个语句块尽量不超过一个屏幕大小，这样，不要卷动屏幕就可以阅读代码。
7. 圆括号规则：
8. 圆括号与关键字之间应放一个空格；
9. 圆括号与函数名之间不要有空格；
10. Return 语句不要使用圆括号。
11. if else语句：

（1）布局

if (条件) // 注释

{

}

else if (条件) // 注释

{

}

else // 注释

{

}

1. 条件格式

总是把常量放在等号或不等于号的左边。

1. switch格式：
2. 直通的case语句，应该放置一条注释说明这个case语句是直通到下一个case语句的。
3. 总是要写default语句，不管是否是需要。
4. 在case中需要定义变量的时候，应把所有代码放在语句块中。
5. 运算符规则：
6. 一元操作符如（!、~ 等等）应贴近操作对象
7. 二元操作符如（+、\*、%、== 等等）应在前后留空格
8. ++ 和 -- 尽量使用前置运算。在C++中，不管 ++i 还是 i++，总是++i更容易生成优化代码
9. 变量声明：
10. 变量应该是随用随声明，不要集中在函数前（有些C语言不支持，则不在此要求之列）。特别是在for语句的循环变量，应只在for语句中定义
11. 声明语句块必须要对齐。
12. 文档及注释:
13. 文件或程序库的文档注释：

可以为整个文件编写文档。

1. 类文档注释：

在类定义前面应加上类说明文档。

1. 函数文档注释：
2. 所有的参数都应该有文档说明(param)，所有的返回代码都应该有文档说明(return)，所有的例外都应该有文档说明(exception)。可以使用(see)引用有关的开发资源
3. 注释属性，一些自动文档工具定义的属性可以包含在文档中。
4. include语句注释
5. 语句块注释：

语句块的注释可以用在语句块的开头和结束位置

1. 编码要求：
2. 不要忽略编译器的警告；
3. 应使用源代码管理器；
4. 固有的类方法成员：

默认构造函数、虚析构函数、拷贝构造函数、赋值操作。

1. 使用命名空间；
2. 初始化所有变量；
3. 使用函数短小精悍；
4. 对空语句进行注释；
5. 避免在语句中内含赋值；
6. 不要在头文件定义数据；
7. 不要直接使用数字；
8. 宏定义：
9. 如果可以，使用内联函数代替宏，使用内联函数可以达到相同的效果，而且更安全；
10. 在宏展开时，使用括号可以避免宏展开后产生的二义性；
11. 保证宏名称的唯一性。