HZ BOOKS
华章科技

创新性地从程序“错误”的角度出发进行讲授，通过对比程序的对错、程序的好坏，使读者具备编制高质量的程序的能力。

从对比程序的对错到对比程序的质量，循序渐进地引领读者成为编程高手。

华章原创精品

C语言解惑

C Language Demystified

刘振安 刘燕君 编著

机械工业出版社
China Machine Press

华章原创精品

C语言解惑

C Language Demystified

刘振安 刘燕君 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

C 语言解惑 / 刘振安，刘燕君编著 . —北京：机械工业出版社，2014.10

ISBN 978-7-111-47985-7

I. C… II. ①刘… ②刘… III. C 语言 - 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2014）第 214927 号

本书分为两篇共 25 章。第一篇共 11 章，重点分析编程中存在的典型错误，通过对比正确与错误的程序，使读者加深印象，尽快掌握 C 语言编程基础知识并提高编程能力。第二篇共 14 章，除保持第一篇的特点之外，重点关注能运行而编程质量不好的程序，通过寻找“好”的替代程序，引导读者提高实用编程能力。本书选用案例讲授，争取起到雨打沙滩、滴滴入骨的效果。

本书涉及的内容深浅均有，其中不乏编程高手也会产生混淆的内容，各类人群都能在其中找到满足自己需要的知识并有一定收获。本书不仅对社会读者极有参考价值，还能帮助在校生进行课程设计训练，完成毕业实习或毕业论文。本书既可以作为手册随时查阅，又可以作为自学或培训班的参考资料。

C 语言解惑

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：李 艺 责任校对：董纪丽
印 刷：三河市宏图印务有限公司 版 次：2014 年 10 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：33
书 号：ISBN 978-7-111-47985-7 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

Preface 前 言

C 语言编程仍然是编程工作者必备的技能。目前有四类典型的学习 C 语言的教材：第一类是以讲授语法为主线，即流行的教科书方式，所涉及的例题均以正确的程序为主；第二类是以案例教学为主的教材，摆脱了语法的部分约束；第三类是以讲解编程技术为主的经验之作，主要针对已有编程基础的读者；第四类是针对编程容易产生错误的专题，对比正确与错误的程序以提高编程能力，涉及的内容比较专业。这些教材各有千秋，其共同的目的都是想教会读者如何编写正确、规范的程序。我们也曾在两部教材的每一章中尝试增加一节错误分析的内容，以期让读者通过识别错误提高编程的能力。虽然反响不错，但教材仍受语法和教学大纲的约束，所涉及的深度和广度均受到限制。

其实，通过比较编程中存在的典型错误，能给人深刻的印象，就像雨珠打在久旱的沙滩上——滴滴入骨，使学习者更容易记住编程的要诀。通过演示如何将一个能运行的程序优化为更好、更可靠的程序，能使读者建立好的编程风格并提高编程质量。因为摆脱了教学大纲的约束，所以能把重点放在学习识别正确与错误及提高编程质量的方法上。基于这一思路，我们编写了本书。它不是学校的教材，但能更好地为初学者打开启蒙之路；它不是纯技术书籍，但能为编程者指出进修之路；它并不面面俱到，但确能起到编程手册的作用。因此，它可以作为编程人员的常备参考书。

本书共分两篇 25 章。第一篇是 C 语言编程中的对与错，主要采用分析编程中存在的典型错误、对比正确与错误程序的方法，使读者加深印象并提高分辨语法对错及编程的能力，进而达到尽快掌握 C 语言编程基础知识的目的。

第一篇共 11 章，包括第 1 章至第 11 章。第 1 章主要涉及刚接触 C 语言易犯的错误。第 2 章通过分析输入、输出语句中的错误，介绍 printf 和 scanf 的使用技巧。第 3 章中的基本数据类型是编程最基础的知识，目的是尽快建立程序，正确使用数据和运算符。第 4 章

中的控制语句是编程的基本功之一，其错误也是五花八门，必须十分小心。第 5 章关注数组与指针，开始接触构造类型的错误。第 6 章给出编写函数的典型错误。第 7 章分析自定义宏时最容易出现的错误。第 8 章除了分析使用库函数的典型错误之外，增加了 printf 的功能，目的是使读者充分利用 printf 函数。第 9 章主要是结构的基本使用方法。第 10 章通过实例分析联合与枚举的正确使用方法。为了适应实际编程，第 11 章增加了利用状态机编程的基础知识。

第二篇是 C 语言编程中的好与坏，这里“坏”的含义是指编程质量差的程序。本篇继续运用第一篇分析对与错的方法，但主要是针对能运行而编程质量不好的程序，寻找质量“好”的替代质量“差”的，从而提高实用编程能力。

第二篇共 14 章，包括第 12 章至第 25 章。第 12 章介绍编译系统的差别，主要目的是利用编译系统预报尽可能多的错误。第 13 章结合实例介绍调试与测试程序的各种典型方法，包括自定义宏、使用系统提供的调试函数、编写自己的调试函数和利用条件编译等技术。第 14 章介绍大端存储和小端存储的概念及变量的存储地址分配，通过对比分析，介绍如何更好地使用各种基本变量、常量和指针。第 15 章重点是正确定义带参数的宏及宏函数。第 16 章重点是如何设计可靠、正确的控制语句，如何正确选择运算符、优先级和求值顺序。第 17 章除分析位运算容易用错之处外，还给出使用位运算的典型例子。第 18 章重点是如何用好数组与指针。第 19 章是如何更好地编写函数，包括解读函数声明的方法。第 20 章重点介绍可变参数的函数的设计方法及 printf 函数、scanf 函数和 sscanf 函数的原型。第 21 章是如何在不同场合下正确地使用结构，并讨论优先使用结构指针传递参数的原因。第 22 章是预防使用文件的常见错误。第 23 章结合实例讨论多文件编程错误、单文件结构、一个源文件和一个头文件的结构以及多文件结构。第 24 章介绍调试版本和发布版本的区别。第 25 章列举 7 个方面的问题，介绍编程优化的典型思路和方法。

本书涉及的内容深浅均有，其中不乏编程高手也会产生混淆的内容，各类人群都能在其中找到满足自己需要的知识并有一定收获。本书不仅对社会读者极有参考价值，还能帮助在校生进行课程设计训练，完成毕业实习或毕业论文。本书既可以作为手册随时查阅，又可以作为自学或培训班的教材。

因为本书不是教材，所以多个作者分别撰写各章的不同小节，然后逐章讨论并独立成章。刘燕君主要负责第 1 ～ 8 章和第 19 ～ 25 章，刘振安负责第 9 ～ 18 章，最后由刘振安统稿。为本书编写工作提供帮助的还有周淞梅实验师、苏仕华副教授、鲍运律教授、刘大路博士、唐军高级工程师等。

在编写过程中，本书得到中国科学院院士、中国技术大学陈国良教授的大力支持，特此表示感谢！刘燕君老师在中国台湾的两年博士后期间，得到了张真诚教授和黄明祥教授的大力支持和帮助，特此感谢。对被引用资料的作者及网络作品的作者表示衷心感谢！

为了学习方便，本书提供全部程序代码，可从华章网站（www.hzbook.com）下载或通过电子邮件联系编者索取：zaliu@ustc.edu.cn。

编者

2014 年 7 月

目 录 *Contents*

前 言

第一篇 C语言编程中的对与错

第1章 初涉C语言者的困惑……2

1.1 中文字符以假乱真……2
1.2 象形字体扰乱视听……3
1.3 都是注释惹的祸……4
1.4 千万不要忘记我……4
1.5 别把分号放错地方……4
1.6 少了花括号就是行不通……6
1.7 scanf要“&”不要“\n”……6
1.8 老大就是要在最前面……6
1.9 记住我就会受益无穷……7

第2章 用好printf和scanf一对活宝……9

2.1 printf输出的小奥妙……9
2.2 printf输出整数或字符……11
2.3 输入的格式配对错误……12
2.4 空格让scanf莫名其妙……14
2.5 回车键打乱scanf的阵脚……15
2.6 字符输入要搞特殊化……15
2.7 别混淆字符数组和字符……17
2.8 一维数组更要特殊对待……19
2.9 输出值的操作符……20
2.10 引入指针更方便……23
2.11 指针的困惑……24

第3章 基本数据类型……26

3.1 混合运算要小心……26
3.2 数据类型的后缀符号……28
3.3 基本数据的初始化……28
3.4 注意编译系统的差别……29
3.5 不要用错等于运算符……30
3.6 不要用错逗号运算符……31

第4章 程序控制语句……34

4.1 控制流程运算容易出现的问题……34
4.1.1 写错关系运算符……34
4.1.2 混淆表达式和关系表达式的值……36
4.1.3 混淆逻辑表达式和逻辑表达式的值……37

4.1.4 混淆逻辑运算符和位运算符 ·· 38
4.2 程序控制语句容易出现的问题 ···· 39
4.2.1 条件分支语句的错误 ····· 39
4.2.2 控制重复的分支语句 ····· 44
4.2.3 运算符优先级错误 ····· 53
4.2.4 求值顺序 ····· 55

第 5 章 数组与指针是重点 ····· 58

5.1 一维数组越界和初始化错误 ····· 58
5.1.1 一维数组越界错误 ····· 58
5.1.2 一维数组初始化错误 ····· 60
5.2 数组赋值错误 ····· 61
5.3 指针地址的有效性 ····· 64
5.4 配合使用一维数组与指针 ····· 69
5.4.1 使用一维数组名简化操作 ····· 69
5.4.2 使用指针操作一维数组 ····· 71
5.4.3 使用一维字符数组 ····· 78
5.4.4 不要忘记指针初始化 ····· 79
5.5 多维数组与指针 ····· 81
5.5.1 数组操作及越界和初始化错误 ····· 81
5.5.2 二维数组与指针 ····· 85
5.5.3 二维数组与指向一维数组的指针 ····· 89

第 6 章 函数是核心 ····· 91

6.1 函数的声明与定义 ····· 91
6.2 函数变量的作用域 ····· 93
6.3 函数变量类型的匹配 ····· 95
6.4 函数的返回路径 ····· 95
6.5 函数参数的设计及传递 ····· 96
6.6 传递指针不一定改变原来参数的值 ····· 98
6.7 函数的返回值 ····· 99
6.7.1 无返回值的 void 类型函数 ····· 100
6.7.2 函数返回值问题 ····· 102

第 7 章 宏与 const ····· 106

7.1 用 const 代替无参数的宏定义 ··· 106
7.2 有参数的宏定义 ····· 109

第 8 章 库函数 ····· 112

8.1 引用的库函数与头文件不匹配 ··· 112
8.2 与库函数的参数类型不匹配 ····· 113
8.3 对库函数的作用理解不对 ····· 114
8.4 充分利用库函数 printf 的功能 ··· 118
8.4.1 printf 的函数原型 ····· 119
8.4.2 printf 函数的格式控制符 ····· 120

第 9 章 结构 ····· 134

9.1 结构定义和赋值错误 ····· 134
9.2 结构作为函数参数及函数的返回值 ····· 140
9.3 使用结构数组和指针容易出现的错误 ····· 145
9.4 其他注意事项 ····· 147

第 10 章 联合与枚举 ····· 148

10.1 联合 ····· 148
10.2 枚举 ····· 153

第 11 章 状态机 ····· 159

第二篇 C语言编程中的好与坏

第12章 注意编译系统的差别 …… 170

12.1 打开所有编译开关 …… 170
12.2 克服依靠编译系统产生的错误 …… 170

第13章 测试与调试程序 …… 174

13.1 预防措施 …… 174
13.1.1 书写格式和注意事项 …… 174
13.1.2 命名注意事项 …… 176
13.1.3 程序注释 …… 178
13.2 使用条件编译 …… 180
13.3 消灭警告信息 …… 183
13.4 使用简单的输出信息调试程序 …… 185
13.5 编写 error 函数 …… 194
13.6 使用集成环境提供的调试手段 …… 197
13.6.1 一个简单的实例 …… 197
13.6.2 编译程序 …… 198
13.6.3 排错 …… 199
13.6.4 基本调试命令简介 …… 200
13.6.5 程序与汇编调试窗口 …… 203
13.7 调试程序实例 …… 204
13.8 软件测试 …… 206
13.8.1 模块测试 …… 209
13.8.2 组装测试 …… 211
13.8.3 确认测试 …… 211
13.9 程序的测试与调试 …… 212
13.10 测试用例设计技术 …… 214
13.10.1 逻辑覆盖法 …… 215
13.10.2 等价划分法 …… 218
13.10.3 边值分析法 …… 219
13.10.4 因果图法 …… 219
13.10.5 错误猜测法 …… 220

第14章 正确使用变量、常量和指针 …… 221

14.1 基本数据类型的变量初始化 …… 221
14.2 不要混淆字符和字符串 …… 222
14.3 指针的初始化 …… 224
14.4 指针相等 …… 228
14.5 使用 const …… 232
14.5.1 左值和右值 …… 232
14.5.2 推荐使用 const 定义常量 …… 234
14.5.3 对函数传递参数使用 const 限定符 …… 236
14.5.4 对指针使用 const 限定符 …… 237
14.6 使用 volatile 变量 …… 240
14.7 变量的存储地址分配 …… 242

第15章 正确使用宏 …… 246

15.1 不要使用不存在的运算符 …… 246
15.2 正确使用定义的宏 …… 247
15.3 正确定义宏的参数 …… 248
15.4 使用宏定义函数 …… 250

第16章 控制语句 …… 252

16.1 运算顺序错误 …… 252
16.2 采用更明确的条件 …… 254
16.3 设计存在的问题 …… 256
16.3.1 没有涵盖全部条件 …… 256
16.3.2 条件超出范围 …… 261
16.3.3 减少循环次数 …… 264
16.4 正确选择运算符 …… 273

16.5 优先级和求值顺序错误 ····· 277

第 17 章 位运算 ····· 285

17.1 位运算典型错误 ····· 285
17.2 位运算典型实例 ····· 290

第 18 章 再论数组与指针 ····· 295

18.1 一维数值数组和指针 ····· 295
18.1.1 使用数组偏移量造成数组越界 ····· 295
18.1.2 使用数组名进行错误运算 ····· 296
18.1.3 错误使用数组下标和指向数组指针的下标 ····· 298
18.1.4 小结 ····· 299
18.2 一维字符数组和指针 ····· 301
18.2.1 字符数组的偏移量 ····· 301
18.2.2 字符数组不对称编程综合实例 ····· 303
18.3 动态内存 ····· 307
18.3.1 非数组的指针 ····· 307
18.3.2 NULL 指针 ····· 309
18.4 二维数组和指针 ····· 310
18.4.1 二维数组的界限 ····· 310
18.4.2 二维数组的一维特性 ····· 312
18.4.3 指向二维数组的指针 ····· 314
18.5 数组和指针应用实例 ····· 318

第 19 章 再论函数 ····· 325

19.1 函数变量的作用域 ····· 325
19.1.1 块结构之间的变量屏蔽规则 ····· 325
19.1.2 程序和文件内的变量 ····· 328
19.1.3 多文件变量作用域 ····· 331
19.2 函数的参数 ····· 337
19.2.1 完璧归赵 ····· 338
19.2.2 多余的参数 ····· 340
19.2.3 传递的参数与函数参数匹配问题 ····· 342
19.2.4 等效替换参数 ····· 345
19.3 函数的类型和返回值 ····· 347
19.3.1 函数的类型力求简单 ····· 347
19.3.2 实参要与函数形参的类型匹配 ····· 349
19.3.3 正确设计函数的返回方式 ····· 351
19.3.4 正确设计和使用函数指针 ····· 356
19.3.5 如何解读函数声明 ····· 361

第 20 章 再论库函数 ····· 365

20.1 getchar 函数的返回类型不是字符 ····· 365
20.2 setbuf 函数与其他函数的配合 ····· 368
20.3 错误使用 errno 函数 ····· 377
20.4 模拟设计 printf 函数 ····· 379
20.4.1 具有可变参数的函数 ····· 379
20.4.2 设计简单的打印函数 ····· 382
20.4.3 利用宏改进打印函数 ····· 387
20.5 scanf 和 sscanf 函数 ····· 392
20.5.1 sscanf 函数的使用方法 ····· 394
20.5.2 sscanf 函数用法举例 ····· 395
20.6 探讨 printf 函数 ····· 398

第 21 章　再论结构……400

21.1　同类型结构变量之间的整体赋值……400

21.2　使用键盘赋值……405

21.2.1　为结构变量赋值……405

21.2.2　为结构指针变量赋值……407

21.2.3　为链表赋值……411

21.2.4　为结构数组的变量赋值……412

21.2.5　为含有指针域的结构数组赋值……413

21.3　使用结构作为函数的参数……417

21.3.1　结构变量的传数值与传地址值……417

21.3.2　结构数组传地址值……418

21.4　结构函数的返回值……421

21.5　修改传递的结构参数的值……430

21.6　优先使用结构指针传递参数……435

第 22 章　使用文件常见错误分析……439

22.1　文件的打开与关闭……439

22.2　文件的读写……450

22.3　其他读写函数……464

22.4　文件的定位……467

22.5　操作出错检测及错误标志的复位……469

22.6　文件的内存分配……470

22.7　小结……470

第 23 章　多文件编程……472

23.1　多文件编程错误浅析……472

23.2　单文件结构……475

23.3　一个源文件和一个头文件……475

23.4　多文件结构……477

第 24 章　发布 C 程序……483

24.1　两种版本的异同……483

24.2　两种版本的设置……484

第 25 章　典型问题……486

25.1　计算机解题具有多解的特点……486

25.2　应对算法进行优化……487

25.3　优化时要避免出现新的错误……488

25.4　扩展程序要注意是否满足全部条件……494

25.5　注意函数设计的多样化和效率……496

25.6　使用多文件编程……502

25.7　使用状态机设计程序……507

附录 A　C 语言操作符的优先级……513

附录 B　简化优先级记忆口诀……515

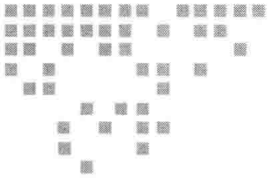
附录 C　7 位 ASCII 代码表……517

主要参考文献……518

第一篇 Part 1

C 语言编程中的对与错

本篇主要采用分析编程中存在的典型错误、对比正确与错误程序的方法，使读者加深印象并提高分辨语法对错及编程的能力，进而达到尽快掌握 C 语言编程基础知识的目的。



Chapter 1 第 1 章

初涉 C 语言者的困惑

初学 C 语言首先要建立良好的习惯并克服最常见的错误。

1.1 中文字符以假乱真

不管是初学者还是有经验的程序员，都会碰到这个问题。这往往是在拼音状态下输入标点符号之类的字符造成的。假设语句

```
printf("%s", "OK");
```

中的“，”号是中文字符，编译器会给出如下信息：

```
error C2018: unknown character '0xa3'
error C2018: unknown character '0xac'
```

只要给出“0xa”的标识，就可断定该行存在中文字符。

只要稍微注意一下就可以避免这个错误。其实，多数的错误不是在输入程序时误输入，而是直接将 Word 文档里或网上的程序拷贝到源文件中造成的。一般是因为整理文档里的程序时，人为地使用中文字符或插入图形符号，例如 &、[、] 、& 、# 、＼等符号。

只要编译系统给出出错信息并定位到所在行，就很容易判断出错误。有点难度的是中文空格，这个空格一般有三种情况：一行的首、尾和其他位置。假设下面程序除第 1 行之外，其他行的首尾均有空格，看看这类错误的表现形式。

```
#include <stdio.h>
void main( )
{
        printf("%s\n","OK");    // 打印输出
}
```

当编译给出第二行有错误的信息时，可以把鼠标光标放到第一行的尾部，按一下“↓”键，这时光标移到下一行并停在离“)”的一段距离处，这说明光标左边有中文空格。用 Backspace 键删除前面的空格，删到“)”处即可。如果将光标放在“#”处，按一下“↓”键，光标会停在离字母“v”的一段距离处，使用删除键删除右边的空格即可。

其他各行同样处理，对于第 4 行，如果“//”号的字体不是绿色的，说明注释语句之前有空格，注释不起作用，往左边删除，直到“//”号变为绿色。

需要注意的是，编译拷贝的程序时，可能会给出很多错误信息，而且可能给出的错误种类也很多。如果第 1 个错误就有“0xa”的标识，则一定要先解决它。有时解决它之后，其他的错误可能就没有了。

1.2　象形字体扰乱视听

要特别注意形状相近的字母，最典型的是小写字母“l”和数字“1”。以下面的程序为例。

【例 1.1】演示混淆字母“l”和数字“1”的错误程序。

```
#include <stdio.h>
void main()
{
    double x=0;
    printf(" 输入 x: ");
    scanf("%lf",&x);
    printf(" 输出 x:%f\n",x);
}
```

问题是编译系统判断不出这类问题，虽然程序编译正确，但运行结果却是错误的。本书约定使用下划线标注输入并以回车键结束，以后不再赘述。下面是演示示范。

```
输入 x: 7.8
输出: 0.000000
```

错误的原因是“lf”中的小写字母“l”错为数字“1”。为了预防这种错误，可以使用大写字母 L，即

```
scanf("%Lf",&x);
```

C 语言是对大小写敏感的，但对 printf 格式符号“F”和“f”、“L”和“l”等是不分大小写的，利用这个特点，既能预防这类笔误，又能提高可读性。

1.3 都是注释惹的祸

注释语句可以增加可读性，但编译系统检查不出不正确的注释，所以会导致错误的结果。

【例 1.2】演示注释错误的程序。

```
#include <stdio.h>
void main()
{
     double x=0;
     printf(" 输入 x: "); /* 给出提示
     scanf("%Lf",&x);  /*  输入信息 */
     printf(" 输出 x:%f\n",x);
}
```

编译正确，但运行后直接输出“输入 x ： 输出 x:0.000000”的错误结果。注释“/*”和“*/”必须配对出现。程序中的第 1 个注释漏掉配对的“*/”号，所以将输入语句屏蔽。

由此可见，若右边的注释符号“*/”错成“/*”或遗漏，而后面又有注释，就可能会使许多行程序变成注释，影响运行结果。

对 Visual C 而言，为了避免这个问题，可以使用与 C++ 兼容的行注释符号“//”。

1.4 千万不要忘记我

【例 1.3】演示没有包含头文件 stdio.h 的例子。

```
void main()
{
     double x=0;
     printf(" 输入 x: ");
     scanf("%Lf",&x);
     printf(" 输出 x:%f\n",x);
}
```

因为没在 main 语句之前使用“#include <stdio.h>”语句，所以编译给出如下警告信息。

```
warning C4013: 'printf' undefined; assuming extern returning int
warning C4013: 'scanf' undefined; assuming extern returning int
```

输入、输出函数是定义在头文件“stdio.h”中的（目前暂不深入讨论，只要知道正确的使用方法就可以了），不要忘记使用这条包含语句。

1.5 别把分号放错地方

分号“;”并不总是出现在语句的尾部。如下的写法

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

虽然可以通过编译并能正确运行，但会出现如下警告信息：

```
warning C4067: unexpected tokens following preprocessor directive - expected a
newline
```

C语言标准规定一行可以有多条语句，例如：

```
int a; double d; char c;
```

但输入输出不是C语言的一部分，而是以标准函数形式提供。在每个引用库函数的源程序文件的开头处必须含有如下一行。

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

文件stdio.h定义了I/O库所用的某些宏和变量，使用#include语句把它包含进来，一起编译。虽然有的C编译器使用scanf和printf函数不需要包含它，但建议养成使用这条语句的习惯。其实，一条预编译语句是以换行作为结束的，也就是说，一行只能书写一条预编译语句，如果书写两条，也会给出如上警告。

包含语句属于预编译语句，“；”号作为语句结束符用在一条程序语句之后，而包含语句不是程序语句，它不是以“；”号作为结束符。这里多出一个符号，编译系统认为你应该从“；”号处换行，以便保证预编译语句正确，所以给出警告信息。这与语句漏掉“;”号不同，如果语句尾部漏掉“;”号，就不是给出警告信息，而是给出出错信息。例如：

```
printf("输入x: ")
scanf("%Lf",&x);
```

会给出如下出错信息：

```
error C2146: syntax error : missing ';' before identifier 'scanf'
```

这条信息明确指出在scanf语句之前漏掉分号，也就是printf少了语句结束符“;”。

如果程序中多用了“;”号，则“;”号构成跳空语句，即

```
printf("输入x: ");;
```

相当于

```
printf("输入x: ");
;
```

两条语句，第2行的“;”语句什么也不做。

由此可知，包含头文件的语句没有“;”号，其他语句必须以“；”结束。当然，主函数不是语句，它的“)”号之后更不能有分号。

关于函数和判别语句，暂不讨论。对于入门，目前掌握这些就足够了。

1.6 少了花括号就是行不通

花括号是成对出现的，遗漏花括号会改变程序的运行方式。一般来讲，人们更容易忘记右边作为程序结束的花括号。目前要记住：主程序以“{”号开始，“}”号结束。记住下面的格式是有益处的。

```
void main()
{     // 开始
      // 程序体
}     // 结束
```

为了避免这个问题，可以在编程需要出现一对“{}”号的地方，直接给出一对“{}”号，然后在“{}”号里面添加程序。

1.7 scanf要“&”不要“\n”

scanf语句中的变量前面应加上“&”号，如果少了“&”号，能编译通过，但运行会出错。同理，如果格式符中多了“\n”号，编译系统也不能查出错误。

【例1.4】演示scanf语句多了“\n”号的错误。

```
#include <stdio.h>
void main()
{
     double x=0;
     scanf("%Lf\n",&x);
     printf("输出x:%Lf\n",x);
}
```

编译系统不能查出scanf语句使用的错误，如果运行这个程序，在接收一个数据之后，并不继续运行。只有再随便输入一个数据，它才会继续运行下去。下面是一个运行示范。

```
45.6
67.8
输出x:45.600000
```

程序需要再输入一次才能继续运行下去，程序的运行无疑不合要求。

1.8 老大就是要在最前面

【例1.5】分析下面的程序错在哪里。

```
#include <stdio.h>
void main()
{
     printf("输入n: ");
```