C 语言实例对比：从抽象概念到代码直观

李睿凡　李蕾
（北京邮电大学计算机学院，北京，100876）

摘　要：本文从程序设计语言实用化的角度出发，使用实例对比的方法，将 C 语言中涉及软件工程而又容易被忽视的抽象概念，如可读性、健壮性、编码风格、设计思路等进行抽象到具体比较分析。明确了 C 语言中语法规则学习的目标，为 C 语言的应用奠定良好的基础。

关键词：比较法；C 语言；可读性；健壮性；编码风格
中图分类号：TP312　　**文献标识码**：A

1　引言

C 语言是计算机、信息类各专业的必修基础课，通常开设于计算机导论等课程之后。虽然学生有了计算机的基础知识，但对于初次接触计算机编程语言的学生而言，C 语言课程仍然具有一定的学习难度与挑战。这些难点与挑战主要表现在：如何将程序思想转化为 C 语言的代码，也就是如何使用语言进行一般问题求解的过程。此外，C 语言本身有难于理解的地方，如指针。除此之外，编写代码风格问题，程序设计的健壮性、可读性是教学中十分重要但又容易忽略的问题。

到目前为止，不少计算机教育的专家提出了很多改善教学的手段与方法，在文献中都有体现。例如，谭浩强教授编写的《C 高级语言程序设计（第二版）》教材中多处提到如何进行 C 语言教学的问题。例如，对于 break 与 continue 等语句的使用，书中也以直接比较的方法展开论述，具有良好的效果。此外，论文提出了在中央民族大学文科专业学生学过 Visual FoxPro 数据库管理系统编程语言后，应如何运用比较教学法、开展相对复杂的 C 语言程序设计的教学工作。这种方法当属借鉴不同计算机语言种类之间相似性的方法。但是，很多计算机、信息类专业是直接开设 C 语言的课程，没有过渡的计算机语言课程，更多是讨论语言规则的差异，因而这种方法的局限性比较大。

基于以上考虑，本文从如何将 C 语言的教学从语言规则的学习转化为以程序设计为中心的程序编写工具的学习思路出发，从程序设计、可读性、健壮性等若干方面采用实例对比法进行了探讨。本文从比较这一基本观点与方法出发，以实例代码的形式启发学生，破解其中的难题，理解、运用 C 语言以及养成良好的编程习惯，为计算机语言的应用打下良好的基础。

2　对比实例

2.1　程序版式与可读性

C 语言的编写版式非常灵活，可以一行中包含多个语句，多种语句之间可以嵌套。但是这

种灵活性给学生的直观感觉是可以随意写。教师反复强调缩进等编写格式的重要性，强调版式对于代码可读性的重要。但学生在作业、上机调试程序中仍然有这样的问题。有些学生认为，程序能运行正确即可，不用管那些格式问题。

笔者采用实例对比的方法，将一段具有良好编写风格的程序与较差编写风格的程序相比较，请学生先阅读后指出差与好的原因与自身体会。这样能够有效地加深学生对于编码格式问题的重视。程序代码 1.a（见图 1）与程序代码 1.b（见图 2）是比较代码片段的例子，主要功能是逆序打印 150～50 之间能被 2 整除但不能被 3 整除的整数。

程序代码 1.a

```
#include <stdio.h>
int main(void)
{int i;for (i = 150; i >= 50; i--)
{if (i%2 == 0 && i%3 != 0)
printf("%d ", i);}printf("\n");return 0;}
```

图 1　较差版式的实例

程序代码 1.b

```
#include <stdio.h>
int main(void)
{
    int i;
    printf("50 到 150 之间能被 2 但不能被 3 整除的数:\n");
    for (i = 150; i >= 50; i--)
    {
        if (i%2 == 0 && i%3 != 0)
        {
            printf("%d ", i);
        }
    }
    printf("\n");
    return 0;
}
```

图 2　良好版式的实例

结果，从学生的代码阅读、回答讨论效果上来看，基本上对程序版式的重要性有了认识与体会。能够总结出一些共性的规则要求，包括空行、代码行内的空格、符号对齐、长行的拆分等内容。这样，有助于学生自我领会以及养成良好的习惯，从而有力地避免了学生对于程序版式的轻视。

2.2　命名规则与可读性

程序的可读性还与命名规则有关，而命名规则在教材中的重视也不足。教材中的程序代码变量等命名并未都做到“见名知意”，这给学生的理解带来了不少困难。在教材中，仅仅以“见名知意”、“选择有含义的英文单词”等原则讲解知识至少有两个问题：一是大学一年级学生的英文水平不足，单词量十分有限，所以在选择单词时无法写出；二是“见名知意”一词的表达太过抽象，没有具体的措施与手段。

对于以上问题，笔者提出以下解决方案：一是建议使用中文的拼音替代英文名字，以方便学生；二是介绍“匈牙利命名法”、“Windows 应用程序命名规则”、“UNIX 应用程序命名规则”等命名规则。例如，已知圆半径 r 和圆柱的高 h，求圆周长、圆面积、圆球表面积、圆球体积、圆柱体积。代码见程序代码 2.a（见图 3）与程序代码 2.b（见图 4）。很容易看到，加入命名规则后，变量的名称确实能够做到“见名知意”。对于 Windows 命名规则，这里不做详细说明，请见相关文献。

程序代码 2.a

```
#include <stdio.h>
void main(void)
{
    const double pi = 3.1415926;

    double r, h;
    double l, s, sq, vq, vz;

    printf("请输入半径与高\n");
    scanf("%lf %lf", &r, &h);

    l=2*pi*r;
    s=pi*r*r;
    sq=4*s;
    vq=3.0/4*s*r;
    vz=s*h;
    printf("圆周长为：\t%6.2f\n", l);
    printf("圆面积为：\t%6.2f\n", s);
    printf("圆球表面积为：\t%6.2f\n", sq);
    printf("圆球体积为：\t%6.2f\n", vq);
    printf("圆柱体积为：\t%6.2f\n", vz);

}
```

图 3　无命名规则的实例

程序代码 2.b

```
#include <stdio.h>
void main(void)
{
    const double pi = 3.1415926;

    double ban_jing, gao;
    double zhou_chang, mian_ji;
    double biao_mian_ji, qiu_ti_ji, zhu_ti_ji;

    printf("请输入半径与高\n");
    scanf("%lf %lf", &ban_jing, &gao);

    zhou_chang = 2 * pi * ban_jing;
    mian_ji = pi * ban_jing * ban_jing;
    biao_mian_ji = 4 * mian_ji;
    qiu_ti_ji = 3.0/4 * mian_ji * ban_jing;
    zhu_ti_ji = mian_ji * gao;

    printf("圆周长为：\t%6.2f\n", zhou_chang);
    printf("圆面积为：\t%6.2f\n", mian_ji);
    printf("圆球表面积为：\t%6.2f\n", biao_mian_ji);
    printf("圆球体积为：\t%6.2f\n", qiu_ti_ji);
    printf("圆柱体积为：\t%6.2f\n", zhu_ti_ji);
}
```

图 4　UNIX 方式的拼音命名规则实例

2.3　程序编写与健壮性

学生编写程序往往更多考虑了程序执行结果是否正确，对于异常数据、规范数据以外的输入等情况考虑不足。本文举了一个例子，即使用 const 增强程序健壮性。如果输入参数采用“指针传递”，那么加 const 修饰可以防止意外地改动该指针，从而起到保护作用。例如，一个字符串拷贝函数 str_copy：void str_copy(char *str_destination, const char *str_source); 其中 str_source 是输入参数，str_destination 是输出参数。给 str_source 加上 const 修饰后，如果函数体内的语句试图改动 str_source 的内容，编译器将指示错误，程序代码 3.a 如图 5 所示。

程序代码 3.a

```
void str_copy(char *str_destination, const char *str_source);
```

图 5　使用 const 增强健壮性的实例

以正切函数的编写为例，进一步说明健壮性的重要性。学生编写好程序后，在没有 assert 语句时，假定输入邻边长为 0，显然程序结果将会出错。这就使学生意识到应当避免“输入错误数据，输出垃圾数据”这种现象的发生。从而启发学生处理这种情形时，增加 assert 语句，一旦发生异常，则会报告错误。这个 assert 在同名的函数库中，一般 C 语言教学中并不提及。但笔者认为直接将其运用到程序当中是十分有益的，能够使学生认识到细致考虑问题对于程序员的重要性。函数代码为程序代码 3.b（见图 6）。代码中以 const 定义处理的精度，也是健壮性的一种方式，避免了程序意外修改 precision 的可能。

程序代码 3.b

```
#include <assert.h>
#include <math.h>
const double precision = 1e-6;     /*替代#define 方式*/
double tan_fun(double opposite_len, double adj_len)
{
        assert(fabs(adj_len) > precision);   /*确保被除数不为零*/
        return(opposite_len / adj_len);
}
```

图 6　使用 assert 增强健壮性的实例

在文件操作过程中，文件打开是否成功的检测也是健壮性的一个很好例子。如果不进行异常检测，程序将出现不可预知的错误。因此，可以加入判断文件是否正常打开的检测机制。程序代码 3.c（见图 7）对比了读取一个文件时，有或无检测打开文件的两种方式的例子，将它们放在一个 main()函数中，只为比较用。

程序代码 3.c

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
        /*文件打开失败而出错*/
        FILE *fp_norobust;
        fp_norobust = fopen("robust.dat", "r");
        fclose(fp_norobust);

        /*文件打开失败正常退出*/
        FILE *fp_robust;
        if ((fp_robust = fopen("robust.dat", "r")) == NULL)
        {
                printf("cannot open file.\n");
                exit(0);
        }
}
```

图 7　文件打开检测健壮性的实例

2.4　程序设计思路比较

初学 C 语言程序设计的学生，能够读懂一般的 C 程序代码。但是，学生自己动手完成从设计思路到编写代码的过程却不易。学生变成了记忆代码的机器，复制教材、习题集中的“标准答案”。当然，最初的记忆是有益和必要的，但长此以往，学生的创新能力、自主能力、编程能力将逐渐退化，这对课程的学习大为不利。

因此，这就需要从观念与方法上进行改变。一方面，向学生传递一个概念：“条条大路通罗马”。由于分析问题时的出发点与考虑的角度不同，对同一问题可以有多种可行有效的解答。不需要、也不应当去刻意记住所谓的“标准答案”。

程序设计思路的形成是抽象的，有时甚至难以用自然语言表达。那么，如何启发学生自觉

思考，并抓住问题求解的一般思路与方法呢？笔者认为分析比较同一问题的多种求解思路与代码是一种有效的方法。这如同高中数学、物理等课程中的一题多解，程序设计的一题多解也能有效促进学生求解问题思路的形成。事实上，很多教材中关于 for 语句、while 语句、do...while 语句等循环语句的讲解，采用同一问题多种语句的表达方式的方法，说明不同语句的特点。但是，这还不够。

这里举两个例子，一个是教材第一章中的题目。这个题目要求打印 3 行内容，为了说明清楚，将打印的内容加入论文，程序代码 4.a 如图 8 所示。这个问题，在讲解完第一章作业时，同学们会有一个简单的解答，即直接输出三行内容。同学们在学完循环语句后，可以思考如何将打印的星号换成其他的内容，如何改变打印字符的长度，并尝试用新的思路求解老问题。最后以程序代码 4.b（见图 9）的代码请学生说明编写的思路。

程序代码 4.a

```
************************
        Very Good!
************************
```

图 8　需要打印输出的内容

程序代码 4.b

```
#include <stdio.h>
#include <string.h>

const int N = 24;
const char ch[] = "*";
const char str[] = "Very Good!";
void main()
{
    int i;
    for (i = 0; i != N; i++)
    {
        printf(ch);
    }
    printf("\n");

    for (i = 0; i != (int)(N-strlen(str))/2; i++)
    {
        printf(" ");
    }
    printf(str);
    printf("\n");

    for (i = 0; i != N; i++)
    {
        printf(ch);
    }
    printf("\n");
}
```

图 9　简单问题的多种思路求解实例

第二个问题是统计一个字符串中数字、大小写字母、其他字符的个数。学生一般能够给出类似于程序代码 4.c（见图 10）的解答。对于库函数的使用，他们较为生疏，不能很好利用库函数中已经有的函数。通过程序代码 4.d（见图 11），可以使同学们认识到库函数对于简化程序设计的重要性。另外，for 语句的使用也使得学生对于循环的使用有了更深的体会。

程序代码 4.c

```
#include <stdio.h>
void main(void)
{
    const int NUM = 100;
    char str[NUM];
    printf("input a string:\n");
    gets(str);

    int digits = 0;
    int lower_case = 0;
    int upper_case = 0;
    int other = 0;

    int i = 0;
    while(str[i] != '\0')
    {
    if ('0' <= str[i] && str[i] <= '9')
        {
            digits++;
        }
        else if ('a' <= str[i] && str[i] <= 'z')
        {
            lower_case++;
        }
        else if ('A' <= str[i] && str[i] <= 'Z')
        {
            upper_case++;
        }
        else
        {
            other++;
        }
        i++;
    }

    printf("%d digits, %d lowercase letters, \
        %d uppercase letters, and %d other \
        characters in your sentence.\n", \
        digits, lower_case, upper_case, other);
}
```

图 10　统计字符求解实例 1

程序代码 4.d

```
#include <ctype.h>
#include <stdio.h>
void main(void)
{
    const int NUM = 100;
    char str[NUM];
    printf("input a string:\n");
    gets(str);

    int digits = 0;
    int lower_case = 0;
    int upper_case = 0;
    int other = 0;

    int i = 0;

    for (i = 0; str[i] != '\0'; i++)
    {
        if (isdigit(str[i]))
        {
            digits++;
        }
        else if (islower(str[i]))
        {
            lower_case++;
        }
        else if (isupper(str[i]))
        {
            upper_case++;
        }
        else
        {
            other++;
        }
        i++;
    }
    printf("%d digits, %d lowercase letters, \
        %d uppercase letters, and %d other \
        characters in your sentence.\n", \
        digits, lower_case, upper_case, other);
}
```

图 11　统计字符求解实例 2

3　结束语

C 高级语言程序设计的第一要素仍然是实践。虽然教师在教学中强调的问题、讲授的方法等对于学生的学习有很大的影响。但是，不通过自己动手、思考、上机实践是无法学会 C 语言程序设计这门课程的。即便学生的考分合格，也只是纸上谈兵。因此，鼓励学生动手、上机、甚至访问国外的程序竞赛网站（如 TopCoder 等）以及鼓励优秀学生参加 ACM 程序设计竞赛等，这些都是十分有益而必要的课堂补充。

参 考 文 献

[1] 谭浩强. C 高级语言程序设计[M]. 2 版. 清华大学出版社，2006.

[2] 陈良银，游洪跃，李旭伟. C 语言程序设计[M]：C99 版. 清华大学出版社, 2006.

[3] Brian W K, Dennis M R. The C Programming Language[M] (2nd edition, 1988). 机械工业出版社，2006.

[4] 潘京. 用比较法进行文科 C 语言程序设计教学. 中央民族大学学报[N]，Vol. 14, No. 2, 2005，5.

[5] TopCoder. http://www.topcoder.com/.