Google C++ Style Guide Reform

郑泉

April 1, 2014

目录

Chapter 1

命名规则

在尽可能遵守本规则的情况下和上下文习惯保持一致。

1.1 通用规则

函数名，变量名，文件名应该使用详细描述，而不用缩写。保证描述明确，避免歧义。易于读者理解。倾向于用长名称保证理解。

1.2 文件名称

文件名使用小写，单词间用下划线（_）或者波折号（-）分隔。

my_useful_class.cpp

my-useful-class.cpp

1.3 类型名

每一个单词首字母大写，没有下划线。

```
// classes and structs
class UrlTable { ...
class UrlTableTester { ...
struct UrlTableProperties { ...
// typedefs
typedef hash_map<UrlTableProperties *, string> PropertiesMap;
// enums
enum UrlTableErrors { ...
```

1.4 变量名

小写，单词间用下划线（_）分隔或者不分隔。特别的成员变量以下划线（_）结尾。(只对 class 中成员变量如此， struct 中不必。)

对于全局变量加 g_ 前缀。

```
string table_name;  // OK - uses underscore.
string tablename;  // OK - all lowercase.
class UrlTable {
  string name_;
  int num_entries_;
}
struct UrlTableProperties {
  string name;
  int num_entries;
}
```

1.5 常量名

对于 const 修饰的常量名前加 k 前缀。

```
const␣int␣kDaysInAWeek␣=␣7;
```

1.6 函数名

每个单词首字母大写，对于可能失败的函数加后缀 OrDie 。

```
AddTableEntry()
DeleteUrl ()
OpenFileOrDie()
```

对于访问和设置变量的函数，用变量名。

```
class␣MyClass␣{
␣public:
␣␣ ...
␣␣int␣num_entries()␣const␣{␣return␣num_entries_;␣}
␣␣void␣set_num_entries(int␣num_entries)␣{␣num_entries_␣=␣num_entries;␣}

␣private:
␣␣int␣num_entries_;
};
```

对于非常小内联函数，名称可以小写。

1.7 名字空间名

小写基于项目和路径名称。(近似于 java 包名)

```
google_awesome_project
```

1.8 枚举名

类似于常量名和宏名。推荐常量名。

```
enum␣UrlTableErrors␣{
␣␣kOK␣=␣0,
␣␣kErrorOutOfMemory,
␣␣kErrorMalformedInput,
};
```

1.9 宏名

宏定义大写，下划线分隔单词。

```
#define␣ROUND(x)␣...
#define␣PI_ROUNDED␣3.0
```

Chapter 2

格式

2.1 细节

推荐使用 80 个字符一行，同时对于非 ASCII 码，建议使用 UTF-8 编码。

就缩进而已，推荐使用两个空格缩进，避免使用 TAB 键。

2.2 函数声明、定义和调用

1. 如果返回值类型和函数名太长，参数列表下移一行
2. 下移后，返回值类型和函数名之间不缩进
3. **(**与函数名同行，且之间有一个空格
4. 不要在参数名之间加空格
5. **{**与最后一个参数同行
6. **}**在最后一行，同样不缩进（看上下文）或者 **{}**同行
7. **) {**
8. 所有参数都要有名字，无名参数加注释
9. 参数尽可能对齐
10. 相对于返回值类型，内层缩进 2 空格，而分行参数缩进 4 空格

```
ReturnType␣LongClassName::ReallyReallyReallyLongFunctionName(
␣␣␣␣Type␣par_name1,␣␣//␣4␣space␣indent
␣␣␣␣Type␣par_name2,
␣␣␣␣Type␣par_name3)␣{
␣␣DoSomething();␣␣//␣2␣space␣indent
}
//␣Always␣have␣named␣parameters␣in␣interfaces.
class␣Shape␣{
␣public:
␣␣virtual␣void␣Rotate(double␣radians)␣=␣0;
}

//␣Always␣have␣named␣parameters␣in␣the␣declaration.
class␣Circle␣:␣public␣Shape␣{
␣public:
␣␣virtual␣void␣Rotate(double␣radians);
}

```

```
// Comment out unused named parameters in definitions.
void Circle :: Rotate(double /*radians*/) {}
```

2.3 条件语句

```
if ( condition ) {  // no spaces inside parentheses
  ... // 2 space indent.
} else if  (...) {  // The else goes on the same line as the closing brace.
  ...
} else {
  ...
}
```

或者，主要是格式统一。

```
if (( condition ) ) {  // spaces inside parentheses - rare
  ... // 2 space indent.
} else {  // The else goes on the same line as the closing brace.
  ...
}
```

当没有 else 时，同行接执行语句是可以的。

```
if (x == kFoo) return new Foo();
if (x == kBar) return new Bar();
```

但存在配对时，最好如下

```
// Curly braces around both IF and ELSE required because
// one of the clauses used braces.
if ( condition ) {
  foo;
} else {
  bar;
}
```

2.4 循环和 switch

空循环体加 {}或者 continue 。

为 swith 提供 default ， 如果不执行加入 assert(false); 。

```
switch (var) {
  case 0: {  // 2 space indent
    ... // 4 space indent
    break;
  }
  case 1: {
    ...
    break;
  }
```

```
  default: {
    assert( false );
  }
}
while ( condition ) {
  // Repeat test until it returns false .
}
for ( int i = 0; i < kSomeNumber; ++i) {}  // Good - empty body.
while ( condition ) continue;  // Good - continue indicates no logic.
```

2.5 指针和引用

* 和 & 不能两边都是空格。

```
x = *p;
p = &x;
x = r.y;
x = r->y;
// These are fine, space preceding.
char *c;
const string &str;

// These are fine, space following.
char* c;     // but remember to do "char* c, *d, *e, ...;"!
const string & str;
```

2.6 逻辑表达式

当逻辑 && 和 || 超长时，参照参数格式缩进。

```
if ( this_one_thing > this_other_thing &&
    a_third_thing == a_fourth_thing &&
    yet_another && last_one) {
  ...
}
```

2.7 返回值

返回值不乱加括号，除非返回值是一个表达式。

```
return result ;                  // No parentheses in the simple case.
return (some_long_condition &&   // Parentheses ok to make a complex
        another_condition);      //      expression more readable.
```

2.8 预处理

在预处理中，# 号必须在行首。

```
// Good - directives at beginning of line
  if ( lopsided_score ) {
#if DISASTER_PENDING       // Correct -- Starts at beginning of line
```

```
␣␣␣␣DropEverything();
#␣if␣NOTIFY␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣OK␣but␣not␣required␣--␣Spaces␣after␣#
␣␣␣␣NotifyClient() ;
#␣endif
#endif
␣␣␣␣BackToNormal();
␣␣}
```

2.9 类格式

public, protected, private 缩进一个空格，后面的成员也只缩进一个空格。且不同组的函数用空格分块。

```
class␣MyClass␣:␣public␣OtherClass␣{
␣public:␣␣␣␣␣␣//␣Note␣the␣1␣space␣indent!
␣␣MyClass();␣␣//␣Regular␣2␣space␣indent.
␣␣explicit␣MyClass(int␣var);
␣␣~MyClass()␣{}

␣␣void␣SomeFunction();
␣␣void␣SomeFunctionThatDoesNothing()␣{
␣␣}

␣␣void␣set_some_var(int␣var)␣{␣some_var_␣=␣var;␣}
␣␣int␣some_var()␣const␣{␣return␣some_var_;␣}

␣private:
␣␣bool␣SomeInternalFunction();

␣␣int␣some_var_;
␣␣int␣some_other_var_;
␣␣DISALLOW_COPY_AND_ASSIGN(MyClass);
};
```

构造函数的初始化列表缩进与函数参数基本一样。

```
//␣When␣it␣all␣fits␣on␣one␣line:
MyClass::MyClass(int␣var)␣:␣some_var_(var),␣some_other_var_(var␣+␣1)␣{}
//␣When␣it␣requires␣multiple␣lines ,␣indent␣4␣spaces,␣putting␣the␣colon␣on
//␣the␣first␣␣initializer␣␣line :
MyClass::MyClass(int␣var)
␣␣␣␣:␣some_var_(var),␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣4␣space␣indent
␣␣␣␣␣␣some_other_var_(var␣+␣1)␣{␣␣//␣lined␣up
␣␣ ...
␣␣DoSomething();
␣␣ ...
}
```

名字空间内的不需要缩进。

```
namespace␣{
␣
void␣foo()␣{␣␣//␣Correct.␣␣No␣extra␣indentation␣within␣namespace.
␣␣ ...
}
␣
}␣␣//␣namespace
```

2.10 注释

在.h.cpp 的开头应有一段格式统一的说明，内容包括:

- 文件名 (FileName)
- 创建人 (Creator)
- 文件创建时间 (Date)
- 简短说明文件功能、用途 (Comment)

除非函数极其简单，否则对函数应有注释说明。内容包括：功能、入口出口参数，必要时还可有备注或补充说明。

- 对于 if、while、do 等其大括号内嵌代码块比较长 (需拖动滚动条来看) 或者多层嵌套时，在结尾的} 后要加上其所对应的语句的注释说明
- 函数入口参数有缺省值时，应注释说明