【京】新登字 158 号

内容简介

本书是面向程序员的“天龙八部”，真正的编程高手是在千锤百炼之后诞生的。其中的酸甜苦辣均凝结在本书的字里行间，尤为珍贵的是文中的“秘笈”将会是无名小卒脱胎换骨的利刃，希望本书能让程序员体味到一个编程的“天高云淡”的最高境界。

书　　名：凌波微步　软件开发警戒案例集
作　　者：王咏刚　周虹
责任编辑：丁岭
出 版 者：清华大学出版社（北京清华大学学研大厦，邮编：100084）
http：//www.tup.tsinghua.edu.cn
印 刷 者：世界知识印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787 × 960 1/16　**印张**：26.25　**字数**：476 千字
版　　次：2002 年 11 月第 1 版　　2002 年 11 月第 1 次印刷
书　　号：ISBN 7-900643-68-0
印　　数：0001～4000
定　　价：42.00 元

问题引入

注释，当然是注释！任何一个有经验、有责任感的项目经理都会反复向我这样的新手强调说：“注释，悦是它的精髓！记住，写程序的同时要写注释，写注释的同时要写得详尽，写得详尽的同时要写得十分详尽！小伙子，照我说的做，你会成为编程高手的。”

好吧好吧，我懂得注释的重要性了。我要给大家看一个功能完整、注释详尽的 悦语言程序——这可是本书的第一个案例噢（也许是最简单的一个吧）。这个 悦程序要做的事儿是这样的：文件 精灵图.悦 中保存了一幅高 源图像素、宽 源图像素的彩色图像，每像素以 圆原位 砸阴彩色格式（郧 月 砸字节顺序）存储，程序自动将该图像转换为 圆级灰度图像，格式是每字节（原位）表示一个像素，并保存在文件 精灵图灰.悦 中。

程序清单 员源摇悦是它的精髓

```
#include <stdio.h>

/* 函数 main，返回 0 表示操作成功，返回 -1 表示失败 */
int main(int argc, char* argv[])
{
    FILE *fin,*fout; /* fin 是输入文件，fout 是输出文件 */
    int i,j,b,g,r,y; /* 计数器和转换用的其他临时变量 */

    /* 以二进制只读方式打开文件并判断打开是否成功 */
    if ((fin=fopen("cat.pic","rb"))==NULL)
    {
        puts("打开文件 cat.pic 时错误");
        /* 如果打开失败则显示错误信息 */
        return -1;
    }
    /* 将文件指针移动到文件尾 */
    fseek(fin,0,SEEK_END);
    /* 获得文件指针的位置并判断该值是否等于 400*400*3 */
    if (ftell(fin)!=400*400*3)
    {
        puts("输入文件 cat.pic 不符合格式要求");
        /* 如果不等，显示错误信息 */
        fclose(fin);
        return -1;
    }
    fseek(fin,0,SEEK_SET); /* 将文件指针移动到文件开头 */
    /* 以二进制写方式打开文件并判断是否成功 */
    if ((fout=fopen("cat2.pic","wb"))==NULL)
    {
        puts("打开文件 cat2.pic 时错误");
        /* 如果打开失败则显示错误信息 */
        fclose(fin);
```

```
        return -1;
    }
    /* 从图像的第 1 行到第 400 行循环 */
    for(i=0;i<400;i++)
        /* 从每行的第 1 列到第 400 列循环 */
        for(j=0;j<400;j++)
        {
            b=fgetc(fin); /* 从输入文件中读入每像素的 RGB 值 */
            g=fgetc(fin);
            r=fgetc(fin);
            /* 按照公式 Y=0.299R+0.587G+0.114B 计算灰度值 */
            y=(299*r+587*g+114*b)/1000;
            fputc(y, fout);
            /* 将计算出来的灰度值写到输出文件中去 */
        }
    fclose(fin); /* 关闭输入文件 */
    fclose(fout); /* 关闭输出文件 */
    return 0; /* 返回 0 表示正确处理完毕 */
}
```

程序看完了吗？别找了别找了，这个程序没有功能上的问题。彩色变黑白，再简单不过了，上面的代码肯定能给出再正确不过的 [illegible]。

大家需要仔细找仔细看的是程序里的注释，你不觉得程序里的注释有问题吗？有什么问题？先不要问，再看一遍程序，你肯定已经找到了。不过在讨论这个案例以前，让我们先谈一些无关紧要的事情。

闲聊一些题外话

一个公式：把彩色图像转换成灰度图像和使用黑白电视机播放彩色电视信号的原理是一样的。计算机里彩色图像大多被存储成 [illegible] 格式，也就是将具体的颜色分成红、绿、蓝三个颜色分量。数字图像处理领域将这一表示方式称作 [illegible] 色彩空间。相应的，还有其他各种色彩空间，如 [illegible] 等等。灰度图像的灰度值（或称明度值）就对应于 [illegible] 色彩空间中的 [illegible] 分量值，也就是说，将彩色图像转换成灰度图像只需要作 [illegible] 到 [illegible] 的色彩空间变换，并取 [illegible] 分量值作为灰度值就可以了。[illegible] 三个分量与 [illegible] 分量的换算关系是：

[illegible]

上面的例子程序里就是使用这样的公式完成图像转换的。

一个小故事：两千多年前的春秋时候，齐鲁大学校长孔子把得意门生颜回叫到跟前面授机宜。颜回听完老师的教导，抬头看看天，俯身看看地，脑筋急转弯，猛然发现自己吹捧老师的时机到了，于是在脸上堆出万分景仰的表情，对孔老师唱道：“大河向东流，天上的星星参北斗。

老师就是北斗星啊，老师走来我也走。老师跑步我跟着溜，老师骑马我也把马鞭儿抖。老师您一骑绝尘、凤舞九天，徒弟我虽然还没把那梯云纵的轻功学到手，也要在后面为您喝彩叫好直喊到江水也倒流！”——顺便说一句，这个故事便是成语“亦步亦趋”的起源（见《庄子·田子方》）。

又一个小故事：清朝康熙年间，有两个大诗人，一个叫做邵长蘅，一个叫做宋荦。乾隆时候又出了一个大诗人叫洪亮吉。这洪亮吉与邵长蘅同乡，却偏偏在作诗的意见上与邵、宋二人相左。俗话说“文人相轻”，洪亮吉后来就在他的一部诗评中指摘邵长蘅说：“我那个同乡邵长蘅开始写诗的时候，只知道模仿盛唐的诗风，模仿得连自己都不知道自己是姓李还是姓杜了，哪里还谈得上‘独到’二字！后来，邵长蘅到了宋荦的府中，又开始模仿宋荦的风格，简直是亦步亦趋，丢足了我们诗人的面子。”——再多说一句，这是古人把成语“亦步亦趋”运用得最为成功的一个例子（见清·洪亮吉《北江诗话》卷五）。

猿遥案例分析

好，现在让我们正式开始讨论案例。代码 悦[illegible]中的注释不可谓不多、不可谓不详，几乎到了一行程序一行注释的“最高境界”了。但仔细读一读就会发现，这些注释的效率实在不高，价值实在不大，与其说是代码的“注释”，倒不如说是对代码的“白话文翻译”。什么叫“白话文翻译”呢？我们来看：

```
/* 以二进制只读方式打开文件并判断打开是否成功 */
if ((fin = fopen("cat.pic","rb")) == NULL)
```

这样的注释分明就是在说：“[illegible]() 函数是用来打开文件的，[illegible]参数是指只读、二进制方式，而非可写、文本方式，条件表达式判断的是打开文件是否成功……”絮絮叨叨，没完没了，好像全世界就只有他一个人会写 悦语言程序，会用 悦程序打开文件。这样的注释如果放在《晕天学会 悦语言编程》之类的畅销书里，就再好不过了。

还有更绝的：

```
/* 按照公式 Y = 0.299R + 0.587G + 0.114B 计算灰度值 */
y = (299 * r + 587 * g + 114 * b)/1000;
```

我看了表达式明白这是在计算灰度值，可就是不知道为什么这样算，不知道这样算是基于什么样的定理或公式；于是我去看注释，因为我相信

注释里会有更多、更有价值的信息。可不看则已，等到我看见这行注释实际上就是把代码中的表达式利用分配律换了种写法，这时候，我就好像听到有人对我说：“你知道吗？一加二可以写成一加一再加一的。哦，什么，你早就知道了？真不可思议，你简直太聪明了！”

这段程序的作者如果被洪亮吉碰到，肯定又会被骂作“亦步亦趋”的典型。这里的“亦步亦趋”一是指作者只知道听老师的话把注释写得十分详尽，却忘了注释的真正意义所在；二是指这段程序中的每条注释都如影随形地跟在被注的代码旁边，一个说“风吹水面层层浪”，另一个讲“浪起水面只因风”，简直没有一点“独到”精神。

实际上，程序中“注释”的最重要的功效在于传承。传承一般有两种情况。第一，写代码的人写完这段代码之后会去写下一段代码、下下一段代码，直写到东西莫辨、朝午不明，也许过了一年半载以后，客户提出新的改动需求时，他才会回过头来看原来的代码。为了在未来的某个时候更快地接上当时编码的思路，更好地理解已尘封数月的程序，当然要未雨绸缪，事先就写好注释，否则不被老板痛骂才怪。第二，没有人愿意永远维护自己写过的代码，也没有老板可以保证手下的编程高手不会另择高枝，所以心地善良的程序员们总会写好注释（当然还有设计文档）以方便他人，自己也可以顺便找些“前人栽树，后人乘凉”的幸福感觉来。因此，就“传承”而言，注释至少应该具备以下这些特点：

第一，注释应当浅显、明白。给程序加上些诘屈聱牙、形同天书的注释还不如不加的好。举个例子，我见过一个程序员把注释当作了他的私人日记本，在代码的注释中用只有他自己才懂得的特殊标记，把他在开发过程中的感想、计划、设计思路都记下来。他提交给项目经理的代码里居然还保留着这些临时性的记录，就像这个样子：

```
/*************************************************
*     5.4：CMA，in prj. C2  ---> 告诉经理，蓝图，4 号
*     5.7：版本，到 CVS，别忘了明天 checkout
*     5.11：修改算法，见会议纪要，在 Jack 那里
*          ------ 快提交了 ------ 变量名 nak2  --> n2
*      * 画图函数(rev.12.5.00.1 beta...b2 ) *
*************************************************/
void draw_picture_on_top_window( unsigned char * pic_buf, int windows_
handle )
```

第二，注释不是程序员指南。在注释中说下面的语句是循环而不是分支，或者在注释中说因为乘号比加号的优先级高所以要在哪里哪里加括号，这样的说法最好出现在编程教科书里。就这一原则来说，本章开头的 [illegible] 中的注释就是最好的反面教材。

第三，注释不是标准库函数参考手册。即使是一个资质一般的程序

员看到前任的代码中 [illegible] 函数旁边注释着“关闭文件”也会暴怒不已。逢到这个时候谁都会勃然大怒：“难道我偏不懂得 [illegible] 就是关闭文件，你竟敢对我的专业精神表示怀疑！”

第四，注释的主要任务是答疑解惑而不是增加程序的行数以换取上司的同情。逻辑复杂、流程冗长的地方注释是绝对必要的；有外部引用关系，不查其他代码不明白个中含意的地方注释也能担当重要角色；但代码本身清晰明确，有“自注解”特性的地方，注释还是深藏不露为好。

比方说，下面这段 [illegible] 代码的片段就异常清晰明白，任何懂得 悦语言或者 [illegible] 语言语法的人通读一遍之后就立刻可以说出这段代码的功用所在。这就是所谓具有“自注解”特性的代码了。在这样的代码里面，任何形式的注释都显得那么的多余和累赘。

```
static int test(int testval) {
    int result = 0;
    if (testval > target)
        result = -1;
    else if (testval < target)
        result = +1;
    else
        result = 0;
    return result;
}
```

最后一条，好的注释（尤其是好的算法注释）是对设计思想的精确表述和清晰展现，好的注释能够揭示代码背后隐藏的重要信息。

我们仍然举实例为证。下面是我从开放源代码的压缩软件 [illegible] 中提取的一个代码片段。[illegible] 是一个压缩或解压缩 [illegible] 格式压缩文件的代码库，完全开放和免费。包括 [illegible] 在内的许多著名压缩工具都使用了 [illegible] 提供的源代码，你可以在网址 [illegible] 上找到关于它的详细说明。一般来说，开发类似压缩、加密、数字签名、图像处理这样包含特定算法的程序时，代码背后多半隐含了许许多多数学上的、逻辑上的，甚至是经验上的东西，开发者如果不在注释中详加阐释，就一定会害得每一位阅读代码的程序员绞尽脑汁、身心憔悴。下面这段代码显然和数据压缩的核心算法有关，其中的英文注释就比较明白地讲述了隐藏在代码背后的故事：

```
/* The pkzip format requires that at least one distance code exists,
 * and that at least one bit should be sent even if there is only one
 * possible code. So to avoid special checks later on we force at least
 * two codes of non zero frequency.
 */
while (heap_len < 2) {
    int new = heap[++heap_len] = (max_code < 2 ? ++max_code : 0);
```

```
    tree[new].Freq = 1;
    depth[new] = 0;
    opt_len--; if (stree) static_len -= stree[new].Len;
    /* new is 0 or 1 so it does not have extra bits */
}
desc->max_code = max_code;
```

好，有了上面的五大原则，我们现在可以尝试着把代码 悦读之路 中的注释修改成更简洁、更有效的样子了。你可以对照着前面的 悦读之路 阅读下面的新代码 悦读之路 找出二者在注释方面的差别。

程序清单 悦读之路

```
#include <stdio.h>

/*
 * 主函数，返回 0 表示成功
 */
int main(int argc, char * argv[])
{
    FILE * fin;
    FILE * fout;
    int i, j;
    int b, g, r;
    int y;

    /* 打开输入文件后判断文件长度是否符合格式要求 */
    if ((fin = fopen("cat.pic", "rb")) == NULL)
    {
        puts("打开文件 cat.pic 时错误");
        return -1;
    }
    fseek(fin, 0, SEEK_END);
    if (ftell(fin) != 400 * 400 * 3)
    {
        puts("输入文件 cat.pic 不符合格式要求");
        fclose(fin); /* 异常处理时关闭已打开的文件 */
        return -1;
    }
    fseek(fin, 0, SEEK_SET);

    if ((fout = fopen("cat2.pic", "wb")) == NULL)
    {
        puts("打开文件 cat2.pic 时错误");
        fclose(fin); /* 异常处理时关闭已打开的文件 */
        return -1;
    }

    /*
     * 下面是图像转换的算法实现.彩色图像到灰度图像的转换主要利用
     * RGB 色彩空间到 YUV 色彩空间的变换公式来取得灰度值 Y，公式是：
     *        Y = 0.299R + 0.587G + 0.114B
     */
    for(i = 0; i < 400; i++)
```

```
        for(j = 0; j < 400; j ++)
        {
            b = fgetc(fin); g = fgetc(fin); r = fgetc(fin);
            y = (299 * r + 587 * g + 114 * b) /1000;
            fputc(y, fout);
        }

    fclose(fin);
    fclose(fout);
    return 0;
}
```

源摇补充说明

让我们再简单回顾一下有关注释的话题。我们除了知道注释必须在合适的时间、合适的地点以合适的形式出现以外，还可以体会到一些其他的东西：

源有些时候代码中的空行可以起到注释的作用。例如我们可以用空行将一段略显冗长的算法实现划分成一个又一个逻辑段落。显然这可以对我们理解代码提供不小的帮助。

源复杂表达式、复杂的参数表中，空格的存在可以让阅读者对代码的内涵一目了然。更进一步，我们还可以发现有些时候适当地换行也可以将一个关系复杂的代码行自然地分成逻辑片段，以帮助读者快速阅读。——好的习惯是无论表达式复杂与否，都在必要的地方（如操作符两边、逗号后面）加上空格。

源引申而言，能够起注释作用的绝不仅仅是程序中那些可见的[illegible]。其他很多东西，像空行、换行、空格、制表符以及代码的排版格式等等都可以起到注释的作用。规范、清晰的变量名或函数名（比如[illegible]，等等）也同样能让人轻松不少。

源注释可以让程序更加清晰易读，但过于繁琐无聊的注释同样会让程序面目全非，使后来者不忍卒读。

缘摇总结一下

第一个案例比较简单，不过大家至少要记住：

- 摇注释不是越多越好。
- 摇不要亦步亦趋。
- 摇多站在后来者的角度想一想。

员摇问题引入

有一位曾经与我共事的程序员，眼下已经成了自由职业者，每天异常悠闲地在家中为多家软件公司提供与 韵猢和图像处理有关的代码库，收入颇丰。这位世外高人的姓氏拼音以 在开头，我们在这里姑且叫他 在先生吧。我有幸拜读过 在先生的大作，也有幸被那些天书般难懂的程序惊得目瞪口呆。直到今天我仍然无法说服自己忘记那些和《蒙古秘史》一样无从索解却又能被任何编译器成功编译的代码。为了让读者也可以分享我所经历的一切，同时又不妨碍本书的主旨和篇幅，我试着用 在先生的风格写了下面一段简单的代码：

程序清单 圆摇杂藻鳖射操槽

```
#include <stdio.h>
#include "SealedBook.h"

zV zS(zI * d, zI l);

zI main()
{
    zI zi, l = 10;
    zI d[] = {20,30,10,1,0,5,100,50,30,1};

    zS(d, l);
    zLoop(zi, 0, l-1)
        printf("%d\n", d[zi]);
    return 0;
}

zV zS(zI * d, zI l)
{
    zPreSw;
    zI zi1, zi2;
    zLoop(zi1, 0, l-2) zLoop(zi2, zi1+1, l-1)
        if (d[zi1] < d[zi2])
            zSw(d[zi1], d[zi2]);
}
```

我不知道你们是否理解了这段代码的含义，我也不知道你们如何看待这样的代码风格；我只知道这一段代码的晦涩程度与 在先生那些包含着复杂的高等数学公式 (我还记得“卷积”、“拉普拉斯算子”之类高深莫测的术语) 的代码相比，只不过是小巫见大巫罢了。当然，上面的代码肯定可以被任何一种符合 粤藻鳖规范的 悦语言编译器编译通过，前提是必须在工程中包含下面这个头文件：

程序清单 圆圈圈名称被删掉了的操蛋版

```
#ifndef __ZSH__
#define __ZSH__

typedef void zV;
typedef char zC;
typedef int zI;
typedef float zF;
typedef double zD;

#define zLoop(i,m,n) for(i=m;i<=n;(i)++)

#define zPreSw zI zM
#define zSw(i,j) {zM=i;i=j;j=zM;}

#endif
```

噢，你一定可以猜出这段代码是干什么的了。有了上面这个头文件的帮助，你也一定可以明白 在先生的玄机所在了。有人会站起来说：“没有意义嘛，鱼目混珠，牵强附会，指鹿为马，指桑骂槐，简直故弄玄虚！让我们花时间读这种东西还不如让我们去做《南方周末》上的‘小强填字’。”先别着急，我们静下心仔细想一会儿，或者先听我讲些无关痛痒的闲话，然后再来关注 在先生风格的代码。——要知道，在先生三杯酒下肚之后，可以一口气说出三百五十八条你不得不信服的理由来证明 在风格的正确和必要。

圆圈圈一些题外话

讲到程序风格就不能不讲到标识符的命名。这里说的标识符包括程序里的变量名、常量名、参数名、函数名、过程名、方法名、类名、结构名、接口名……一般说来，程序中最能体现程序员风格的大概就是标识符的命名方法了。举个简单的例子，我见过的一段管理银行对公账户的 交圈圈圈圈程序为账户信息定义的数据结构是这样子的：

```
'账户信息的数据结构
Public Type ZH
    ZH As String        '账号
    DQH As String       '地区号
    WDH As String       '网点号
    BZ As Long          '币种
    KHRQ As Date        '开户日期
    ZHDZ As String      '账户地址
    ZHDH As String      '账户电话
    YJDM As Long        '印鉴代码
```

```
    BZH As String      '备注
End Type
```

我几乎第一眼就发现了这一 [illegible] 结构中，“币种”字段和“备注”字段的汉语拼音缩写同样是“BZ”或者“bz”，天才的作者将“币种”写成“BZ”而将“备注”写成“bz”，有效地防止了编译失败。很难想象编程新手该如何去继承这样一个超过 [illegible] 万行代码的庞大软件，我猜程序的作者一定会再次发挥他天才的想象力，为后来者编写一本“标识符速查手册”，按字母顺序为程序中每一个标识符注明其含义和使用方法。

很显然，不规范或者不合逻辑的标识符命名规则是必须摈弃的编程风格。不过现在我们还是先抛开它们，来看一看规范的命名规则中，是不是还有不同风格的存在。

在 [illegible] 操作系统上进行软件开发的程序员们一定很熟悉“微软风格”，这一风格在标识符命名上的具体表现就是将所谓的“匈牙利规则”发展到了极致，从 [illegible] 到 [illegible]，从 [illegible] 到 [illegible]，微软的源代码中到处都弥漫着西雅图和布达佩斯街头的气味儿。下面这些标识符名称都取自微软公司的源代码：

```
dwFileAttributes          [Win32 SDK]
lpsaActions               [Win32 SDK]
bNoByteSwap               [MFC]
m_lpBufMax                [MFC]
ISupportErrorInfoImpl     [ATL]
AtlReportError            [ATL]
daoTable1                 [Visual Basic]
xlBook                    [Visual Basic]
objBrowsCap               [ASP]
```

与比尔·盖茨倡导的风格相左，长期以来在 [illegible] 世界占据主流的是传统的 [illegible] 或者 [illegible] 风格，具体到标识符命名来说，[illegible] 代码中最常见的是以下划线分隔的小写单词组。以下标识符名称选自 [illegible] 公司支持多种操作系统的数据库软件 [illegible] 的源代码：

```
expand_filename
in_sw_value
out_file
ib_fprintf
dtype_sql_time
field -> fld_array_info -> ary_dtype
```

[illegible] 平台下的代码也具有大体类似的特征，以下标识符名称选自 [illegible] 的源代码：

```
ip6_dev_loopback_xmit
```

```
skb->sk->net_pinfo.af_inet6.mc_loop
clear_page_tables
lock_limit
request_module
```

当然还应当提到 先生风格。典型 先生风格的 程序中，类名和接口名是单词间没有分隔符的，首字母大写的单词或单词组，但变量名、方法名则采用第一个单词全小写，其后的单词首字母大写的形式。例如下面这些标识符（选自 网上随意操）：

```
CreateSuppliers             [类名]
CustomControlsContext       [接口名]
faceLabel                   [变量名]
toolPalette                 [变量名]
buildEditMenu               [方法名]
openInBox                   [方法名]
```

猿猿案例分析

好，现在让我们回到 在先生风格的程序上来。在先生一向对他那天书般难懂的代码无比自豪。据我揣摩他的理由大致包括：

即使用 打头的变量名、函数名、宏定义名就像是为自己的旅行箱贴标签，或者为自己开出的个人支票签名。这一行为可以在最大程度上保证 在先生这样的自由职业者的利益——即使他人未经授权使用了 在先生的代码，使用者要想在短期内消除遍布于程序中的“ 打”标记也绝非易事。

即将 增强 变成 变量等基本数据类型重新命名，可以在最大程度上保证代码的可移植性。例如从 元位程序向 猿位程序移植时的问题就可以使用此方法简单解决。这一理由还有微软公司为其提供佐证：宰强国网运中大量出现的 战量栽 阅宰部驱 结等硕韩测量栽延等定义均属此列。

猿即使用宏定义表示一些基本的功能运算可以大幅度减少源程序的行数，也可以被后来者方便地复用。代码 杂猿猿用黑操中定义的数据交换宏 挖槽以及 托猿猿槽就是这个意思。虽然 托猿猿槽完全是为了预先定义中间变量 插以供 挖槽使用，但既然有微软 粤栽模板中 战猿测量栽延等宏为 部猿延栽 栽用抹延等宏提供中间变量在先，在先生的做法也不算特别的唐突。

源即使用高度简练的缩写可以加强源代码的保密程度。即使 在先生的代码被他人剽窃，小偷们也未必能懂得 挖中的 杂就是 杂猿猿 挖槽中的

[illegible]就是 [illegible]，[illegible]就表示 [illegible]类型。这一原则对于 [illegible]之类的算法程序尤其重要。

我相信大家早已知道了代码 [illegible]就是一段冒泡排序的小程序，我也相信大家对以上 [illegible]源条理由没有办法接受。我和大家的立场相同，我们坚决反对程序员使用类似 [illegible]先生这样的风格编写程序。从某种意义上说，当所有程序员都在编写天书般的代码时，计算机就可以休长假了（我非常想在这里再讲一遍上帝和通天塔的老故事，但由于篇幅的限制还是忍住了）。上面 [illegible]先生的 [illegible]源种说法都有道理，但这与代码的可读性之间并没有根本的矛盾，因为：

[illegible]即个人风格不等于个人标签。个人标签可以用版权信息、[illegible]活节彩蛋等形式展现。如果项目组中都按姓氏缩写给变量取名，你的变量叫 [illegible]，他的变量叫 [illegible]，还有一个的变量叫 [illegible]，那不乱套才怪。

[illegible]即在特定的条件下，重新包装的数据类型名称确实有利于移植，但前提是新的名字必须明白易懂。比如 [illegible]，[illegible]等就是不错的名字，而 [illegible]，[illegible]或者 [illegible]就不那么好懂了。

[illegible]即宏定义的名字同样应当清晰可读。当然 [illegible]这样替换标准 [illegible]则循环的写法我们是不推荐的。还有，像微软的 [illegible]这样比较难于使用，甚至还包括了隐含的变量定义的宏组合，我们还是尽量避免为好（完全可以把它们封装成类或者库函数的形式）。至少我个人就从不认为微软的代码处处都是学习的范本。

[illegible]即保密和代码可读性是两回事。如果你在数据安全、通讯安全、密码学等方面一无所知的话，那你宁可去买一个可以把计算机锁起来的保险柜也不要去选择 [illegible]先生那种愚蠢的“保密”方式。

现在我们知道，[illegible]先生的风格根本不能叫做风格。那么，什么才是程序员的个人风格呢？或者说程序员在清晰、规范、可读以及标准化的前提下还有机会展现个人风格吗？让我们看下面的代码片段：

```
/* 风格1 */
for(int i = 0; i < 100; i ++) {
    a[i] + = i;
}

/* 风格2 */
for(int i = 0; i < 100; i ++)
{
    a[i] + = i;
}
```

[illegible]在 [illegible]，[illegible]，[illegible]乃至 [illegible]中，你都有权利选择这两种大括号风格中的一种。此外，你还可以在下面几种注释风格中选择你的最爱：

```
/*
 * C 风格
 */

/*
 * javadoc 风格
 *
 */

//////////////////////////////////////////////////////////////////
//Visual C ++ 风格

//
//比较另类的风格
//

//代码行之上的注释
i = j + 1;

i = j + 1;    //代码行旁边的注释
```

当然还应该包括上面提到的标识符命名的风格、代码中空行的风格、代码中常数数组的排列风格、源代码文件的划分风格、头文件的使用风格、版本记录风格、版权信息风格、类成员的排列风格、接口定义的风格……诸如此类，不一而足。程序员在方便沟通、规范开发的前提下，可以独立抉择、体现个性特点的地方是很多很多的。优秀的程序员可以使自己的产品同时满足以下要求：

- 摇继任者可以轻松阅读自己的代码
- 摇继任者可以轻松复用自己的代码
- 摇继任者可以轻松修改自己的代码
- 摇继任者可以轻松地辨别出哪些是自己写的代码，哪些是别人写的代码

源摇补充说明

就我个人而言，我从来也不认为编程是一种机械化的劳动；即使一个软件开发的项目组在项目管理的标准化和软件过程的规范化方面已经登峰造极（比如已经通过了 灿遍 惆酝转的认证），我也不会相信该项目组中产生的所有代码都会像是同一台代码自动加工机的产物，以至于没有注释就无法辨别这是谁的大作。毕竟，编写代码的工作和拿着游标卡尺加工标准件的工作相去甚远。哪怕是未来的某个时候我们掌握了统一项目组内所有程序员的编程风格的技术手段，我们也很难下决心让我们的

程序员们抛弃那些他们多年来积累形成的、在一定程度上代表着他们每个人个性特质的东西——只要这些东西还没有妨碍到我们之间的交流和沟通，只要我们还没有下决心让机器人来替换项目组的所有成员。

回到我们刚才讨论过的标识符命名问题上来。其实，无论是“微软风格”、“[illegible]风格”、“[illegible]风格”还是“[illegible]风格”都同时具备优越的一面和不那么优越的一面。我们在选择一种风格的同时也要冒被这种风格束缚的危险。比方说，[illegible]风格的标识符名称有助于定义与类名相同又没有歧义的变量名，例如：

```
ComboBoxDemo comboBoxDemo = new ComboBoxDemo()
```

但很难说这种做法不是一种混淆视听的隐患，我就经常在这样的代码中把类名和变量名搞混。类似的，“匈牙利风格”在诞生之初曾被认为是表达能力最强的命名风格。但很快，人们在“匈牙利风格”的拥护者微软那里发现了许多严重的问题，其中之一是：一旦修改变量的类型，也就必须修改变量的名字。这一问题在必须保持接口稳定性的[illegible]中更为棘手。最著名的例子是微软的窗口回调函数：

```
LRESULT CALLBACK WindowProc(
  HWND hwnd,      //handle to window
  UINT uMsg,      //message identifier
  WPARAM wParam,  //first message parameter
  LPARAM lParam   //second message parameter
);
```

其中的参数[illegible]在[illegible]平台上是[illegible]型，在[illegible]平台上明明已经升作了[illegible]型，却还保留着傻傻的前缀“[illegible]”。微软为了自圆其说，才又引入了更加让人摸不着头脑的[illegible]和[illegible]类型。

总的说来，个性、气质、风格都是些很难琢磨的东西。就像你说你喜欢黑泽明而非张艺谋是因为你喜欢黑泽明那种可以在暗夜的荒沼中让你心弦悸动的风格，但你又没法用[illegible]或者随便什么你认为最精确的语言来说清这种风格到底是什么东西，有着什么样的内在结构，在你心中最火爆的电影频道里究竟占据了多少份额。大多数人总愿意相信，通过项目经理的努力，可以在最好的软件项目组中统一所有人的编程风格，可以达到“沟通无障碍”这一传说中的境界。但事实是，程序员们必须在编程前做出痛苦的抉择，然后又在编程中更加痛苦地克服某一风格的局限。承受这些痛苦的最好回报是我们终于可以在更高的层次上驾驭我们的代码，我们终于可以让我们的代码看起来更舒服、流畅、成熟和有魅力。

缘摇总结一下

- 摇风格不要妨碍沟通。
- 摇混合多种风格等于没有风格。
- 摇没有个性特点的代码未必就是好代码。

员摇问题引入

我非常高兴在这本书里和大家分享软件开发的乐趣，我尤其高兴在这一节的开始部分向大家展示一下我们业已熟悉的那些程序设计语言的魔力。我说“魔力”的意思是我真的不敢相信世界上还存在着这么些稀奇古怪的代码。我一边小心地逐个编译、调试这些代码，一边谨慎地总结、归纳出以下的结论：请你仔细阅读以下代码，一旦你掌握了这些代码的编写秘诀，你就掌握了某种让你的后来者（比如测试员、代码维护员）目光呆滞、精神失常的有效手段，你就拥有了让老板在明天一早就把遣散费塞到你的手里把你解雇的最佳方法，你就学会了数十种可以让你眼前的电脑在顷刻间不省人事的绝招……我实在忍不住不向大家介绍这些代码。

下面这段 悦垣回代码是一个成绩评价系统的一部分。代码要求用户输入分数 灶，然后根据分数的高低给出“不及格”、“良好”、“优秀”等评语，其中使用了 泽翻翻渠() 分支结构。

程序清单 猿员摇韵瑶瑶瑶瑶瑶瑶责

```
#include <iostream.h>

void main()
{
    int n;
    cout << "请输入 n = ?"; cin >> n;

    if (0 <= n && n <= 100)
        switch(n/10)
        {
        case 0: case 1: case 2:
        case 3: case 4: case5:
            cout << "不及格";
            if (0)
        case 6: case 7:
            cout << "及格";
            if (1 == 0)
        case 8:
            cout << "良好";
            if (1 < -1)
        case 9:
            cout << "优秀";
            if (!!!!!1)
        default:
            cout << "满分";
        }
```