**C++语言编码规范**

信1606 20163534吕新

# 命名约定

* 1.1类型名必须混合大小写且以大写开始。

Line, SavingsAccount

* 1.2变量名必须混合大小写且以小写开始。

line, savingsAccount

* 1.3常量（含枚举值）的命名必须全部为大写，以下划线分隔单词。

MAX\_ITERATIONS, COLOR\_RED, PI

int getMaxIterations() // NOT: MAX\_ITERATIONS = 25

{

return 25;

}

* 1.4方法与函数的命名必须是动作性的，采用大小写混合的写法以小写开始。

getName(), computeTotalWidth()

* 1.5名字空间的命名表应该全部小写。

analyzer, iomanager, mainwindow

* 1.6模板类型名应该用单个大写字母表示。

template<class T> ...

template<class C, class D> ...

* 1.7缩写与首写字母缩写作为名字不要全部大写。

exportHtmlSource(); // NOT: exportHTMLSource();

openDvdPlayer(); // NOT: openDVDPlayer();

* 1.8全局变量应该通过:: 操作符使用。

::mainWindow.open(), ::applicationContext.getName()

通常，避免使用全局变量。考虑用singleton对象代替。

* 1.9全局变量的命名加g\_前缀，类得成员变量加m\_前缀，静态变量加s\_前缀。

除名字与类型，变量的范围也是非常重要的。通过标识类范围的变量会很容易将它们与局部临时变量区分。

* 1.10普通变量应该与它们的类型使用同样的名字。

void setTopic (Topic topic) // NOT: void setTopic (Topic value)

// NOT: void setTopic (Topic aTopic)

// NOT: void setTopic (Topic x)

void connect (Database database) // NOT: void connect (Database db)

// NOT: void connect (Database oracleDB)

Name loginName;

* 1.11所有的名字应该使用英语的写法。

fileName; // NOT: filNavn

# 源代码文件

* 2.1C++头文件应该有.h扩展。源文件一般有.c++或.cpp扩展。

MyClass.cpp, MyClass.h

* 2.2类应该在头文件中声明，在源文件中定义，文件名与类名相匹配。

MyClass.h, MyClass.cc

* 2.3所有的定义应该在源文件中。

class MyClass

{

public:

int getValue () {return value\_;} // NO!

...

private:

int value\_;

}

**2.4文件的开头应标注题目、日期、作者**

# 语句

## 3.1类型

* 只针对一个文件的局部类型一般在相应文件内部声明。
* 类必须按照public，protected和private的顺序分几部分。每部分必须被显式的标识。没有用到的部分应该预留。
* 类型转换必须显式进行。不要依赖隐式的类型转换。

floatValue = static\_cast<float> (intValue); // YES!

floatValue = intValue; // NO!

## 3.2变量

* 变量应该在它们声明的地方初始化。

确保变量在任何时间是合法的。有时变量不能在声明时被初始化为合法的值，如：

int x, y, z;

getCenter (&x, &y, &z);

这种情况下，保持变量未被初始化胜过给一个假冒的值。

* 变量不要有双重含义。

通过确保所有概念被唯一表示增强可读性，同时减少负作用错误。

* 应该尽量少使用全局变量。
* 类的变量应该不要声明为public。

public变量与信息隐藏相矛盾，使用private变量与访问函数代替。一个例外是当类的本质是一个数据结构，没有行为（等效于C的结构）。在这种情况下，类的实例变量为public。

注意在C++中保留struct仅仅为兼容C，避免使用它们通过减少使用构造的数目来增加代码的可读性。使用类来代替。

* 同一类型的相关的变量一般在一条语句中声明。

不相关的变量不要在同一语句中声明。

float x, y, z;

float revenueJanuary, revenueFebruary, revenueMarch;

通过分组变量增强可读性。

* C++ 指针和引用的符号应该靠近变量名而非类型名。

float \*x; // NOT: float\* x;

int &y; // NOT: int& y;

* const 关键字应该在类型名之前。

void f1 (const Widget \*v) // NOT: void f1 (Widget const \*v)

* 布尔型变量和指针之外，应该不用隐式的0值测试。

if (nLines != 0) // NOT: if (nLines)

if (value != 0.0) // NOT: if (value)

* 变量应该在尽可能小的范围内声明。

保持在一个小的范围内操作对象，这样容易控制变量的影响与边界影响。

## 3.3循环

* 必须只有控制循环的语句包含在for( )的构造中。

sum = 0; // NOT: for (i = 0, sum = 0; i < 100; i++)

for (i = 0; i < 100; i++) // sum += value[i];

sum += value[i];

增加可维护性与可读性。清晰循环的控制与循环体。

* 循环变量应该在循环体前被初始化。

isDone = false; // NOT: bool isDone = false;

while (!isDone) { // :

: // while (!isDone) {

} // :

}

* 一般避免do-while循环。

do-while 循环因为条件在底部，没有while 循环与for 循环可读性好。读者必须扫描整个循环体才明白循环范围。

另外do-while 循环不是必须的，任何do-while循环均可以用while 循环或for循环重写。减少这种机制可以增强可读性。

* 在循环内应该避免使用break和continue。

这些构造可以与goto 比较，仅仅用在必须的地方。

## 3.4条件

* 必须避免非常复杂的条件表达式。通过引入临时的布尔变量代替。

if ((elementNo < 0) || (elementNo > maxElement)||

elementNo == lastElement) {

:

}

应该替代如下：

isFinished = (elementNo < 0) || (elementNo > maxElement);

isRepeatedEntry = elementNo == lastElement;

if (isFinished || isRepeatedEntry) {

:

}

* 对于if语句，正常的分支应该放在if部分，异常部分放在else部分。

isError = readFile (fileName);

if (!isError) {

:

}

else {

:

}

* 条件应该放在单独的一行。

if (isDone) // NOT: if (isDone) doCleanup();

doCleanup();

* 必须避免在条件部分包含执行语句。

// Bad!

if (!(fileHandle = open (fileName, "w"))) {

:

}

// Better!

fileHandle = open (fileName, "w");

if (!fileHandle) {

:

}

## 

## 3.5杂项

* 应该在代码中避免出现神秘的数。除0和1以外的任何数均为神秘数，应该用命名常量代替。
* 建议通过方法来访问常量。
* 函数必须总是显式列出返回值。

int getValue() // NOT: getValue()

{

:

}

* 检查函数所有参数输入的有效性；并对所调用函数的错误返回码要仔细、全面地处理。
* 这样能保证函数的稳定性，健壮性。
* 应该不用goto。

goto 语句与结构化代码相矛盾。仅仅在证明必须使用时才使用。

* 在不包含注释和空行的情况下，函数的规模尽量限制在200行以内。
* 减少函数本身或函数间的递归调用；
* 应该用”0”替代”NULL”。

NULL是标准C库的一部分，但在C++中是过时的。

# 布局与注释

## 4.1布局

* 基本的缩排为4。

for (i = 0; i < nElements; i++)

a[i] = 0;

* 块的缩排应该使用下面的例1(java默认编排方式)或例2的布局，但不可以使用例3。函数和类的块必须使用例2的布局。建议使用例2。

while (!done) {

doSomething();

done = moreToDo();

}

while (!done)

{

doSomething();

done = moreToDo();

}

while (!done)

{

doSomething();

done = moreToDo();

}

* 函数或过程的开始、结构的定义及循环，判断等语句中的代码都要采用缩进风格。
* 函数的申明应该使用如下的形式：

void someMethod()

{

...

}

* 各类if-else语句应该使用如下的形式：

if (condition)

{

statements;

}

if (condition)

{

statements;

}

else

{

statements;

}

if (condition)

{

statements;

}

else if (condition)

{

statements;

}

else

{

statements;

}

下面部分遵循块的通用规则，但是else语句与前一个if或else语句的右括号在同一行：

if (condition)

{

statements;

}

else

{

statements;

}

* for 语句应该使用如下的形式：

for (initialization; condition; update)

{

statements;

}

* 空的for语句应该使用如下的形式：

for (initialization; condition; update)

;

强调for语句是空的，显式表明意图。然而，空循环应避免。

* while 语句应该使用如下的形式：

while (condition)

{

statements;

}

* do-while 语句应该使用如下的形式：

do

{

statements;

} while (condition);

* switch语句应该使用如下的形式：

switch (condition)

{

case ABC :

statements;

// Fallthrough

case DEF :

statements;

break;

case XYZ :

statements;

break;

default :

statements;

break;

}

每个case关键字的缩进与switch语句相关作为一个整体，这样突出显示switch语句。注意在每个：之前使用格外的空白符。当对应的case语句没有break语句时，需要显式的注释说明，防止因为无意漏加。

* try-catch语句应该使用如下的形式：

try

{

statements;

}

catch (Exception exception)

{

statements;

}

部分遵循通用块的规则。对于if-else语句使用的闭括号规则同样适用于try-catch语句。

* 单一的if-else，for或while语句一般可以不用大扩号。

if (condition)

statement;

while (condition)

statement;

for (initialization; condition; update)

statement;

通用的推荐(包括Sun Java推荐在内) 这种情况总是使用大括号。但是，大括号作为语言机制是用于分组多条语句的，对单一语句使用是多余的。

## 4.2空白符

* 一般操作的前后加一个空白符。
* C++的保留字后需跟随一个空白符。
* 逗号后应该跟随一个空白符。
* 冒号的前后加一个空白符
* 在for语句中的分号后加一个空白符。

a = (b + c) \* d; // NOT: a=(b+c)\*d

while (true) { // NOT: while(true) ...

doSomething (a, b, c, d); // NOT: doSomething (a,b,c,d);

case 100 : // NOT: case 100:

for (i = 0; i < 10; i++) { // NOT: for (i=0;i<10;i++){

* 方法名后如有其他名字，应该紧随一个空白符。

doSomething (currentFile); // NOT: doSomething(currentFile);

突出显示个体的名字，增强可读性。

一种替代的方法是在括号后跟随一个空白符，如doSomething( currentFile )；这突出了名字的意图，但是，后面的括号是多余的，如去掉，空白语句不对称(doSomething( currentFile);)。

* 相对独立的程序块之间，变量说明之后必须加空白行分隔。
* 一个块内的逻辑单元应该使用1个空白行分隔。

在块内的逻辑单元间使用空白将增强可读性。

* 方法应该用3个空白行分隔。
* 变量在声明时应该左对齐。

AsciiFile \*file;

int nPoints;

float x, y;

增强可读性。通过对齐很明显认出变量与类型。

* 通过对齐增强可读性。

if (a == lowValue)

compueSomething();

else if (a == mediumValue)

computeSomethingElse();

else if (a == highValue)

computeSomethingElseYet();

value = (potential \* oilDensity) / constant1 +

(depth \* waterDensity) / constant2 +

(zCoordinateValue \* gasDensity) / constant3;

## 4.3注释

* 不好的代码不应该试图通过注释解决，应该重写代码！

一般的，代码应该通过适当的名字选择与显式的逻辑结构做到自身文档化，应该少使用注释来弥补差的代码。

* 源文件头部应进行注释，列出：版权说明、版本号、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。
* 对所有需要注释的使用//注释符，包括多行注释。

// Comment spanning

// more than one line.

既然C注释不支持多层，使用//注释确保当为了调试等原因时总能使用/\* \*/在文件中注释一整块。

在“//“与实际注释间适用一个空白符，注释以大写字母开始以句号结束。

* 注释的位置应该与代码相关。注释的缩进应该和代码同步。

while (true) // NOT: while (true) {

{

// Do something // // Do something

something(); // something();

} // }

这样避免注释破坏了程序的逻辑结构。