第 17 章 风　格 17

计算机程序并不仅仅是写来供计算机处理的，它也会用来供其他的程序员阅读。为了提高程序的可读性（和可维护性及减少程序的错误），除了让编译器接受之外，还得有些额外的考量。风格上的考虑是计算机程序设计中最不具有客观性的方面：关于代码风格的观点，就像门派之争一样，可以无休无止地辩论下去。良好的风格是个有价值的目标，这点通常也被广泛认可。但要严格规定它却也不能。无论如何，缺乏良好风格的客观标准乃至业界的共识并不意味着程序员就可以放弃对程序可读性的关注和努力。

17.1

问：什么是C最好的代码布局风格？

答：Kernighan和Ritchie提供了最常被复用的范例，但同时他们并不要求大家沿用他们的风格。

大括号的位置并不重要，尽管人们对此怀有执着的热情。我们在几种流行的风格中选了一种。选一个适合你的风格，然后坚持使用这一风格。

保持布局风格跟自己、其他人及通用源码的一致性比使之“完美”更重要。如果你的编码环境（本地习惯或公司政策）没有建议一个风格，而你也不想发明自己的风格，当然可以沿用K&R的风格。

几种流行的风格各有优缺点。将左括号独立放在一行会浪费垂直空间；把它跟下一行结合会难以编辑；跟上一行结合又会导致它和右括号不能对齐，从而更难看到。

每级缩进8列最常见，但常常又会让你太接近右边界（这可能也暗示你该分解一下你的函数了）而很不舒服。如果缩进一个tab但把tab值设定为8以外的值，你就得要求其他人用跟你一样的软件设置来阅读你的代码。参见文献[23]。

“好风格”的品质并不简单，它包含的内容远远不止代码的布局细节。不要把时间都花在格式上而忽略了更实质性的代码本身的质量。

参见问题17.2。

参考资料：[18, Sec. 1.2 p. 10]
[19, Sec. 1.2 p. 10]

17.2

问：如何在源文件中合理分配函数？

答：通常，相关的函数放在同一个文件中。有时候（例如开发库的时候），一个源文件（自然也就是一个目标文件）放一个函数比较合适。有时候，尤其是对某些程序员，太多的源文件可能会很麻烦，将多数以至所有的程序都放入少数几个大的源文件中也很诱人，甚至也是合适的。希望用`static`关键字限制某些函数或全局变量的作用域时，源文件的分配就有更多限制了：静态函数和变量以及共享它们的函数都必须在同一个源文件中。

换言之，这里有些权衡，因此很难给出一般的规则。参见问题1.7、1.9、10.6和10.7。

17.3

问：用`if(!strcmp(s1, s2))`比较两个字符串是否相等是个好风格吗？

答：这并不是个很好的风格，尽管这是个流行的习惯用法。如果两个字符串相等，这个测试返回真，但`!`（“非”）的使用容易引起误会，以为测试不相等情况。

另一个选择是定义一个宏：

```
#define Streq(s1, s2) (strcmp((s1), (s2)) == 0)
```

然后这样使用：

```
if(Streq(s1, s2))
```

另一种选择（可以防止宏的滥用，参见问题10.2）是定义

```
#define StrRel(s1, op, s2) (strcmp(s1, s2) op 0)
```

然后你可以这样使用：

```
if(StrRel(s1, ==, s2)) ...
if(StrRel(s1, !=, s2)) ...
if(StrRel(s1, >=, s2)) ...
```

参见问题17.10。

17.4

问：为什么有的人用`if(0 == x)`而不是`if(x == 0)`？

答：这是用来防止一个常见错误的小技巧：

```
if(x = 0)
```

如果你养成了把常量放在==前面的习惯，那么当你意外地把代码写成了：

```
if(0 = x)
```

编译器就会报错。显然，一些人会觉得记住倒转测试比记住输入两个=号容易。（的确，就算是经验老道的程序员有时也会错把==写成=。）当然这个技巧只对和常量比较的情况有用。

另一方面，有的人又觉得这样倒转的测试既难看又影响注意力，因而提出应该让编译器对`if(x = 0)`报警。（实际上，很多编译器的确对条件式中的赋值报警，当然如果你真的需要，你总是可以写`if((x = expression))`或`if((x = expression)!= 0)`。

参考资料：[11, Sec. 7.6.5 pp. 209-210]

17.5

问：为什么有些代码在每次调用`printf`前增加了类型转换`(void)`？

答：`printf`确实有返回值（输出的字符个数或错误码），但几乎没有谁会去检验每次调用的返回值。由于有些编译器和`lint`对于被丢弃的返回值会报警告，显式地用`(void)`作类型转换相当于说：“我决定忽略这次调用的返回值，请继续对于其他（也许不慎）忽略返回值的情况提出警告”。通常，`(void)`类型转换也用于`strcpy`和`strcat`的调用，它们的返回值从没有什么惊人之处。

参考资料：[19, Sec. A6.7 p. 199]
[14, Sec. 3.3.4]
[11, Sec. 6.2.9 p. 172, Sec. 7.13 pp. 229-230]

17.6

问：既然`NULL`和`0`都是空指针常量，我到底该用哪一个？

答：参见问题5.9。

17.7

问：是该用`TRUE`和`FALSE`这样的符号名称还是直接用`1`和`0`来作布尔常量？

答：参见问题9.4。

17.8

问：什么是“匈牙利表示法”（Hungarian Notation）？是否值得一试？

答：匈牙利表示法是一种命名约定，由Charles Simonyi发明。他把变量的类型（或者它的预期使用）等信息编码在变量名中。在某些圈子里，它被极度热爱，而在另一些地方，它又受到严厉的批评。它的主要优势在于变量名就说明了它的类型或者用法。它的主要缺点在于类型信息并不值得放在变量名中。

参考资料：[32]

17.9

问：哪里可以找到“Indian Hill Style Guide”及其他编码标准？

答：各种文档在匿名ftp都可以得到：

地 址	文档及目录
ftp.cs.washington.edu	pub/cstyle.tar.Z （更新的Indian Hill Guide）
ftp.cs.toronto.edu	doc/programming （包括Henry Spencer的《C程序员的十诫》（“10 Commandments for C Programmers”））
ftp.cs.umd.edu	pub/style-guide

也许你会对这些书感兴趣：*The Elements of Programming Style* [17]、*Plum Hall Programming Guidelines* [28]和*C Style：Standards and Guidelines*[33]。

参见问题18.8。

17.10

问：有人说goto是邪恶的，永远都不该用它。这是否太极端了？

答：程序设计风格就像写作风格一样，是某种程度的艺术，不能被僵化的教条所束缚。虽然风格的探讨经常都是围绕着这些规则。

对于goto语句，很早以前人们就注意到随意地使用goto会很快地导致难以维护的混乱代码（spaghetti code）。然而，不经思考就简单地禁止goto的使用并不能立即得到优美的程序。一个无规划的程序员也许使用奇怪的嵌套循环和布尔变量来取代goto，一样能构造出复杂难懂的代码。

通常，把这些程序设计风格的评论或者“规则”（结构化编程好、goto不好、函数应该在一页以内等）当作指导准则比当作规则要更好。如果程序员理解这些指导准则所要实现的目标，其效果就会更好。盲目地回避某种构造或者死套规则而不融会贯通，最终还会导致这些规则试图避免的问题。

此外，许多程序设计风格的观点只是“观点”而已。某些观点看似经过了充分的讨论、得到了强烈的支持，但跟它相对的观点可能也得到了同样多的支持。通常卷入“风格战争”是毫无意义的。对于某些问题（像问题5.3、5.9、9.2和10.7）争辩的双方是不可能同意、认同对方的不同或者是停止争论的。

最后，正如William Strunk所写的（引自Strunk和White的经典著作*Elements of Styles*的序）：人们早就发现最好的作家有时候对花言巧语的规则置之不顾。然而，当他们不守规则的时候，读者往往会在字里行间发现以违规为代价得到补偿价值。除非他确信能做好，否则最好还是遵守规矩。

参考资料：[7]
[20]

17.11

问：人们总是说良好的风格很重要，但当他们使用良好的风格写出清晰易读的程序后，又发现程序的效率似乎降低了。既然效率那么重要，是否可以为了效率牺牲一些风格和可读性呢？

答：的确，效率低下的程序是个问题，但很多程序员有时对效率的盲目追求也是个问题。麻烦晦涩的编程技巧不仅降低可读性和可维护性，同时跟选择合适的设计或算法相比，也可能导致更微不足道的长期效率提升。小心对待，设计出既清晰又高效的代码也是可能的。

参见问题20.14。