《编程格调》

图书基本信息

书名：《编程格调》

13位ISBN编号：9787115379521

出版时间：2015-3

作者：[美] Brian W. Kernighan,[美] P.J. Plauger

页数：180

译者：高博,徐章宁

《编程格调》

内容概要

《编程格调》是编程惯用法和规则的实践指南。全书从表达、控制结构、程序结构、输入和输出、常见错误、效率和测试工具、文档等多个角度，概括了程序设计中若干的最佳实践或规则，并通过代码示例加以分析和阐释。
本书两位作者都是程序设计领域的大师级任务。他们四十年前在本书中给出的70多条最佳实践和规则，大多数在今天仍然适用。
本书堪称计算机专业人士和程序员的必读的经典之作，适合于不同层级的程序员和计算机相关专业的学生参考阅读。

《编程格调》

作者简介

作者简介
Brian W. Kernighan，全球知名、德高望重的计算机先驱，在程序设计方法论和软件工程方面做了大量开创性的工作。他曾长期在贝尔实验室工作，现在普林斯顿大学计算机科学系任教。他著有数本经典教材，包括与Dennis Ritchie合著的传世之作The C Programming Language、与Rob Pike合著的The Practice of Programming，以及最近出版的科普图书D is for Digital等。他还是AWK编程语言的发明者，这种语言广泛地应用在UNIX/Linux应用中。“ K&R C ”和“ AWK ”中的“ K ”都代表Kernighan。
P.J. Plauger，全球知名的计算机科学家、C/ C++技术专家以及技术图书作者，更是数个标准C/C++程序库的作者。他曾经在贝尔实验室工作，现在任美国Dinkumware公司总裁。他曾经担任 C/C++ Users Journal高级编辑，也是The Standard C Library、Standard C : A Reference 和The Standard Template Library等图书的作者。
译者简介
高博，1983年生，毕业于上海交通大学。目前在互联网金融创业公司任首席产品官兼首席质量官，在信息科学和工程领域有近15年实践和研究经验。酷爱读书和写作，业余研究兴趣涉猎广泛。译著包括图灵奖作者高德纳的《研究之美》和布鲁克斯的《设计原本》，以及Jolt大奖作品《元素模式》等，出版翻译作品计近百万字。
新浪微博：但以理_高博
微信公众号：高博的世界
徐章宁，1984年生，就读于上海交通大学，硕士毕业后就职于EMC中国卓越研发集团，现任EMC公司高级系统管理工程师，从事软件运维工作多年，钟爱开源软件。对各类知识有广泛兴趣，平日喜爱参与问答网站讨论，热爱读书摄影和写作。

《编程格调》

书籍目录

《编程格调》

《编程格调》

精彩短评

1、精辟，就是fortran看着脑仁疼，还有那个年代还在跟goto斗争，看着满篇的如何消除goto的介绍也是醉了
2、精辟的书，但是没有什么实际价值，有自我修养的程序员看一眼目录就知道这本书写的是啥，修养没到位的光看也不一定起作用，还是要实践
3、40年前的内容，大部分还仅限于fortran这种古老的语言缺陷方面，看看目录就知道大致的内容了，推荐给编程新人吧
4、原著书名为 The Elements of of Programming Style, 胆敢取名 elements 的书还可以一版再版的，都牛得不得了，比如《几何原本》。此书与《几何原本》从几条简单原理推导出整个世界类似，给出若干来自实践的原则和原则应用的案例，如何应用还得你我在实践中参详，像习题集。对于计算机专业短短的历史，这些原则非常古老，书成于FORTRAN流行的时代。但是真正本质的原则往往从行业诞生之日就隐约存在，并不需世事变迁而有所损益，只是等待时间雕琢慢慢让它们清晰起来。比如C语言的成功并不像课本前言中写得那么轻易和简单，超越同时代诸多竞争语言赢得声名和后继影响决非偶然。本书作者之一即是C语言和UNIX创造者之一。

5、语言使用fortran 有点过时
6、虽然出版时间很久了，但是里面的道理仍然不过时
7、很多编程的原则不会随着事件流失，但是 LISP的确看的很蛋疼
8、写代码应该注意的一系列问题。

《编程格调》

精彩书评

1、一、表达1.写清楚2.保守使用临时变量3.明白无歧义4.不要自己造所有的工具5.确保判断测试条件清楚易读二、控制结构1.先用你顺手的高级语言写一遍程序，这时可以看清楚算法并作相应的调试，程序正确之后，再翻译成你编译器所处理的语言2.子例程和函数，把代码分成可独立管理的小段3.规划数据结构的时候要像规划控制流一样小心，尝试找到可以起到简化程序作用的数据表示形式三、程序结构1.编写和维护大型程序唯一的方法是把它分解成一组函数、子例程和过程2.每一个模块应该只处理问题的一个方面，否则的话，模块就会变得大而复杂3.相对于其他模块，一个模块应该隐藏它如何完成工作的细节，否则这个模块就无法独立于其他模块来进行修改四、输入和输出1.校验输入数据的合法于合理性2.保证输入数据不会违背程序的限制3.利用文件结束符号或结束标志来终止输入，不要让用户去计数4.识别输入错误并尽可能地恢复，不要一碰到错误就停止，也不要忽略错误了事5.在输入和输出时采用助记符，使得输入数据易于正确地准备，在输出时同时也將输入和默认值输出，并使得输出数据不言自明6.將I/O局限起来，不要散布在整个程序中。將处理文件结束和缓冲区等细节隐藏在函数中7.保证程序的结构反映出其所处理的数据五、常见错误1.使用变量之前要初始化2.小心"差一"错误，确保在正确的次数下完成循环，对于等式的比较判断，确保有正确的分支处理3.检查数组的下标不要越界4.避免循环有多个出口5.检查程序的内部边界6.预防性编程，搞清楚哪些事可能出问题，并且添加代码以检查7.不要用浮点数来做累计，不要期望浮点小数数值遵循你熟悉的算术法则，他们不适用六、效率和测试工具1.假如程序是错误的，再快也没用。2.让代码言简意赅。撰写时不要惦记着它运行得快。过早优化是万恶之源3.别为每个优化细节去操心，让编译器去照顾这些4.用心于算法，而非代码细节。记住，数据结构对算法的实现方式有显著影响5.在构造程序时放置测试语句，在决定"增效"之前先执行测试。在程序演进过程中，可將测试语句留下来随时运用七、文档1.如果程序不正确，文档写得再好都没有用2.如果文档和代码不一致，那程序就没什么价值3.要在最大程度上使得代码本身就是自己的文档，如果做不到，重写代码而不是做文档的补充。好的代码需要的注释量要比坏的代码少4.注释提供的信息应该是从代码上无法获得的5.帮助记忆的变量名和标签，以及强调逻辑结构的代码布局，都有助于程序自身的文档化

《编程格调》

章节试读

1、《编程格调》的笔记-第155页

原文

```
      DIMENSION X(300)
      READ 1,N,(X(I),I=1,N)
1     FORMAT(I3/(F5.1))
      K=N-1
6     J=1
19    L=0
      DO 2 I=J,K
      IF(X(I)-X(I+1)) 2,2,3
3     IF(L) 20,21,20
21    J1=I-1
20    SAVE=X(I)
      X(I)=X(I+1)
      X(I+1)=SAVE
      L=I
2     CONTINUE
      IF(L) 8,9,8
8     K=L
      IF(J1) 6,6,7
7     J=J1
      GO TO 19
9     END
```

对应的C代码如下。

```
void f_sort(float*x, unsigned int n)
{
int profile_cmp=0;
int profile_swap=0;
int profile_check_L=0;
int profile_check_J1=0;
int profile_move_K=0;

int i__1;

int i__, j, k, L;
int j1;
float save;


    k = n - 1;
L6:
   j = 1;
L19:
   L = 0;
   i__1 = k;
```

```
   for (i__ = j; i__ &lt;= i__1; ++i__) {
       profile_cmp++;
       if (x[i__ - 1] - x[i__] &lt;= 0.f ) {
           goto L2;
       } else {
           goto L3;
       }
L3:
    if (L != 0) {
       goto L20;
    } else {
       goto L21;
    }
L21:
    j1 = i__ - 1;
L20:
    save = x[i__ - 1];
    x[i__ - 1] = x[i__];
    x[i__] = save;
    L = i__;
    profile_swap++;
L2:
    ;
   }
    profile_check_L++;
   if (L != 0) {
    goto L8;
   } else {
    goto L9;
   }
L8:
    profile_move_K++;
   k = L;
    profile_check_J1++;
   if (j1 &lt;= 0) {
    goto L6;
   } else {
    goto L7;
   }
L7:
   j = j1;
   goto L19;
L9:
   printf("\nf n: %d\ncmp: %d\nswap: %d\ncheck_L %d\ncheck_J1 %d\nprofile_move_K
%d\nprofile_check_tatal %d\n",
        n, profile_cmp, profile_swap, profile_check_L, profile_check_J1, profile_move_K,
        profile_check_L + profile_check_J1 + profile_move_K +profile_cmp);
/*    int i=0;
   for(i=0;i&lt;n;i++)
```

```
  {
    printf("%f\t", x[i]);
  }*/
  return;
}
```

2、《编程格调》的笔记-第155页

下述问题部分已解决，见[http://book.douban.com/annotation/37079021/].

我试图把书中第155页的烧脑FORTRAN代码移值为C代码，重复排序的实验，失败。
不能重现出 比较次数少 的效果。程序太乱了，另外fortran下标下界是1，且程
序把下标与0比较。这可能是我的C版本代码的错误所在。

求教过路君子，FORTRAN到C的移植。

1.1 原书155页烧脑版代码如下。
1.2 变量说明
2. 我实现的C版本（有问题的）如下:

```
void brainbreak_sort (float* x, unsigned int n)
{
    int profile_cmp=0;
    int profile_swap=0;
    int profile_check=0;
    int c=0;
    int K=n-1; // n-1
    int J=0;
    int L=0;
      int i=0;
      double save=0;
      int J1=0;

label_6:
    J=1; //1
label_19:
    L=0;

    for(i=J;i&lt;K;i++)
    {
        profile_cmp++;
        if(x[i]-x[i+1]&lt;0) goto label_2;
        if(x[i]-x[i+1]==0) goto label_2;
        if(x[i]-x[i+1]&gt;0) goto label_3;

        label_3:
        if(L&lt;0) goto label_20;
        if(L==0) goto label_21;
        if(L&gt;0) goto label_20;
```

```
        label_21:
        J1=i-1;//i-1

        label_20:
        save=x[i];
        x[i]=x[i+1];
        x[i+1]=save;
        profile_swap++;
        L=i;// i

        label_2:;
    }

    if(L&lt;0) goto label_8;
    if(L==0) goto label_9;
    if(L&gt;0) goto label_8;

    label_8:
    K=L;

    if(J1&lt;0) goto label_6;
    if(J1==0) goto label_6;
    if(J1&gt;0) goto label_7;

    label_7:
    J=J1;
    goto label_19;
    label_9:
 printf("brainbreak n: %d\ncmp: %d\nswap: %d\n", n, profile_cmp, profile_swap);
        ;
/*    for(c=0;c&lt;n+1;c++)
    {
        printf("%f\t",x[c]);
    }*/
}
```

3. 作为对照，原书的平凡版排版代码如下:
4. 我实现的平凡版的C版本如下:

```
void plain_sort (float* x, unsigned int n)
{
    if (n &lt; 2 ) return;
    int i, j;
    double save;
    int profile_cmp = 0;
    int profile_swap = 0;
    for (i=2; i&lt;n; i++)
    {
        for(j=0;j&lt;i;j++)
```

```
        {
            if(x[i] &gt;= x[j]) {profile_cmp++; continue;}
             save = x[i];
             x[i] = x[j];
             x[j] = save;
             profile_swap++;
        }
    }
    printf("plain n: %d\ncmp: %d\nswap: %d\n", n, profile_cmp, profile_swap);
/*   for(i=0;i&lt;n;i++)
   {
     printf("%f\t", x[i]);
   }*/
}
```

5. 原书中profile对比
我的对比如下，问题是 我的烧脑版的比较次数比平凡版要高。

6. 我的版本: 完整的程序、随机数生成、脚本。
6.1 完整的程序

```
#include "stdlib.h"
#include "stdio.h"

void plain_sort (float* x, unsigned int n);
void brainbreak_sort (float* x, unsigned int n);


int main()
{
   float x[2000];
   float y[2000];
   int count=0;
   while(scanf("%f", &amp;x[count++])!=EOF);  //{printf(":%f\n",x[count-1]);}
   int i=0;
/*   for(i=0;i&lt;count;i++)
   {
      printf("%f\n", x[i]);
   }
   printf("\n");
 */
   for(i=0;i&lt;count;i++)
   {
    y[i]=x[i];
   }
   plain_sort(x, count-1);
   brainbreak_sort(y, count-1);

   return 0;
}

void brainbreak_sort (float* x, unsigned int n)
```

```
{
    int profile_cmp=0;
    int profile_swap=0;
    int profile_check=0;
    int c=0;
    int K=n-1; // n-1
    int J=0;
    int L=0;
      int i=0;
      double save=0;
      int J1=0;

label_6:
    J=1; //1
label_19:
    L=0;

    for(i=J;i&lt;K;i++)
    {
        profile_cmp++;
        if(x[i]-x[i+1]&lt;0) goto label_2;
        if(x[i]-x[i+1]==0) goto label_2;
        if(x[i]-x[i+1]&gt;0) goto label_3;

        label_3:
        if(L&lt;0) goto label_20;
        if(L==0) goto label_21;
        if(L&gt;0) goto label_20;

        label_21:
        J1=i-1;//i-1

        label_20:
        save=x[i];
        x[i]=x[i+1];
        x[i+1]=save;
        profile_swap++;
        L=i;// i

        label_2:;
    }

    if(L&lt;0) goto label_8;
    if(L==0) goto label_9;
    if(L&gt;0) goto label_8;

    label_8:
    K=L;
```

```
    if(J1&lt;0) goto label_6;
    if(J1==0) goto label_6;
    if(J1&gt;0) goto label_7;

    label_7:
    J=J1;
    goto label_19;
    label_9:
 printf("brainbreak n: %d\ncmp: %d\nswap: %d\n", n, profile_cmp, profile_swap);
        ;
/*    for(c=0;c&lt;n+1;c++)
    {
       printf("%f\t",x[c]);
    }*/
}

void plain_sort (float* x, unsigned int n)
{
     if (n &lt; 2 ) return;
     int i, j;
     double save;
     int profile_cmp = 0;
     int profile_swap = 0;
     for (i=1; i&lt;n; i++)
     {
         for(j=0;j&lt;i;j++)
         {
               if(x[i] &gt;= x[j]) {profile_cmp++; continue;}
                save = x[i];
                x[i] = x[j];
                x[j] = save;
                profile_swap++;
         }
     }
     printf("plain n: %d\ncmp: %d\nswap: %d\n", n, profile_cmp, profile_swap);

/*    for(i=0;i&lt;n;i++)
    {
      printf("%f\t", x[i]);
    }*/

}6.2 随机数生成
#include &lt;stdlib.h&gt;
#include &lt;stdio.h&gt;

int main(int argc, char* argv[])
{
    if(argc!=2)return 1;
    int count=0;
```

```
    int i=0;
    const int MAX =100;
    const int MIN =0;

    srand((int)getpid());
    sscanf(argv[1], "%d", &amp;count);

    for(i=0;i&lt;count;i++)
    {
        printf("%d\n", rand()% (MAX + 1 - MIN) + MIN);
    }

    return 0;
}6.3 脚本
gcc rand.c -o rand; ./rand 10 &gt; rand2000.txt
gcc sort.c -o sort; ./sort &lt; rand2000.txt
```

《编程格调》

版权说明