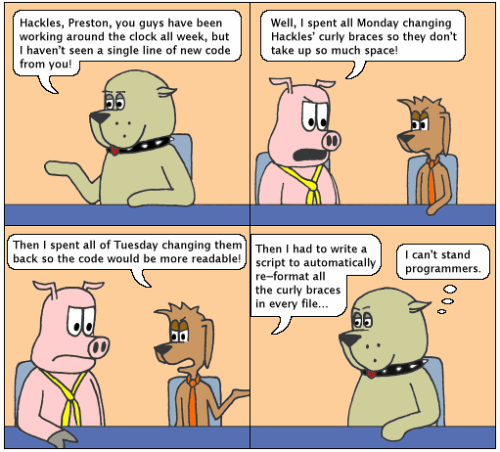
Coding Style

写程序的人都或多或少会有这种感觉，别人的code看起来总不是那么地顺眼，阅读自己的code才是像阅读好书一样如行云流水般顺畅。其实写code如写书，不仅写给自己看，同时也写给别人看。开发软体也往往有如打造一件工艺品，投入其中的巧妙心思及用心，会影响到最后呈现出来的结果。所以，写程式本身可以是一种艺术，而不仅仅是一件耗费劳力的枯燥工作。这也是为什么Knuth要把他的巨著取名为The Art of Computer Programming，他认为打造软体是困难的，是一种复杂度以及最后呈现结果足够作为一件艺术品的一种过程。当然要迈入如创造艺术品般地去打造软体这样的一个境界，需要十足的编码功力和对程序的深入理解。今天我们只针对写代码如写书来谈谈Coding Style。

既然写code如写书，那么最重要的就莫过于条理清晰，脉络分明的内容了，于是我们就需要一套令人容易了解的呈现手法来写作了，一方面让自己好读，一方面也让读者好读。于是coding style这种东西就出现了。没有了Coding Style就会发生如下情况：



要做好一件事情，我们首先要认清一个事情的重要性，那么Coding Style为什么值得我们在编码过程中付诸一定的关注度，这就要从我们写代码给谁看这一点说起。我们写代码给谁看呢？给老板看？老板不稀罕看。给用户看？用户也不稀罕看。一般来说我们写代码给三个人看。

1. 给编译器看：每个程序员都会供佛一样供编译器，确保自己写出来的东西不会让编译器吐出一堆红叉叉，因此我们会乖乖地遵循着编程语言的语法，好好地写给编译器看，小心翼翼地不要犯错。但编译器不会计较你写code的style，只在意你的语言语法。
2. 给自己看：不幸的是，很多人从头到尾都没想到，写code也要写给自己看，而且不仅仅是让自己现在看得懂，也要让自己以后看得懂，很多程序员年轻时不懂事，在程式里用了高超的语法和神奇的算法，写了执行速度快人一等的程序，却就是忘了写注解，一段时间后翻开一看，虽然大概知道以前用的技巧和语法，却要重新咀嚼一番才能完全理解。
3. 给teammates看：大作业deadline快到了，12点就要交了，还要写文档啊，赶紧赶代码呀，先交上去吧。现在这种情况下，能有东西交差就不错了，还管什么style和注解啊。ok，连夜赶工终于写完了，但是下一个project又接踵而至，哪还有时间回头去把code好好重新整理一番。下次作业迭代时，各自写的代码要整合，傻眼了，自己代码都不清楚干嘛的，别人的更不清楚。

于是在现在的学习和以后的工作中，我们往往就会看到一大堆没有注解和coding style混乱的代码，这时候大家就会发自内心的说出那三个字“求文档”，然后鄙视前人，“这代码能看？”。但自己却也不会去注意这些，然后过一段时间后另外一批人再咒骂我们，生生不息，这是一种循环。根据业界的统计，一个软体的生命周期之中，有80%是花在维护上面，让他人容易读懂和维护你的代码，对于软体的成功和演进至关重要。

在业界一些比较大型的软体公​​司，自然会有一套方法来避免这种现象，所以coding standard应运而生，它规定了团队里写代码的所需遵循的规范，保证了风格的统一。现在大家都是一个正在学变成，写写学校作业和project的学生，培养好的coding style对我们来说同样重要。因此下面我列了一些业界的自订标准和经典的coding style，以下简单列出一些给大家参考：

1. Coding Conventions for the Java Programming Language（见附件）
2. [Google C++ code style](http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml)

<http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

1. Linux Coding Style

<https://www.kernel.org/doc/Documentation/CodingStyle>

1. Hungarian Notation（匈牙利命名法）

<http://msdn.microsoft.com/en-us/library/aa260976(v=vs.60).aspx>

Coding Style没有绝对的好与坏。重要的是一但你选定自己遵循的规则后，之后最重要的就是维持这个style的一致性，不要变来变去。