谭浩强《C 程序设计》（第四版）错误不完全汇集

前言

p12

① 数据类型介绍中，增加了 C99 扩充的双长整型(long long int)、复数浮点型(float complex,double complex ,long long complex)、布尔型(bool)等，使读者有所了解。
② C99 要求，main 函数的类型一律指定为 int 型，并在函数的末尾加一个返回语句“return 0;”。

评：
long long complex，bool 根本是子虚乌有的。数据类型都整不明白，还谈什么语言、算法呢？
C99 并没有要求 main 函数的类型一律指定为 int 型
main 函数的类型一律指定为 int 型其实是 C89 要求的

p16

“位运算”是 C 语言区别于其他高级语言的一个重要特点。

不顾常识，胡说八道

第一章　程序设计和 C 语言

1.2 什么是计算机语言

p1

计算机发展的初期，一般计算机的指令长度为 16，

完全是在信口开河，压根没有的事情。

p2

符号语言……LD 代表“传送”等————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p2

LD 一般表示赋值

p2

高级语言……用到的语句和指令是用英文单词表示的————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p2

高级语言这个层次没有指令这个概念

语句也未必用单词表示，例如：“1 + 2 ； ”，这在 C 语言中是一个表达式语句，其中根本就没英文单词。再比如“;”，是一条空语句，同样也没有英文单词。

p3

C（系统描述语言）————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p3

这是把张飞当张良了。C 语言可以写操作系统，但和“系统描述语言”是两回事

p3

C++（支持面向对象程序设计的大型语言）————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p3

C++是一种支持多重编程范式的通用程序设计语言

1.3 C 语言的发展及其特点

p4

……增加了一些功能，尤其是 C++中的一些功能，命名为 ISO/IEC 9899：1999。

又在信口开河

p4

1983 年，美国国家标准协会（ANSI）成立了一个委员会，根据 C 语言问世以来各种版本对 C 语言的发展和扩充，制定了第一个 C 语言标准草案（‘83 ANSI C）

曾经的子虚乌有的“87 ANSI C”不见了，很好。但这个‘83 ANSI C 是哪来的呢？

p4

1990 年，国际标准化组织 ISO（International Standard Orgnization）接受 C89……

评：ISO 啥时候改名了？

我记得这个组织的正式名称是 International Organization for Standardization

难怪书这么“水”

ISO 都被山寨了

p5

C 语言的运算符包含的范围很广泛，共有 34 种运算符(见附录 C)。

评：这个恐怕要麻烦老谭自己去亲自数数了。34 种究竟是怎么数出来的呢？再说附录 C 也不是关于运算符的啊

p5

C 语言把括号、赋值和强制类型转换等都作为运算符处理，从而使 C 语言的运算类型极其丰富，表达式类型多样化。

评：怎么就“从而使”了呢？哪来的这种因果关系？
“运算类型极其丰富”应该是“运算种类极其丰富”
“表达式类型多样化”，不知所云

p5

C 语言是完全模块化和结构化的语言。

评：什么叫“完全模块化”？什么叫完全结构化？这两个概念的定义是什么？

p5

例如，整型量与字符型数据以及逻辑型数据可以通用。

评：这是对初学者很严重的误导。另外“量”与“数据”并列，从小学生作文的角度也是说不过去的

p5

C 语言……是成功的系统描述语言……

评：请问是描述什么“系统”？如何“描述”？知不知道什么叫“系统描述语言”

p5

而且 C 编译系统在新的系统上运行时，可以直接编译“标准链接库”中的大部分功能，不需要修改源代码，因为标准链接库是用可移植的 C 语言写的。

评：C 编译系统在新的系统上运行：估计老谭在 linux 下成功地运行了他的 VC6
可移植的 C 语言：说实话，没见过这种 C 语言
可以直接编译“标准链接库”中的大部分功能：看不懂这是说什么哪

1.4 最简单的 C 语言程序

p6

int main()

这是一种过时的写法，不符合 C99 的精神。C99 只是为了兼容现有代码才允许这种写法

p6

而返回程序窗口

实际上恰恰是关闭程序窗口，返回 IDE 界面

p6

C99 建议把 main 函数指定为 int 型（整型）

其实是 C89 标准的规定，老谭大概是为以前不规范的 void main 找台阶

p6

在 main 函数中，在执行的最后设置一个“return 0;”语句。当主函数正常结束时，得到的函数值为 0，当执行 main 函数过程中出现异常或错误时，函数值为一个非 0 的整数。

评：写了“return 0;”居然还可能得到“一个非 0 的整数”？太神奇了吧

p7

文件后缀.h 的意思是头文件(head file),因为这些文件都是放在程序各文件模块开头的。

评：head 未必是以文件形式存在。
文件放在“文件模块开头”是莫名其妙的说法

p7

则表示从到本行结束是“注释”。

语文问题

p8

printf("sum is %d\n", sum); ……在执行 printf 函数时，将 sum 变量的值（以十进制整数表示）取代双撇号中的%d。

执行 printf 函数:怎么读怎么别扭，应该是调用 printf 函数
将 sum 变量的值（以十进制整数表示）取代双撇号中的%d：变量的值是取代不了%d 的

p9

第 2 行输出“大数为 8”。

输出的是“max=8”

p9

程序第 4 行是……

那是第 5 行

p9

执行 scanf 函数，从键盘读入两个整数，送到变量 a 和 b 的地址处，然后把这两个整数分别赋给变量 a 和变量 b。

哪来的什么“然后”？

p10

%d 由变量 c 的值取代之。

不是由“变量 c 的值”取代之

p10

例如可以把例 1.2 程序中的“int a,b,sum”放到 main 函数前面，这就是全局声明，在函数外面声明的变量称为全局变量

这是教唆，多数情况下是很糟糕的写法。
此外 C 没有全局变量这种说法，只有外部变量

p10

源积程序……

p11

①函数首部……包括函数名、函数类型、函数属性、函数参数（形式参数）名、参数类型。

“函数类型”：错误使用术语
“函数属性”：莫名其妙的说法

p11

函数体一般包括以下两部分。

- 声明部分
- 执行部分

作为一本号称“按照 C99 标准”的教科书来说，这种说法显然是错误的

p12

在每个数据声明和语句的最后必须有一个分号。分号是 C 语言的必要组成部分。

武断+无知

此外“数据声明”也很荒唐，C 语言没有这样的概念

p10~12

“1.4.2 C 语言程序的结构”这一小节下面的小标题是

> 1.4.2 C 语言程序的结构
> (1) 一个程序由一个或多个源程序文件组成。
> (2) 函数是 C 语言的主要组成部分。
> (3) 一个函数包括两个部分
> (4) 程序总是从 main 函数开始执行的
> (5) 程序中对计算机的操作是由函数中的 C 语句完成的。
> (6) 在每个数据声明和语句的最后必须有一个分号。
> (7) C 语言本身不提供输入输出语句。
> (8) 程序应当包括注释。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p10~12

仔细回味一下，很有喜感。
即使当作小学生作文，恐怕也不合格吧
讲 C 语言的程序结构，怎么冒出来“C 语言本身不提供输入输出语句”了呢？
程序结构和“程序总是从 main 函数开始执行的”又有什么关系呢
说老谭离题万里、云山雾罩不冤枉他吧

1.5 运行 C 程序的步骤与方法

p12

1.5 运行 C 程序的步骤与方法

这一小节讲的实际上是用 C 语言开发程序的过程与步骤（编辑、编译、链接及运行）
从内容上看，标题文不对题

这一小节的重大疏失是
只讲了编译和运行时发现错误应该返回修改源程序
但对链接时可能也会发生错误却只字未提
事实上链接错误也是编程最常见的错误之一
对这样常见的一大类错误怎么可以忽视呢

语言罗嗦和概念不清的毛病在这一小节同样存在
譬如

(1)上机输入和编辑源程序。通过键盘向计算机输入程序，……————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p12

什么叫“通过键盘向计算机输入程序”？难道这不属于“编辑”的范畴吗？
试问老谭，“输入“和”编辑”之间的区分到底应该如何界定？

p13

经过编译得到目标程序文件 f.obj,再将所有目标模块输入计算机————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p13

“将所有目标模块输入计算机”：老谭能否演示一下怎么输入？

p14

集成环境(IDE)————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p13

直接无视 D 字的含义

1.6 程序设计的任务

p14

> 1.6 程序设计的任务————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p14

这是第一章最糟糕的一个小节
首先是文不对题
程序设计的任务只能是解决问题，不可能是别的什么
但这一小节的内容其实是对完整的程序设计过程和步骤的描述
如果仅仅如此
问题并不严重
可是书中对完整的程序设计过程和步骤的描述居然是

> 从确定问题到最后完成任务，一般经历以下几个工作阶段：
> （1）问题分析。……
> （2）设计算法。……
> （3）编写程序。……
> （4）对源程序进行编辑、编译和连接。……
> （5）运行程序，分析结果。……
> （6）编写文档。……

这种“谭式”工作流程几乎是置几十年来人类程序设计的经验和思考于不顾，全然自创的
不过说实话，我倒更希望老谭的这一小节能找本象样的书抄一抄，因为那样至少不会荒谬的如此离谱

首先，轻飘飘的一句“从确定问题到最后完成任务”
把确定问题当成了轻而易举的一件事
事实上除了判断一个正整数是否是素数这样的问题
“确定问题”的难度常常并不亚于解决问题

其次，老谭的“问题分析”居然是

> 对于接手的任务要进行认真的分析，研究所给定的条件，分析最后应该到达的目标，找出解决问题的规律，选择解题的方法。在此过程中可以忽律一些次要的因素，使问题抽象化，例如用数学式子表示问题的内在特性。这就是建立模型。

我的印象，这不过是中学数学解题方法与数据建模方法的一种杂交而已，不伦不类四不象。
“对于接手的任务要进行认真的分析，研究所给定的条件”：空对空的废话
“分析最后应该到达的目标”，目标都不明就编程？
“找出解决问题的规律，选择解题的方法”，耳熟，很有亲切感。后来想起来中学数学老师总这么说
“在此过程中可以忽律一些次要的因素”：从胆量方面来说不得不赞一个。复杂的东西咱都

推还给客户好了
“使问题抽象化”：不知所云
“用数学式子表示问题的内在特性”，软件设计解决的问题有几个有数学式子？
“这就是建立模型”：不打自招

第三，“设计算法”中只字未提数据结构。老谭是强调“算法—程序的灵魂”的，可惜他一点不懂什么叫数据结构
在老谭的影响和带动下，经常看到一群整天把算法挂在嘴边但却连最基本的数据类型都搞不懂的家伙，根本就不能指望他们能懂得数据结构

第四，（3）的“编写程序”和（4）的“对源程序进行编辑”到底有什么分别

第五，（6）编写文档。程序开发完了，终于想起来“文档”了。任何一个软件工程专业的二年级本科生都能指出为什么这很荒谬。更荒谬的是，在老谭那里，“文档”居然不过是“程序说明书”、“帮助(help)”或“readme”。

习题

p15

这一章的习题也很糟糕

习题 6.编写一个 C 程序，输入 a，b，c 三个值，输出其中最大者。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p15

这道题不具备程序设计习题的最基本要求，是错误的题目。没有人可以做出这种题目。这反映了作者对软件开发基本上一窍不通

第二章　算法——程序的灵魂

p16

> 第二章 算法——程序的灵魂
> ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p16

这一章总的来说就如同阑尾，而且是发炎的阑尾。

p16

首先映入眼帘让人感到眼前一亮的是

> 著名计算机科学家沃思(Nikiklause Wirth)……
> ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p16

又来了。有人做过实验，google Nikiklause Wirth 搜来的全是中文网页。不要问我为什么，只有智商约等于 0 的人才会问这样的问题。
由此可见，垃圾也是会疯狂地繁殖的。有人总喜欢沾沾自喜地拿“1100 万”说事儿， 我倒想问一句，“1100 万”个错误是个什么概念，“1100 万”堆垃圾又会造成多大的污染。

p16

对于 Wirth 提出的

算法+数据结构=程序

老谭评论道：

> 直到今天，这个公式对于过程化程序来说依然是适用的。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p16

没有任何理由和依据，而敢于指责大师的公式具有局限性，不能不承认谭大师很有“勇气”。

接着，老谭提出

> 实际上，一个过程化的程序除了以上两个主要因素之外，还应当采用结构化程序设计方法进行程序设计，并且用某一种计算机语言表示。因此，算法、数据结构、程序设计方法和语言工具 4 个方面是一个程序设计人员所应具备的知识，在设计一个程序时要综合运用这几方面的知识。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p16

感慨：大师总是一针见骨地道出事物的本质；“伪大师”总是点金成铁狗尾续貂并且把水搅混。不妨再续狗尾：小学数学也是“程序设计人员所应具备的知识”。

2.1 什么是算法

p16~17

> 2.1 什么是算法————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p16

本小节前半部分介绍什么是算法，后半部分讨论了数值算法与非数值算法。主要毛病在后半部分。

后半部分实际上毫无必要；数值运算算法和非数值运算算法属于作者闭门造车自创的不规范用语（估计是不怎么读文献所导致）；

> 对各种数值运算都有比较成熟的算法可供使用————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p17

乃极其武断的信口开河，表明作者对数值计算领域惊人的无知。

2.2 简单的算法举例

p17

2.2 简单的算法举例

这一小节读起来犹如嚼棉花的感觉，只好快些扫过。

p18

例 2.1
…由于计算机是高速运算的自动机器，实现循环是轻而易举的……
————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p18

脑残是怎样炼成的，这种似是而非的“由于”句型功不可没。“循环”居然是由于“高速”，咄咄怪事。

p19~20

例 2.2
…从图 2.1 可以看出：“其他”这一部分，包括不能被 4 整除的年份，以及能被 4 整除，又能被 100 整除，但不能被 400 整除的那些年份（如 1900 年），它们都是非闰年。
————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p19

敢问大师有没有亲自看过图 2.1 中“其他”这部分写的究竟是什么？要是看过的话，我们可就知道什么叫睁着眼睛胡说了

p20

例 2.4
————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p20
其中描述的算法必然导致拖泥带水的代码。

p21

例 2.5……
所谓素数(prime)，是指除了 1 和该数本身之外，不能被其他任何整数整除的数
————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p21

厉害！把素数的定义都给改了，数学家们该感到汗颜了吧。
另外作者连“整数”和“正整数”的区别都搞不清楚

p21

> 判断一个数 n（n≥3）是否为素数……
> 实际上，n 不必被 2~n-1 的整数除，只须被 2~n/2 间的整数除即可，甚至只须被 2~n 之间的整数除既可。
> 例如，判断 13 是否为素数，只须将 13 被 2，3 除即可，如都除不尽，n 必为素数。
> ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p21

2~n/2 , 2~n 这两个区间到底哪个大？怎么后者竟然成了“甚至” 了呢

2.3 算法的特性

p21

> 2.3 算法的特性
> ……
> (2)确定性。算法中的每一个步骤都应当是确定的，而不不应当是含糊的、模棱两可的。…… 也就是说，算法的含义应当是唯一的，而不应当产生“歧义性”。所谓“歧义性”，是指可以被理解为两种（或多种）的可能含义。 ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p21

用老谭自己的矛攻一下他自己的盾
前一章的一道习题

> 6.编写一个 C 程序，输入 a，b，c 三个值，输出其中最大者。

这个题目本身是否“含糊”，是否具有“歧义性”？
问题本身就“模棱两可”，又何谈算法的“确定性”呢？

2.4 怎样表示一个算法

p22

> 2.4 怎样表示一个算法
> 流程图是用一些图框来表示各种操作。用图形表示算法，直观形象，易于理解。美国国家标准化委员会 ANSI(American National Standard Institute)规定了一些常用的流程图符号(见图 2.3)，已为世界各国程序工作者普遍采用。 ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p22

首先 ANSI 是 American National Standards Institute.
其次，老谭把 ANSI 搬出来是无知的卖弄。理由是：
这个标准有 ISO 版本，ANSI 采纳了这个 ISO 标准。而且对应于 ISO 标准，我国有自己的 GB。这些标准

的内容是一致的。老谭不谈 GB，不谈 ISO，却舍近求远地扯什么 ANSI 标准，又有什么特殊的理由呢？只能说他自己并不熟悉这些标准及其相互的关系。
而且书中的流程图根本不符合无论是 ISO 标准还是 GB 还是 ANSI 标准，流线乱窜，甚至全书中压根就没有标准用来表示循环的基本图形。
既然连这些标准读都没读过，把 ANSI 搬出来不是唬人又能是什么呢？

p26

> 2.4.3 三种基本结构和改进的流程图
> 1.传统流程图的弊端————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p26

成功地把滥用 goto 的危害移花接木地指误为传统流程图的弊端。

> ……人们规定出几种基本结构，然后由这些基本结构按一定规律组成一个算法结构（如同用一些基本预制件来搭成房屋一样），如果能做到这一点，算法的质量就能得到保证和提高。

说的象真的似的。C 语言再好也没耽误老谭写出那么多垃圾代码呀。由“基本结构”“按一定规律”就一定能保证算法的质量？！

p26

> 2.三种基本结构
> 1966 年，Bohra 和 Jacopini 提出了以下 3 种基本结构，用这 3 种基本结构作为表示一个良好算法的基本单元。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p26

首先把 Böhm 的名字错误地写成了 Bohra。

其次更严重的是 Böhm 和 Jacopini 根本就没有提出什么 3 种基本结构，更没有提出“用这 3 种基本结构作为表示一个良好算法的基本单元”。3 种基本结构在之前的高级语言中普遍存在。

Böhm 和 Jacopini 仅仅不很严格地证实了向前的 goto 语句可以用其他语句实现，他们的论文发表的时候，Dijkstra 反对 goto 语句的论文（1968）还没发表呢

p28

> 2.4.4 用 N-S 流程图表示算法————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p28

N-S 图废除了流线，哪里还是什么“流程图”？
事实上通用的说法是 Nassi–Shneiderman diagram (NSD)
老谭的一个毛病就是喜欢发明一些狗屁不通的新概念
比如二级指针，行指针，数值运算算法
这些都是他自己根本没学懂的表现

p29

例 2.15 ……
图 2.31————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p29

图 2.31 的算法其蠢无比
明明一次循环就可以解决非要用两次循环
明明一个变量就可以解决非要用一个数组
而且那个 50 基本没有用处
因为可以轻易写出人数为一正整数时的算法

三鹿往奶粉里掺加大量三聚氰胺使婴儿身体受到摧残而被判刑
老谭往教科书里掺加的无知、错误和愚蠢使初学者大脑变残为什么不被判刑？
我看刑法应该参照食品卫生法考虑一下这个问题
在教科书中传播愚蠢败坏民族的智力难道不是犯罪？

p31

2.4.5 用伪代码表示算法————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p31

两道例题都是用 for 循环可以很漂亮完成
却莫名其妙的选择了用罗嗦的 while 循环描述
误导初学者

p32

2.4.6 用计算机语言表示算法
……用计算机语言表示的算法是计算机能够执行的算法。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p32

胡扯，计算机能够执行只有机器语言

p33

例 2.19……

```
double deno=2.0,sum,term;
while(deno<=100)
{
……
deno=deno+1;
```

}————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p33

明显的错误，缺乏最基本的编程素质

2.5 结构化程序设计方法

p34

2.5 结构化程序设计方法

……关键是算法，有了正确的算法，用任何语言进行编码都不是什么困难的事情。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p34

这种对算法的片面强调是对初学者绝对的误导。误导的结果是，许多初学者往往只会用最拙劣数据结构绞尽脑汁地苦思复杂无比的“算法”。

此外，有经验的人都知道，即使懂得了算法，学习用另一种语言编程往往也不是那么容易的事，比如 C++。

第三章最简单的 C 程序设计——顺序程序设计

P37

对第 3 章的点评

第 3 章 最简单的 C 程序设计——顺序程序设计

开场白讲了两件事：作者对如何学习编程的看法，介绍该书的做法。与该章的主题没什么关系。前者在第二章已经说过，后者应该是前言或第一章的内容，写在这里有没话找话自吹自擂之嫌。

作者对如何学习编程的看法基本是片面、偏颇及错误的，是对 Wirth“算法+数据结构=程序”的曲解和错误发挥。这对学习者是一种根本上的误导。

3.1 顺序程序设计举例

两个例子从问题的提法到最后的代码都糟糕透顶、漏洞百出。这两个例题给学习者带来的负面影响有很多。从这两个例题中学习者不可能领悟到究竟什么是顺序程序设计，更不可能领悟到究竟什么是程序设计。他们很可能误以为编程就是把代数式写成表达式 + 进行函数调用。

更严重的是，谭书居然告诉读者“警告”可以作为正常来接受，这是教唆式的误导。

关于例 3.1

例 3.1 有人用温度计测量出用华氏法表示的温度(如 69°F)，今要求把它转换为以摄氏法表示的温度(如 20°C)。

这个题目看似描述的平易近人通俗易懂。但我想小学语文老师会评价说，温度就是温度，和“有人”或没人、用或不“用温度计测量”有什么关系？“有人用温度计测量出”完全是废话。而小学数学老师会说，谭同学，69°F 根本不是 20°C。

这就是老谭。有人说他的书通俗易懂，其实多半只是似是而非且毫无内容的废话甚至病句而已。如果阅读不仔细很容易被他的似是而非唬过去，但只要稍微一推敲，立刻就能把他戳出几个洞。赞赏他的书的人，通常小学数学和小学语文都不咋样

此外，在题解中有这样一些毛病：

把算法说成是找公式

流程图写输入，但代码中是赋值运算

代码使用不合适的数据类型

容易出错的代码风格

……

3.2 数据的表现形式及其运算

这部分总的来说就是概念不清，逻辑混乱。

3.2.1 常量和变量

1.常量

“常用的常量有以下几类：（1）整型常量……（2）实型常量……（3）字符常量……（4）字符串常量……（5）符号常量”

2.变量
3.常变量
4.标识符
从这个标题结构很容易发现其荒谬，总体来说就是不伦不类：
把标识符和常量、变量并列；把“常变量”和“变量”并列；
对常量的分类同样是不伦不类
先介绍“符号常量”后介绍“标识符”是本末倒置

此外还有如下错误
3.2.1 之 1.常量 部分
"整型常量。如 1000，12345，0，-345 等都是整型常量。",这里把“-345”这个常量表达式说成了常量
刚说完“十进制小数形式，由数字和小数点组成”，立刻给出了“-56.79”这个例子，明显自相矛盾。试问，“-”是数字还是小数点？
对字符常量和符号常量的讲解均存在错误。

3.2.1 之 2.变量 部分
对编程最重要的概念之一——变量的介绍存在惊人的致命错误（“变量名实际上是以一个名字代表的一个存储地址”）
“C99 允许在函数中的复合语句（用一句花括号括起来）中定义变量。”属于用 C89 的知识冒充 C99

3.2.1 之 3.常变量 部分
把 const 变量想当然地说成是 C99 的，实际上 C89 就有 const 这个关键字

3.2.1 之 4.标识符 部分
介绍的是过时（对于 C99 来说）的说法

P37

> 第 3 章 最简单的 C 程序设计——顺序程序设计
> ……
> 为了能编写出 C 语言程序，必须具备以下的知识和能力：
> (1) 要有正确的解题思路，即学会设计算法，否则无从下手。
> (2) 掌握 C 语言的语法，知道怎样使用 C 语言所提供的功能编写出一个完整的、正确的程序。也就是在设计好算法之后，能用 C 语言正确表示此算法。
> (3) 在写算法和编写程序时，要采用结构化程序设计方法，编写出结构化的程序。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p37

这段总的、指导性的叙述，由于并不全面，具有本质性的缺失，因而是对初学者不折不扣的误导。

事实上仅有这些是不够的，当初学者仅仅注意这几点的时候是绝对不可能学好程序设计的。

众所周知，程序由数据、运算、控制和传输这四个基本成分。谭的叙述中完全看不到“数据”这个最基本成分的影子，这就造成了这样一种现象，许多学习谭书的人往往津津乐道地把“算法才是王道”挂在嘴边，但却对数据缺乏起码的必要了解。谭书本身最致命的一个根本缺陷就是对数据缺乏重视，不甚了了甚至存在大量的根本性的错误。

此外不难发现，谭的这段叙述和 Wirth 所说的“算法+数据结构=程序”在精神上是背道而驰的。

3.1 顺序程序设计举例

P37

3.1 顺序程序设计举例

例 3.1 有人用温度计测量出用华氏法表示的温度(如 69° F)，今要求把它转换为以摄氏法表示的温度(如 20° C)。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p37

这个题目看似描述的平易近人通俗易懂。但我想小学语文老师会评价说，温度就是温度，和“有人”或没人、用或不“用温度计测量”有什么关系？“有人用温度计测量出”完全是废话。而小学数学老师会说，谭同学，69° F 根本不是 20° C。

这就是老谭。有人说他的书通俗易懂，其实多半只是似是而非且毫无内容的废话甚至病句而已。如果阅读不仔细很容易被他的似是而非唬过去，但只要稍微一推敲，立刻就能把他戳出几个洞。赞赏他的书的人，通常小学数学和小学语文都不咋样。

P37~38

例 3.1 的讲解存在这样几个问题

1.

解题思路：这个问题的算法很简单，关键在于找到二者间的转换公式……
————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p37~38

找个简单的公式也是算法？

2.

图 3.1（N-S 图）中
输入 f 的值

但代码中却是

```
f = 64.0;
```

这是“输入”吗

3.

代码中出现了从来没介绍过的关键字 float

4.

代码中计算摄氏温度的语句是

```
c=(5.0/9)*(f-32);
```

这个风格怪异不说

初学者不可能明白为什么原来公式中的 5 写成了 5.,而原公式中的 9 和 32 却没变

5.

老谭最后说

读者应能看懂这个简单的程序。

我看读者根本看不懂

P38

例 3.2 计算存款利息。有 1000 元，想存一年。有 3 种方法可选：（1）活期，年利率为 r1；(2)一年定期，年利率为 r2；(3)村两次半年定期，年利率为 r3。请分别计算出一年后按三种方法所得到的本息和。

这也叫例题？谁知道你 r1,r2,r3 是神马东西啊
谁能做出来？我出 1000 元悬赏！
在公司里搞出这样的需求分析老板不抽他才怪

1000 明明是问题中的常量
却非要设个变量（float p0=1000,）
难道不是脱裤子放屁吗？
既丑陋无比又容易弄出差错
这难道不是教小朋友们学坏吗

P38

例 3.2————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p38

再次出现了“N-S 流程图”这个词
在书中一会儿“N-S 流程图”，一会儿“N-S 图”
明显的缺乏概念的统一
是形式逻辑中所说的“概念不清”的一种
N-S 图中写的是“输入 p0,r1,r2,r3 的值”
代码中却是赋初值“float p0=1000,r1=0.0036,r2=0.0225,r3=0.0198,p1,p2,p3;”

P38

更可笑的是

运行结果： p1=1003.599976 p2=1022.500000 p3=1019.898010 ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p38

竟然给出了小数点后面的第 6 位
银行会按这个给你付钱吗？
不懂银行的业务写出这种代码只能是一个笑柄
更惊人的是
1000 元按照年利率 0.0036 计
小学生用心算都能脱口而出本息是 1003.6 元
而你这个程序给出的结果竟然是 1003.599976
这不是糟蹋 C 语言是什么
所以读老谭的书会变傻
这是确凿无疑的

P38

程序分析：第 4 行，在定义实型变量 p0,p1,p2,p3,r1,r2,r3 的同时，对变量 p0,r1,r2,r3 赋予初值。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p38

啥叫“实型变量”啊？
再说，在 N-S 图里不是“输入 p0,r1,r2,r3 的值”吗？
怎么到这里又成了“赋予初值”了呢？

P38

第 8 行，在输出 p1,p2,p3 的值之后，用\n 使输出换行。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p38

这段文字出现在第 38 页
到这里为止，代码中的\n 至少出现过 20 次左右
老谭已经讲过十几次\n 是换行了
车轱辘话来回说是老谭的主要作文技巧

P37~39

```
#include <stdio.h>
int main()
{
   float f,c;
   f=64.0;
   c=(5.0/9)*(f-32);
   printf("f=%f\nc=%f\n",f,c);
  return 0;
}
```

在使用 Visual C++6.0 编译系统时，……，会显示出“警告”信息：“在初始化时把一个双精度常量赋给一个 float 型变量”。

评：这条警告信息是捏造的，在这段代码中根本不存在“初始化”

P39

> 注意：……这是因为有的 C 编译系统（如 Visual C++）把所有的实数都作为双精度处理。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p39

毫无根据的臆断，信口开河

P39

> 对这类“警告”，用户知道是怎么回事就可以了。承认此现实，让程序继续进行连接和运行，不影响运行结果。
> ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p39

为自己的垃圾代码辩解
教唆初学者放过警告
严重地误导初学者

3.2 数据的表现形式及其运算

P39

> 3.2 数据的表现形式及其运算
> ……
> 3.2.1 常量和变量
> ……
> （1）整型常量。如 1000，12345，0，-345 等都是整型常量。
> ————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p39

首先“整型”这个词含义是不明确的，在谭书中一直有 n 种并存的含义
其次，-345 不是常量，而是常量表达式。如果可以把-345 说成常量，那么 3-4 也可以说成是常量。岂不荒谬

P39

(2)实型常量。有两种表示形式：————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p39

老谭既然号称“按照 C 语言的新标准 C99 进行介绍”，怎么会写出实型常量有两种表示形式这样的胡话？
这只能说明老谭对 C99 缺乏最基本的了解
还是老作风：信口开河

P39

（3）字符常量。有两种形式的字符常量：
①普通字符，用单撇号括起来的一个字符，如：'a'，'Z'，'3'，'？'，'#'。不能写成'ab'或'12'……字符常量只能是一个字符，……。————谭浩强 ，《C 程序设计》（第四版），清华大学出版社，2010 年 6 月，p39

这一页上的第 6 个硬伤。

P37

评：开场白讲了两件事：作者对如何学习编程的看法，介绍该书的做法。与该章的主题没什么关系。前者在第二章已经说过，后者应该是前言或第一章的内容，写在这里有没话找话自吹自擂之嫌。
作者对如何学习编程的看法基本是片面、偏颇及错误的，是对 Wirth“算法+数据结构=程序”的曲解和错误发挥。这对学习者是一种根本上的误导。

评：两个例子从问题的提法到最后的代码都糟糕透顶、漏洞百出。这两个例题给学习者带来的负面影响有很多。从这两个例题中学习者不可能领悟到究竟什么是顺序程序设计，更不可能领悟到究竟什么是程序设计。他们很可能误以为编程就是把代数式写成表达式 + 进行函数调用。
更严重的是，谭书居然告诉读者“警告”可以作为正常来接受，这是教唆式的误导。

例 3.1 有人用温度计测量出用华氏法表示的温度(如 69°F)，今要求把它转换为以摄氏法表示的温度(如 20°C)。

评：这个题目看似描述的平易近人通俗易懂。但我想小学语文老师会评价说，温度就是温度，和“有人”或没人、用或不“用温度计测量”有什么关系？“有人用温度计测量出”完全是废话。而小学数学老师会说，谭同学，69°F 根本不是 20°C。
这就是老谭。有人说他的书通俗易懂，其实多半只是似是而非且毫无内容的废话甚至病句而已。如果阅读不仔细很容易被他的似是而非唬过去，但只要稍微一推敲，立刻就能把他戳出几个洞。赞赏他的书的人，通常小学数学和小学语文都不咋样
此外，在题解中有这样一些毛病：
把算法说成是找公式
流程图写输入，但代码中是赋值运算

代码使用不合适的数据类型
容易出错的代码风格
……

3.2 数据的表现形式及其运算
评：这部分总的来说就是概念不清，逻辑混乱

3.2.1 常量和变量
1.常量
“常用的常量有以下几类：（1）整型常量……（2）实型常量……（3）字符常量……（4）字符串常量……（5）符号常量”
2.变量
3.常变量
4.标识符
评：从这个标题结构很容易发现其荒谬，总体来说就是不伦不类：
把标识符和常量、变量并列；把“常变量”和“变量”并列；
对常量的分类同样是不伦不类
先介绍“符号常量”后介绍“标识符”是本末倒置

此外还有如下错误
3.2.1 之 1.常量 部分
"整型常量。如 1000，12345，0，-345 等都是整型常量。",这里把“-345”这个常量表达式说成了常量
刚说完“十进制小数形式，由数字和小数点组成”，立刻给出了“-56.79”这个例子，明显自相矛盾。
试问，“-”是数字还是小数点？
对字符常量和符号常量的讲解均存在错误。

3.2.1 之 2.变量 部分
对编程最重要的概念之一——变量的介绍存在惊人的致命错误（“变量名实际上是以一个名字代表的一个存储地址”）
“C99 允许在函数中的复合语句（用一句花括号括起来）中定义变量。”属于用 C89 的知识冒充 C99

3.2.1 之 3.常变量 部分
把 const 变量想当然地说成是 C99 的，实际上 C89 就有 const 这个关键字

3.2.1 之 4.标识符 部分
介绍的是过时（对于 C99 来说）的说法

P39~41

评：这一小节讲的是“常量”
除了前面提到的错误、概念或知识缺失、含糊不清、误导等
回过头来看还会因为分类错乱而感觉很滑稽
（1）整型常量
（2）实型常量

（3）字符常量
（4）字符串常量
（5）符号常量
这完全是一脚天上一脚地下，关公战秦琼式的写法

p40

③ 转义字符……'\t'代表将输出的位置跳到下一个 tab 位置（制表位置），一个 tab 位置为 8 列
评：tab 位置是实现定义的

表 3.1 转义字符及其作用

转义字符	字符值	输出结果
\'	一个单撇号(')	具有此八进制码的字符
\"	一个双撇号(“)	输出此字符

……

评：对“\' ”输出结果的描述不知所云

p40

字符串常量是双撇号中的全部字符（但不包括双撇号本身）
评：这个表述很似是而非
没有讲清 String literal 的本质
如果是讲述 String literal 中的字符，那么忽视了还有一个'\0'

单撇号内只能有包含一个字符， 双撇号内可以包含一个字符串。

前半句是错的，后半句看不懂究竟想说什么。

p41

（5）符号常量。……
#define PI 3.1416
经过以上指定后，本文件中此行开始的所有的 PI 都代表 3.1416。在对程序进行编译前，预处理器先对 PI 进行处理，把所有的 PI 全部置换为 3.1416。
评：成问题的地方在“所有的”那三个字，这给人以严重的误导。至少我见过读了这句话之后试图替换“…… PI ……” 或 v_PI 中的 PI 的学习者。

p41

2.变量
评：这一小节错得很不靠谱
已经突破了信口开河的境界达到胡说八道的高度了
例如
C99 允许在函数中的复合语句（用一句花括号括起来）中定义变量。
C99 允许使用常变量，如：
const int 3=3;

我注意到本书到此为止关于 C99 的论述至少有九成都是错误的
而且这些错误并没涉及到什么复杂的概念，只涉及一些简单的基本常识
在这些常识性的地方居然大错特错
那么就有理由怀疑老谭究竟看过 C99 没有（哪怕是粗略地看过）
根本没看过 C99 就在书里自称“按照 C 语言的新标准 C99 进行介绍”
这不是学识问题
是品质问题

p41

请注意变量名和变量值这两个不同的概念，……
评：实际上这两个概念有时是同一的，是区分不开的。
而且老谭根本也没有讲清楚这两个概念的区别究竟何在

p41

变量名实际上是以一个名字代表的一个存储地址。
评：这是简直是胡扯！
假如有
int i;
如果 i 代表地址，那么 &i 又是什么呢？

p41

C99 允许使用**常变量**

评：且不谈“常变量”这个翻译有多么拙劣
const 关键字早在 C89 中就有了
关 C99 什么事

P42

常量是没有名字的不变量。
评：常量就一定没有名字吗？显然不是

P42

定义符号常量是用#define 指令，它是预编译指令，它只是用符号常量代表一个字符串，……
评：“字符串”这个词在 C 语言中是有特定含义的，这里使用这个词明显不当

P42

C 语言规定标识符只能由字母、数字和下划线 3 种字符组成，
评：不是号称“按照 C99 介绍”的吗？
C99 是这样规定的吗？！
麻烦老谭自己亲自去看看
不要把几十年前的规定拿来冒充 C99

P42

3.2.2 数据类型
评：数据类型是 C 语言最基础最根本性的东西。对数据类型的理解程度基本上决定了一个人 C 语言的水平。同样一本书对数据类型的介绍是否准确全面也在很大程度上决定了书的优劣。下面来看看谭书对数据类型的介绍。

P42

所谓类型，就是数据分配存储单元的安排，包括存储单元的长度（占多少字节）以及数据的存储形式。不同的类型分配不同的长度和存储形式。
评：除了最后一句是病句以外，这段话最大的毛病在于并不全面，仅仅把数据类型理解为数据在存储单元中的表示形式大约只理解了数据类型全部含义的 45%。纵观全书，这个毛病始终存在。这一点我认为是谭书一个致命的硬伤。由于这个致命的硬伤，所以这本书的错误并非象有些天真的人们以为的那样可以通过修修补补改好——因为根本性的基调就是不准确和错误的。

该书把数据类型分为“基本类型”、“枚举类型”、“空类型”和“派生类型”。(表 3.4)
这也是要命的，反映了作者对数据类型的一知半解并且根本不懂得数据类型的重要性。
首先“枚举类型”与“基本类型”、“派生类型”并列是不伦不类的半调子分类。回顾本书的前几版，这个“枚举类型”的位置始终很尴尬，有时根本没有，有时与“整型”、“字符型”、“实型”并列。作者一直把“枚举类型”当作临时工一样今天放这，明天放那，后天还可能给解雇了。这些说明作者这几十年一直没能把数据类型搞清。

表 3.4 中的另外几个错误是

缺乏不完全数据类型，只写了一个 void 类型

*布尔型(bool) ,这个写错了，再次表明老谭根本没读过 C99，但他却宣称“按照 C99 进行介绍”，品质问题。

浮点类型中缺少 long double。

复数浮点型（float_complex,double_complex,long long_complex）:这个非但不全，而且全错。

仅仅这一张表上就有如此多的严重错误，这本书绝对是垃圾到家了

P42

3.2.2 数据类型

评：“为什么在用计算机运算时，要指定数据的类型呢？”这个设问的可笑之处在于后面根本没有回答，而是驴唇不对马嘴地瞎扯了一通。

“所谓类型，就是数据分配存储单元的安排，包括存储单元的长度（占多少字节）以及数据的存储形式。不同的类型分配不同的长度和存储形式。”
表明作者根本不懂得数据类型的含义。

把数据类型分为“基本类型”、“枚举类型”、“空类型”和“派生类型”（表 3.4）。很荒唐
更荒唐的是一口气写错了 4 个描述类型的关键字

3.2.3 整型数据

评：概念不清，逻辑混乱。（比如，整型变量的符号属性）纠结于具体编译器下各数据类型的空间（范围）和所谓的“补码”，并没有真正讲清楚数据类型本质性的东西。
一些原则性的错误：
在将一个变量定义为无符号整型后，不应向它赋予一个负值，否则就会得到错误的结果。
变量值在存储单元中都是以补码形式存储的

3.2.4 字符型数据

评：“因此 C99 把字符型数据作为整数类型的一种”，信口开河。

p43

例如，Visual C++ 6.0 为 char 型(字符型)数据分配 1 个字节，

评：居然是“例如”!
敢问老谭你能否找到一个“例外”？

p44

3.2.3 整型数据

1.整型数据的分类

本节介绍最基本的整型类型。

（1）基本整型(int 型)

编译系统分配给 int 型数据 2 个字节或 4 个字节（由具体的 C 编译系统自行决定）。如 Turbo C2.0 为每一个整型数据分配 2 个字节（16 个二进位），而 Visual C++为每一个整型数据分配 4 个字节（32 位）。在存储单元中的存储方式是：用整数的补码(complement)形式存放。

评：这段中“整型”的含义是前后不一的，这是最基本的逻辑错误：概念不清，其后果必然是逻辑错乱
“编译系统分配给 int 型数据 2 个字节或 4 个字节”：字面意思是 int 类型只可能有两种可能，这也是错误的。

“在存储单元中的存储方式是：用整数的补码(complement)形式存放”：这是一相情愿的想当然

p44

在存放整数的存储单元中，最左面一位是用用来表示符号的，如果该位为 0，表示数值为正；如果该位为 1，表示为负。

评：1. unsigned 算不算整数？ 哪位表示符号？
2. 什么叫“最左面一位”，拜托，那叫“高位”
3. 0000 0000 “表示数值为正”吗？乱

p44

1.整数类型的分类
（1）基本整型（int 型）
（2）短整型（short int）
（3）长整型（long int）
（4）双长整型（long long int）

评：最多是对 Turbo C2.0 和 Visual C++6.0 的情况进行反复举例而已，并且错误地把补码、一个字节 8 位这一特殊情况作为了唯一情况

p45

编译系统分配给 long 数据 4 个字节

评：又是想当然

p45

2.整型变量的符号属性

评：这是个不通的标题。暂且不说“符号属性”这个概念是否恰当，即使真有“符号属性”这个概念，那么它也是数据类型的性质，而不是“变量”的性质

p45

以上介绍的几种数据类型，变量值在存储单元中都是以补码形式存储的

评：胡扯！

p46

因此无符号整型变量中可以存放的正数的范围比一般整型变量中正数的范围扩大一倍。

评：抛开语文等方面的问题不谈
就以两个字节的补码表示形式的 int 以及 unsigned 来说
int:1~32767
unsigned:1~65535
这难道是扩大一倍吗？
老谭显然掰不开正数和非负数这两个概念的区别

p46

（2）对无符号整型数据用%u 输出。%u 表示用无符号十进制数的格式输出。如：
unsigned short price = 50 ;
printf("%u\n",price);
评：不否认能输出正确的结果
但是“对无符号整型数据用%u 输出”显然是荒谬的
这里的“整型”是个集合概念还是个体概念？
如果是集合概念，陈述显然是错误的
如果是个体概念（unsigned int）怎么可以用 short 举例呢
老谭显然根本不清楚 printf("%u\n",price)函数调用中的 price 表达式的真实类型
输出正确结果其实是张冠李戴之后瞎蒙碰出来的结果

“无符号十进制数”也属于莫名其妙的说法
老谭好像是在自言自语地说着自己才仿佛懂得的话

p47

在将一个变量定义为无符号整型后，不应向它赋予一个负值，否则就会得到错误的结果。如：
unsigned short price = - 1 ;//不能把一个负整数存储在无符号变量中
printf("%d\n",price);
得到的结果为 65535.显然于原意不符。
思考：这是为什么？
原因是：系统对-1 先转换成补码形式。就是全部二进制位都是 1（见图 3.8），然后把它存入变量 price 中。由于 price 是无符号短整型变量，其左面第一位不代表符号，按%d 格式输出，就是 65536.

price
1111111111111111
图 3.8
评：这段短短的文字至少有四个错误
找出 3 个 C 语言算入门了

p47

3.2.4 字符型数据

由于字符是按其代码（整数）形式存储的，因此 C99 把字符型数据作为整数类型的一种。但是，字符型数据在使用上有自己的特点，因此把它单独列为一节介绍。

评：1.代码、编码，两个概念。老谭这个“代码”大概是翻译软件翻译的吧

2.字符类型作为整数类型的一种，C89 就已然如此，和 C99 有什么关系。老谭这是在不懂装懂地蒙人

3.每种数据类型都有自己的特点。在这点上，字符型数据没什么特殊的

p47

字符与字符代码并不是任意写一个字符，程序都能识别的。

评：这是人话吗

主语是什么

谓语是什么

宾语又是什么

程序识别什么

p47

例如圆周率π 在程序中是不能识别的，只能使用系统的字符集中的字符集，目前大多数系统采用 ASCII 字符集。

评：谁说π 就是圆周率的？

什么叫“识别”

什么“系统”

“字符集”有几套？

“大多数系统采用 ASCII 字符集”吗？

前面有网友说老谭无论对 C89 还是 C99 都还没入门

确实一针见血

p47

在将一个变量定义为无符号整型后，不应向它赋予一个负值，否则就会得到错误的结果。如：

```
unsigned short price = - 1 ;//不能把一个负整数存储在无符号变量中
printf("%d\n",price);
```

得到的结果为 65535.显然于原意不符。

思考：这是为什么？

原因是：系统对-1 先转换成补码形式。就是全部二进制位都是 1（见图 3.8），然后把它存入变量 price 中。由于 price 是无符号短整型变量，其左面第一位不代表符号，按%d 格式输出，就是 65536.

price

1111111111111111

图 3.8

评：向无符号整数类型变量赋值，C 语言是有定义的。不是什么错误的结果。不能因为自己不理解 C 语言的真正语义就说那是错误的

printf("%d\n",price);得到的结果为 65535 。不是什么与“原意不符”，而是作者根本不懂得那段代码的真正含义（尤其是其中存在的类型转换）

C 语言没说一定采用补码

那个 price 其实并不是 unsigned short，而是一个 int 类型

……

此外很多不严谨的地方就不多说了

p48

字符是以整数形式（字符的 ASCII 代码）存放在内存单元中的。

评：C 语言从来没说过只处理 ASCII 码

p48

所以在 C 中，指定用 1 个字节（8 位）存储一个字符（所有系统都不例外）

评：据我所知，有的系统的 1 字节是 9 位

老谭你怎么解释这个“所有系统都不例外”

p48

2. 字符变量

字符变量是用类型符 char 定义字符变量。

评：这是人话么？

p48

字符变量是用类型符 char 定义字符变量

这是话么？

这就是简介说的“文字流畅”？

当然

与以前的“字符型变量用来存放字符常量”还是天壤之别

以前的是错话

现在的是废话

(1)+、-、*、/运算的两个数中有一个数为 float 或 double 型，结果是 double 型，因为系统将所有的 float 型数据都先转换为 double 型，然后进行运算。

评：一句不长的话错了两次。

1.“+、-、*、/运算的两个数中有一个数为 float 或 double 型，结果是 double 型”

老谭既然在前面吹嘘这本书是“按照 C 语言的新标准 C99 进行介绍”

难道不知道 C 语言有复数类型吗？

如果“+、-、*、/运算的两个数中有一个数为 float 或 double 型”，另一个是复数类型，“结果是 double 型”吗？

就算你根本不懂得 C99

虚张声势地出来蒙人

long double 类型可是几十年前就有了

如果“+、-、*、/运算的两个数中有一个数为 float 或 double 型”，另一个是 long double，“结果是 double 型”吗？

2.“系统将所有的 float 型数据都先转换为 double 型，然后进行运算。”，简直是胡扯

(2)如果 int 型与 float 或 double 型数据进行运算，先把 int 型和 float 型数据转换为 double 型，然后进行运算。结果是 double 型

评：第一次读完这个(2)，我愣住了，简直不敢相信

再仔细看一遍，发现这个(2)真的是很(2)

因为什么呢

因为(1)已经说过“+、-、*、/运算的两个数中有一个数为 float 或 double 型”

现在(2)居然又开始大谈特谈“如果 int 型与 float 或 double 型数据进行运算”

车轱辘话来回说是老谭的一个特点

但刚说完就 repeat 一遍还是罕见的

中等程度的神经错乱都写不出来这种东西

而且，char 、long 等与 float 或 double 型数据进行运算的情况，阙如

这段中的严重错误就不谈了

与错乱相比，错误简直不算什么了不起的事情

(3)字符(char)型数据与整型数据进行运算，就是把字符的 ASCII 代码与整型数据进行运算。如：12 + 'A'，由于……

评：前面说的是“字符(char)型数据与整型数据进行运算”

立刻就拿出个两个 int 类型数据的加法来举例

很明显

老谭根本不知道 'A' 的类型究竟是什么

评：暂且不谈 C 语言的执行环境未必使用 ASCII 码的问题）

这里的“整型数据”是个莫名其妙的概念

在该书的 43 页写到

| 基本整型 （int）
| 短整型 （short int）
| 长整型 （long int）
整型类型 | *双长整型 （long long int）
| 字符型 （char）
| *布尔型 （bool） [注：这里的 bool 也是错误的]

所谓“字符(char)型数据与整型数据进行运算”
说的到底是哪种"整型类型"呢？
况且这根本就是一个不必要的条件限制
难道 char 类型数据 与 double 类型数据 做加法运算时那个 char 类型数据不是字符编码吗？
它不是字符的编码还能是什么别的东西吗?

字符数据可以直接与整型数据进行运算。
评：不知所云。
什么叫“直接”进行运算？
是否还有“间接”进行运算？

⑤将 10+'a'+i*f 的结果与 d/3 的商 2.5 相减，……
评：1. “实型数据”，应为（实）浮点型数据。因为在 C 语言中 real types 是整数类型与（实）浮点类型的合称。
2.“字符的 ASCII” ，运行环境中的字符不一定是用 ASCII 码表示的。况且，即使运行环境也使用 ASCII 码，字符数据中存储的也不一定是 ASCII 码。比如，字符数据可能是一个负值
3.“转换为 double 型数据” ，完全无视了 integer promotions。而且最终可能转换成三种而不是只有 double 这一种。
紧接着

以上的转换是编译系统自动完成的，用户不必过问。
评：不知所云。

分析下面的表达式，假设已指定 i 为整型变量，值为 3，f 为 float 型变量，值为 2.5，d 为 double 型变量，值为 7.5 。
10+'a'+i*f-d/3
编译时，从左到右扫描，运算次序为：
①进行 10+'a'的运算，……
②由于“*”比“+”优先级高，先进行 i*f 的运算。先将 i 和 f 都转成 double 型，……
③整数 107 与 i*f 的积相加。
④进行 d/3 的运算……

⑤将 10+'a'+i*f 的结果与 d/3 的商 2.5 相减，……

评：最雷人的是：“编译时……运算次序为”

编译时就运算，那运行时计算机干些什么呢？

“②由于“*”比“+”优先级高，先进行 i*f 的运算”

可是“进行 10+'a'的运算”为什么先做了呢，

难道“*”比第 1 个“+”优先级低、比第二个“+”优先级高吗？

“先将 i 和 f 都转成 double 型”

错的。

小学语文问题：

“i*f 的积”，

“将 10+'a'+i*f 的结果与 d/3 的商 2.5 相减”

4.不同类型数据的混合运算

……如果一个运算符的两侧的数据类型不同，则先自动进行类型转换，使两者具有同一类型，然后进行运算。

评：若有

int a[1];

那么对于表达式

*(a + 0)

显然“+”“两侧的数据类型不同”。

老谭能否给解释一下，如何“自动进行类型转换”？使两者具有了什么样的“同一类型”？

3.算术表达式和运算符的优先级和结合性

用算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式。运算对象包括常量、变量、函数等。例如，下面是一个合法的 C 算术表达式：

a*b/c-1.5+'a'

C 语言除了规定了运算符的优先级外，还规定了运算符的结合性。在表达式求值时，先按运算符的优先级别顺序执行，例如先乘除后加减。如表达式 a-b*c，b 的左侧为减号，右侧为乘号，而乘号的优先级高于减号，因此，相当于 a-(b*c)。

如果在一个运算对象两侧的运算符的优先级别相同，如 a-b+c，则按规定的“结合方向”处理。C 语言规定了各种运算符的结合方向（结合性），算术运算符的结合方向都是“自左至右”，即先左后右，因此 b 先与减号相结合，执行 a-b 的运算，然后再执行加 c 的运算。“自左至右的结合方向”又称“左结合性”，即运算对象先与左面的运算符结合。以后可以看到有些运算符的结合方向为“自右至左”，即右结合性（例如，赋值运算符，若有 a=b=c，按从右到左的顺序，先把变量 c 的值赋给变量 b，然后变量 b 的值赋给 a）。关于“结合性”的概念在其他一些高级语言中是没有的，是 C 语言的特点之一，希望能弄清楚。附录 D 列出了所有运算符以及它们的优先级别和结合性。

评：首先，来看一下引用部分小标题里的第一个词——“算术表达式”。

这个词很给人一种“亲切感”特别有迷惑力，然而它确是一个模糊的、似是而非而且毫无用处的概念。

据我所知，这个词是老谭自创的。C 语言中并没有这样的概念。

C 语言中只有算术类型（arithmetic types）和算术运算符（arithmetic operators）这样的概念，并没有“算术表达式”这种概念。

没有这样的概念，难道不可以自己创造概念吗？当然可以。但必须知道的是，创造概念是有前提的：

创造者要给出概念的定义；

概念要科学严谨；

这个概念有用，或者方便简洁地描述了一个道理，或者帮助别人认识了一类现象或规律。

这样才可以创造新概念。

不满足这三个前提，自创概念不是吃饱了撑的就是假行家故弄玄虚的蒙人行为。考察一下“算术表达式”这个概念。

作者给出了定义：“用算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式。”

（很好，老谭自创概念不给定义的例子比比皆是，这次很有进步）

然而，这个概念并不科学，也不严谨。为什么这么说呢？简单地考察一下下面的表达式就会知道了：

```
1+(2+3)
```

在这里，如果把“+”说成是“连接”“操作数”还是勉强说得过去的，但是“()”的意义则绝对不是为了“连接”“操作数”。“()”的意义是为了表明其内部的子表达式作为一个整体——仿佛一个独立的操作数参与运算，这与“连接”是八竿子打不着的。再比如“(1)”这个表达式中，“()”连接了什么呢？其实它什么也没连接，它只表明它与其扩起来的部分是一个完整的整体而已。

所以说，这里的“()”是一种“界定”(to delimit)范围的符号，把它与表示运算的运算符并列在一起是很荒唐的。

作者接着补充道：“运算对象包括常量、变量、函数等”。

这就让人迷惑不解了，这里的“函数”究竟是指的什么呢？是“函数调用”还是“函数名”本身呢？如果是“函数调用”，就成了自己打自己耳光，因为函数调用一定是通过函数调用运算符“()”实现的（注意这里这个"()"不是括号），这和作者在前面说的“用算术运算符和括号将运算对象（也称操作数）连接起来的”自相矛盾。如果这里的所谓“函数”是“函数名”的话，总所周知，“函数名”这种数据类型怎么可能进行算术运算呢？

所以必须感谢作者的及时补充，他成功地使用“函数”这个词戳穿了他自己创造的伪概念——“算术表达式”，这个概念是经不起推敲的。

紧接着的“例如”也很成问题。权且承认他给出“a*b/c-1.5+'a' ”是合法的，但是这个表达式即不是“用算术运算符和括号将运算对象（也称操作数）连接起来的”（因为没有括号），“运算对象”也不“包括常量、变量、函数”（“函数”在哪？），作者举例到底想说明什么呢？无厘头么！

或许，有人很不以为然：“括号”和“函数”都是随手写的，不算大错。好，至少现在你已经承认这书很不严谨了。问题是我没提到这些错误之前，你发现这些错误了吗？就算你也和我一样发现了这些错误，那些把这本书作为教材的初学者怎么可能发现这些错误呢？你老人家胡写一通不要紧，你让那些初学者情何以堪呢？

又或许有人觉得这段文字只要像下面那样修改一下就可以了

“用算术运算符和将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式。运算对象包括常量、变量等。例如，下面是一个合法的 C 算术表达式：

a*b/c-1.5+'a'　”

Sound good!这段文字似乎基本没什么问题了。然而我们还是要问，这个自创的概念有什么用吗？继续往下看。下面一段文字是

“C 语言除了规定了运算符的优先级外，还规定了运算符的结合性。在表达式求值时，先按运算符的优先级别顺序执行，例如先乘除后加减。如表达式 a-b*c，b 的左侧为减号，右侧为乘号，而乘号的优先级高于减号，因此，相当于 a-(b*c)。”

我在读这一段第一句话的时候被骇了一大跳。这句型，简直太惊人了。前面压根没提过“优先级”，抽冷子就来了一句“C 语言除了规定了运算符的优先级外，还……”，有这么说话的吗？如果连话都说不利索，但却非要写在教科书里，您老人家又成天把这个“1100 万”挂在嘴上，您说这和一只著名的苍蝇成天炫耀自己污染过很多食物有区别吗？

也许有人觉得这只是语文问题，不是 C 语言方面的问题。好吧，我让步！问题是这段文字和前面作者自创的“算术表达式”没有任何关系，没有人能否认这一点吧？！

再看引用文字的最后一段。这段文字同样和“算术表达式”这个伪概念没有任何关系。唯一可能牵强地和“算术表达式”扯上一点关系的是“算术运算符的结合方向都是‘自左至右’”，然而这句话本身就是错误的，因为作者在书中的第 52 页明确地说明一元“+”、“-”都是算术运算符，然而我们都知道一元“+”、“-”的结合性是“自右至左”，更何况“算术运算符”和“算术表达式”分明就是两个不同的概念。

而且，优先级和结合性的规律是对所有运算符而言的，算术运算符并没有什么特殊的优先级和结合性规律；而且所谓的“算术表达式”在这方面也没有任何特殊性而言，创造“算术表达式”比画蛇添足还要无聊。不仅无聊，而且有害。这个概念甚至无法比喻成懒婆娘的裹脚布，最多只能称得上是懒婆娘丢弃的裹脚布而已。

问题出来了，谭大爷为什么要自创这个不伦不类而且漏洞百出的伪概念——“算术表达式”，他到底想说什么？我估计他自己也不清楚。

话说到这里，我想我已经证明了这段引文的“胡扯性”。但是，根据我的经验，这时候一定会有无耻之徒跳出来无赖地狡辩：“毕竟还介绍了优先级和结合性么”——没办法不让他们跳出来，“卑鄙是卑鄙者的通行证”。然而，在这里，即使是这种最无耻的狡辩也是不可能得逞的，因为这段引文对“优先级”和“结合性”的讲解也是错的！

关于优先级，作者是这样写的：

“在表达式求值时，先按运算符的优先级别顺序执行，例如先乘除后加减。如表达式 a-b*c，b 的左侧为减号，右侧为乘号，而乘号的优先级高于减号，因此，相当于 a-(b*c)。”

这段文字的错误在于作者把运算符的优先级和表达式求值的执行次序混为了一谈。尤其是“先乘除后加减”更是对优先级的曲解。实际上，优先级和表达式求值的执行次序基本上是不相干的两回事情。

优先级的意义在于表明表达式的含义（结合性也是），而求值的执行次序则是编译器的自选动作。只要不违反表达式的含义，编译器可以按照自己的爱好安排求值次序，编译器也没有义务告诉你它是按照什么次序求值的。这种自选动作叫 implementation-defined behavior。

举例来说,对于表达式“ 1+2-3*4 ”，“*”的优先级高于“-”，其意义在于表明减去的是（3*4）而不是减 3。只要你清楚地表明了这一点，你的任务就完成了。至于计算次序，编译器至少有两种选择：

1）　1+2-3*4　=> 1+2-12 => 3-12 => -9

2）　1+2-3*4　=> 3-3*4　　=> 3-12 => -9

按照外交部的惯用句型来说就是，怎么选择求值次序是编译器自己的事，程序员无权干涉（甚至无权了解，好奇也不行，编译器没义务告诉你）。 程序员的责任在于把表达式的含义写清楚，写正确；编译器的义务仅仅在于给出正确的结果而已。至于算出结果的过程（次序），对不起，关您什么事啊？您管的也太宽了吧。

总之，“优先级”的意思就是优先级高的运算符先选“对象”——运算对象。谭大爷明显是不明白这点小道理，可笑地把优先级和求值次序“绑定”到了一起，一相情愿地拉郎配。

如果优先级高的运算符选完了“对象”，剩下的运算符优先级相同，怎么办呢？

答案也简单，按照结合性挑选“对象”，如果是从左到右的结合方向，就左边的先挑，依次向右进行；如果是从右到左，则顺序相反。例如：

1+2-3*4

由于“*”的优先级最高，所以先挑运算对象，表达式的含义为

1+2-(3*4)

剩下的两个运算符“+”和“-”的优先级相同，所以看结合性，这两个运算符的结合性是从左到右，因此左面的先挑，表达式的含义可以进一步明确为

(1+2) - (3*4)

最后，可以确定“-”的运算对象分别为 (1+2) 和 (3*4)。

这就是优先级和结合性的全部含义。如此而已，非常简单。再来看看老谭是怎么讲的。

“如果在一个运算对象两侧的运算符的优先级别相同”，看到这句我差点没吐出来。老谭居然给来了个“如果”“运算对象”“两侧”，这跟优先级、结合性有什么关系吗？如果运算符在运算对象的同一侧怎么办？比如 - - i （不是"--",两个“-”之间有空格），这应该怎么办呢？再比如 a[3]，那个 3 应该算是在运算符的哪一侧呢？

所以看老谭的书，如果你不能发现他的愚蠢，那就只剩下了两种可能：1.你自己愚蠢；2.你正在变得愚蠢。

再往下看：

“算术运算符的结合方向都是'自左至右”，即先左后右'”，这句话本身就是错误的，前面已经提到过。

“因此 b 先与减号相结合”，这句更可笑，纯粹是望文生义式的妄断臆测。老谭把左结合性理解成了“运算对象先与左面的运算符结合”，简直是荒唐透顶。结合性，即使从字面上来说，也不是简单地指操作数与左面或右面的运算符相“结合”。简单举个例子就说明老谭的荒谬，"[]"运算符的结合性是从左至右，那么表达式 a [b[0]] 中 ，无论是 a 或是 b，怎么与左面的运算符结合？所以“运算对象先与左面的运算符结合”这样的昏话绝对不可能是荒谬，只可能是荒谬透顶。

“即右结合性（例如，赋值运算符，若有 a=b=c，按从右到左的顺序，先把变量 c 的值赋给变量 b，然后变量 b 的值赋给 a）。”，这里也是错的。根据结合性，a=b=c 的含义是 a=(b=c) ，也就是说是把 b=c 的值赋给 a，而不是老谭所说的“然后变量 b 的值赋给 a”。

“关于“结合性”的概念在其他一些高级语言中是没有的，是 C 语言的特点之一”，这是井底之蛙坐井观天式思考的结果，根本不用反驳。

“希望能弄清楚”，希望很好，问题是您老人家自己弄清楚没有啊？您总这么瞎忽悠，让学习者怎么弄清楚啊？

“附录 D 列出了所有运算符以及它们的优先级别和结合性。”，第一，附录 D 并没有列出“所有”的运算符，第二，列出的部分不少是错误的。

如果在一个运算对象两侧的运算符的优先级别相同，如 a-b+c，则按规定的“结合方向”处理。

评：这句话蠢就蠢在那句“如果”

它好像是在暗示如果在一个运算对象一侧的运算符的优先级别相同，则不按规定的“结合方向”处理

实际上这个问题和运算符在运算对象的两侧还是一侧根本没有关系
只要是优先级相同，那么总是按结合性处理的

也就是说
那句蠢话相当于说：如果明天下雨，那么明天就是星期一
但就连弱智都懂得，明天是不是星期一和明天是否下雨是没什么关系的

算术运算符的结合方向都是“自左至右”
评：这简直是自己往自己脸上扇了一记大耳光
因为在该书的 52 页
表 3.5 最常用的算术运算符
中
老谭明确无误地把一元 + 和一元 - 运算符都列为了“最常用的算术运算符”
难道一元+和一元-的结合性也是“自左至右”的吗？
（顺便说一句，不知道在老谭眼里，哪些算术运算符不是“最常用的”算术运算符）

p49

可以用以下方法测定本系统的处理方式：
char c=255;
printf("%d\n",c);

在 Visual C++系统上进行编译时，系统给出“警告”（warning）:"把一个整常数赋给 char 变量"。表示 255 已超过 char 变量的数值允许值，在运行时输出-1.说明 Visual C++是把 char 默认为 signed char 类型的。如果把第 1 行改为“unsigned char c=255;”，则不出现“警告”，且输出 255
评：如果一个初学者这样测，是应该夸奖一下的，尽管这是错的
但作为“大师”，居然如此，呵呵，……

p49

用后面的 7 位存放 0 到 127，即 127 个字符的代码。
评：老谭究竟会不会数数？

p49

把可用的字符由 127 个扩展为 255 个，即扩大了一倍。
评：老谭真的是不识数

p49

……把本来不用的第一位用起来。把 char 变量改为 unsigned char，即第一位并不固定设为 0，而是把 8 位都用来存放字符代码。这样，可以存放（2^8-1）即 255 个字符代码。

评：还是不会数数的问题

p49

……为什么在 C 中把实数称为浮点数呢？在 C 语言中，实数是以指数形式存放在存储单元中的。一个实数表示为指数可以有不止一种形式，

评：第一句的设问简直太莫名其妙了，什么时候在 C 中把实数称为浮点数了呢？
整数难道不是实数？是以指数形式存放在存储单元中的吗？
至于“一个实数表示为指数”这简直就不是话。哪怕作为小学生的作文这样写也不合格呀

p49~51

“3.2.5 浮点型数据”小节的主要错误

评：1.

浮点型数据是用来表示具有小数点的实数的 。

C99 中浮点类型包括实数浮点类型和复数浮点类型

2.

为什么在 C 中把实数称为浮点数呢？

没有的事情

在 C 中实数类型是指实数浮点类型和整数类型

3.

实数是以指数形式存放在存储单元中的

错。理由同上

4.

由于小数点的位置可以浮动，所以实数的指数形式称为浮点数

臆测。

5.

小数点后第一位数字不为 0 的表示形式称为规范化的指数形式

规范化的指数形式是作者随心臆造的一个似是而非且含混不清的概念。

浮点数中有“normalized floating-point number”这样的概念，但和作者所谓的“规范化的指数形式”无关。

6.

在程序以指数形式输出一个实数时，必然以规范化的指数形式输出，如 0.314159e001。

无中生有，信口开河

7.

编译系统为每一个 float 变量分配 4 个字节，

用 8 个字节存储一个 double 型数据

C 语言并没有规定浮点数据的存储空间的大小
8.
Turbo C 对 long double 型分配 16 个字节。

无中生有，信口开河
9.
表 3.4 实型数据的有关情况
……
2.3*10^-308 ……1.7*10^308

“实型数据”：概念错误
这个表实际是 IEEE 浮点数的特征
精度差的惊人：
2.3*10^-308 应为 2.225*10^-308
1.7*10^308 应为 1.79769*10^308
10.
例如 float 型变量能存储的最小正数为 1.2*10^-38，不能存放绝对值小于此值的数，例如 10^-40

不准确且自相矛盾。浮点数 0.F 的绝对值就小于 1.2*10^-38。
11.
在 C 语言中进行浮点数的算术运算时，将 float 型数据都转换为 double 类型，然后进行计算。

错！

P50

编译系统为每一个 float 型变量分配 4 个字节。
评：毫无根据的说法

P50

浮点数类型包括 float(单精度浮点型)、double（双精度浮点型）、long double（长双精度浮点型）。
评：这个说法证明老谭对 C99 还处于根本没入门的地步。
然而他居然在内容简介中号称“按照 C 语言的新标准 C99 进行介绍。”

P50

Turbo C 对 long double 型分配 16 个字节。
评：信口开河

P50

3.2.5 浮点型数据
……编译系统为每一个 float 型变量分配 4 个字节，……
……为了扩大能表示的数值的范围，用 8 个字节存储一个 double 型数据，……
评：C 语言并没有规定浮点数据的存储空间的大小

P51

float 型变量能存储的范围见图 3.12。
……
即数值可以在 3 个范围内：（1）-3.4×10^38 到-1.2×10^-38；（2）0；（3）1.2×10^-38 到 3.4×10^38；
评：图 3.12 画的很蠢
它和这段文字给了读者一个误导，就是 0 是孤立的而另外两段区间是连续的

P51

3.2.6 怎样确定常量的类型

在程序中出现的常量是要存放在内存单元中的
评：至少是不严格的。

P51

从常量的表示形式可以判断其类型
评：123456 是什么类型单从其表示形式上是判断不出的

此外，这本书从头到尾没有介绍各种常量的写法
这也算是个“独创”了
如果常量都不会写
读者怎么学习写代码？

P51

不带小数点的数值是整型常量
评："123E4"带不带小数点？是整型常量吗

P51

整型常量。不带小数点的数值是整型常量，但应注意其有效范围。如在 Turbo C 中，系统为整型数据分配 2 个字节，其表值范围为-32768~32767，如果在程序中出现数值常量 23456，系统把它作为 int 型处理，用 2 个字节存放。如果出现 49875，由于超过 32768，,2 个字节放不下，系统会把它作为长整型（long int）处理，分配 4 个字节。在 Visual C++中，凡在-2147483648~2147483647 之间的不带小数点的数都作为 int 型，分配 4 个字节，在此范围外的整数，而又在 long long 型数的范围内的整数，作为 long long 型处理。

评：一系列的逻辑错乱

首先，“整型常量”中的“整型”显然是一个集合名词，而“整型数据”中的“整型”是另一个概念，似乎只能理解为 int 类型。这违反了形式逻辑中的同一律

其次，int 类型的表示范围和它为 2 个字节没有必然的因果关系

“由于超过 32768”更是胡说八道

“2 个字节放不下，系统会把它作为长整型（long int）处理，分配 4 个字节”表明作者根本不清楚整数常量的类型划定原则

“在 Visual C++中，凡在-2147483648~2147483647 之间的不带小数点的数都作为 int 型”，事实上根本不存在负常量

“在此范围外的整数，而又在 long long 型数的范围内的整数，作为 long long 型处理”，这是胡扯

P51

C 编译系统把浮点型常量都按双精度处理，分配 8 个字节。

注意：C 程序中的实型常量都是双精度浮点型常量。

评：从这里可以看出，老谭根本不懂得“浮点型”、“双精度”、“实型”这三个概念的区别

概念混乱不清是老谭这本书的一个重要特色

至于“分配 8 个字节”云云更是属于不懂得 C 语言的基本原理

P51~52

float a=3.14159;

……可以容忍这样的“警告”，使程序接着进行连接和运行。

评：容忍“警告”的程序员是不可容忍的

3.2.7 运算符和表达式

P52

1.基本的算术运算符

P53

……两个实数相除的结果是双精度实数。
评：没这回事。
3.F/5.F 的值就不是 double
3.L/5. 也不可能是 double

更荒谬的是
这句话明显地把整数从实数中给开除了
老谭征求过数学家的意见没
人家答应不

P53

两个整数相除的结果是整数，如 5/3 的结果值为 1，舍去小数部分。但是如果除数或被除数中有一个为负值，则舍入的方向是不固定的。
评：老谭要是在前面不吹嘘他这本书是按照 C99 标准介绍的
我还真不好批他

P53

%运算符要求参加运算的运算对象（即操作数）为整数，结果也是整数。如 8%3，结果为 2。
评：除数或被除数中有负值的情况这回压根给忘了

P53

2. 自增、自减运算符
评：最主要的是两个问题
1.对运算的介绍不全面（“作用是使变量的值加 1 或减 1”）
2.四个运算符当成了两个来介绍

P53

自增运算符(++)和自减运算符(--)只能用于变量，而不能用于常量或表达式
评：变量本身就是表达式
怎么可以说不能用于表达式呢

再给老谭看一个
```
int *p = (int *)malloc( sizeof (int) );
*p=6;
++*p;
```

"*p"难道不是表达式？

P53

使用++和--运算符时，常常会出现一些人们想不到的副作用，如 i+++j，是理解为(i++)+j 呢？还是 i+(++j)呢？

评：1.没出现过什么想不到的副作用啊。如果想不到，是没有正确理解这些运算。而没能正确理解，那首先是因为教科书本身就没讲正确，没讲清楚

2.后面的例子并不能支持“出现一些人们想不到的副作用”

3.“副作用”这个术语在 C 语言中是有特定含义的，用在这里显然是概念糊涂错乱

P53

熟练的程序开发人员喜欢在使用++和--运算符时，采取一些技巧，以体现程序的专业性，建议初学者慎用。

评：听起来专业人员特别喜欢炫耀技巧。没这回事情。专业人员只是恰当地使用运算符罢了

要初学者慎用，是在掩饰作者自己的一知半解，逃避向初学者讲明白这些运算的责任。

这是一种不懂装懂式的装腔作势

P53

自增（减）运算符常用于循环语句中，使循环变量自动加 1；也用于指针变量，使指针指向下一个地址。

评：没有的事情。大概是老谭自己只晓得这两种用法

“循环变量自动加 1”的说法很搞笑，什么叫“自动加 1”？怎么“自动”了？老谭给说说什么样不是“自动加 1”

指针指向地址的说法也是错误的

P54

3.算术表达式和运算符的优先级与结合性

……在表达式求值时，先按运算符的优先级别顺序进行，例如先乘除后加减。

评：这完全是对优先级的歪曲

C 语言里根本就没有先乘除后加减这回事

运算次序是一种 Unspecified behavior

P54

“自左至右的结合方向”又称“左结合性”，即运算对象先与左面的运算符相结合。

评：这就不是一般的胡扯了

估计是喝高了才敢这么扯

P54

若有 a=b=c,按从右到左顺序，先把变量 c 的值赋给变量 b，然后把变量 b 的值赋给 a
评：这是对表达式“a=b=c”的错误解释
完全不懂得 C 语言最基本的运算规则
这会使学习者误入歧途，永远学不懂 C 语言

P54

关于“结合性”的概念是在其他一些高级语言中是没有的，是 C 语言的特点之一
评：把优先级和结合性都讲错了不说
又信口雌黄地胡说其他高级语言没有“结合性”的概念
居然把“结合性”说成了 C 语言的特点，简直是胡扯

不过这段胡扯倒是和把“a=b=c 的意思是 a=(b=c)”说成“是 C++的观念”有异曲同工之妙

P54

如果在一个运算对象两侧的运算符的优先级别相同，如 a-b+c，则按规定的“结合方向”处理。
评：这句话蠢就蠢在那句“如果”
它好像是在暗示如果在一个运算对象一侧的运算符的优先级别相同，则不按规定的“结合方向”处理
实际上这个问题和运算符在运算对象的两侧还是一侧根本没有关系
只要是优先级相同，那么总是按结合性处理的

也就是说
那句蠢话相当于说：如果明天下雨，那么明天就是星期一
但就连弱智都懂得，明天是不是星期一和明天是否下雨是没什么关系的

P54

算术运算符的结合方向都是“自左至右”
评：这简直是自己往自己脸上扇了一记大耳光
因为在该书的 52 页
表 3.5 最常用的算术运算符
中
老谭明确无误地把一元 + 和一元 - 运算符都列为了“最常用的算术运算符”
难道一元+和一元-的结合性也是“自左至右”的吗？
（顺便说一句，不知道在老谭眼里，哪些算术运算符不是“最常用的”算术运算符）

P54

3.算术表达式和运算符的优先级和结合性

用算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式。运算对象包括常量、变量、函数等。例如，下面是一个合法的 C 算术表达式：

a*b/c-1.5+'a'

C 语言除了规定了运算符的优先级外，还规定了运算符的结合性。在表达式求值时，先按运算符的优先级别顺序执行，例如先乘除后加减。如表达式 a-b*c，b 的左侧为减号，右侧为乘号，而乘号的优先级高于减号，因此，相当于 a-(b*c)。

如果在一个运算对象两侧的运算符的优先级别相同，如 a-b+c，则按规定的“结合方向”处理。C 语言规定了各种运算符的结合方向（结合性），算术运算符的结合方向都是“自左至右”，即先左后右，因此 b 先与减号相结合，执行 a-b 的运算，然后再执行加 c 的运算。“自左至右的结合方向”又称“左结合性”，即运算对象先与左面的运算符结合。以后可以看到有些运算符的结合方向为“自右至左”，即右结合性（例如，赋值运算符，若有 a=b=c，按从右到左的顺序，先把变量 c 的值赋给变量 b，然后变量 b 的值赋给 a）。关于“结合性”的概念在其他一些高级语言中是没有的，是 C 语言的特点之一，希望能弄清楚。附录 D 列出了所有运算符以及它们的优先级别和结合性。

评：关于“算术表达式”、“优先级”和“结合性”的胡扯

首先，来看一下引用部分小标题里的第一个词——“算术表达式”。

这个词很给人一种“亲切感”特别有迷惑力，然而它确是一个模糊的、似是而非而且毫无用处的概念。

据我所知，这个词是老谭自创的。C 语言中并没有这样的概念。

C 语言中只有算术类型（arithmetic types）和算术运算符（arithmetic operators）这样的概念，并没有“算术表达式”这种概念。

没有这样的概念，难道不可以自己创造概念吗？当然可以。但必须知道的是，创造概念是有前提的：

创造者要给出概念的定义；

概念要科学严谨；

这个概念有用，或者方便简洁地描述了一个道理，或者帮助别人认识了一类现象或规律。

这样才可以创造新概念。

不满足这三个前提，自创概念不是吃饱了撑的就是假行家故弄玄虚的蒙人行为。考察一下“算术表达式”这个概念。

作者给出了定义：“用算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式。”

（很好，老谭自创概念不给定义的例子比比皆是，这次很有进步）

然而，这个概念并不科学，也不严谨。为什么这么说呢？简单地考察一下下面的表达式就会知道了：

1+(2+3)

在这里，如果把“+”说成是“连接”“操作数”还是勉强说得过去的，但是“()”的意义则绝对不是为了“连接”“操作数”。“()”的意义是为了表明其内部的子表达式作为一个整体——仿佛一个独立的操作数参与运算，这与“连接”是八竿子打不着的。再比如“(1)”这个表达式中，“()”连接了什么呢？其实它什么也没连接，它只表明它与其扩起来的部分是一个完整的整体而已。

所以说，这里的“()”是一种“界定”(to delimit)范围的符号，把它与表示运算的运算符并列在一起是很荒唐的。

作者接着补充道：“运算对象包括常量、变量、函数等”。

这就让人迷惑不解了，这里的“函数”究竟是指的什么呢？是“函数调用”还是“函数名”本身呢？如果是“函数调用”，就成了自己打自己耳光，因为函数调用一定是通过函数调用运算符“()”实现的（注意这里这个"()"不是括号），这和作者在前面说的“用算术运算符和括号将运算对象（也称操作数）连接起来的”自相矛盾。如果这里的所谓“函数”是“函数名”的话，总所周知，“函数名”这种数据类型怎么可能进行算术运算呢？

所以必须感谢作者的及时补充，他成功地使用“函数”这个词戳穿了他自己创造的伪概念——“算术表达式”，这个概念是经不起推敲的。

紧接着的“例如”也很成问题。权且承认他给出“a*b/c-1.5+'a' ”是合法的，但是这个表达式即不是“用算术运算符和括号将运算对象（也称操作数）连接起来的”（因为没有括号），“运算对象”也不“包括常量、变量、函数”（“函数”在哪？），作者举例到底想说明什么呢？无厘头么！

或许，有人很不以为然：“括号”和“函数”都是随手写的，不算大错。好，至少现在你已经承认这书很不严谨了。问题是我没提到这些错误之前，你发现这些错误了吗？就算你也和我一样发现了这些错误，那些把这本书作为教材的初学者怎么可能发现这些错误呢？你老人家胡写一通不要紧，你让那些初学者情何以堪呢？

又或许有人觉得这段文字只要像下面那样修改一下就可以了

“用算术运算符和将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式。运算对象包括常量、变量等。例如，下面是一个合法的 C 算术表达式：

a*b/c-1.5+'a' ”

Sound good！这段文字似乎基本没什么问题了。然而我们还是要问，这个自创的概念有什么用吗？继续往下看。下面一段文字是

“C 语言除了规定了运算符的优先级外，还规定了运算符的结合性。在表达式求值时，先按运算符的优先级别顺序执行，例如先乘除后加减。如表达式 a-b*c，b 的左侧为减号，右侧为乘号，而乘号的优先级高于减号，因此，相当于 a-(b*c)。”

我在读这一段第一句话的时候被骇了一大跳。这句型，简直太惊人了。前面压根没提过“优先级”，抽冷子就来了一句“C 语言除了规定了运算符的优先级外，还……”，有这么说话的吗？如果连话都说不利索，但却非要写在教科书里，您老人家又成天把这个“1100 万”挂在嘴上，您说这和一只著名的苍蝇成天炫耀自己污染过很多食物有区别吗？

也许有人觉得这只是语文问题，不是 C 语言方面的问题。好吧，我让步！问题是这段文字和前面作者自创的“算术表达式”没有任何关系，没有人能否认这一点吧？！

再看引用文字的最后一段。这段文字同样和“算术表达式”这个伪概念没有任何关系。唯一可能牵强地和“算术表达式”扯上一点关系的是“算术运算符的结合方向都是‘自左至右’”，然而这句话本身就是错误的，因为作者在书中的第 52 页明确地说明一元“+”、“-”都是算术运算符，然而我们都知道一元“+”、“-”的结合性是“自右至左”，更何况“算术运算符”和“算术表达式”分明就是两个不同的概念。

而且，优先级和结合性的规律是对所有运算符而言的，算术运算符并没有什么特殊的优先级和结合性规律；而且所谓的“算术表达式”在这方面也没有任何特殊性而言，创造“算术表达式”比画蛇添足还要无聊。不仅无聊，而且有害。这个概念甚至无法比喻成懒婆娘的裹脚布，最多只能称得上是懒婆娘丢弃的裹脚布而已。

问题出来了，谭大爷为什么要自创这个不伦不类而且漏洞百出的伪概念——“算术表达式”，他到底想说什么？我估计他自己也不清楚。

话说到这里，我想我已经证明了这段引文的“胡扯性”。但是，根据我的经验，这时候一定会有无耻之徒跳出来无赖地狡辩：“毕竟还介绍了优先级和结合性么”——没办法不让他们跳出来，“卑鄙是卑鄙者的通行证”。然而，在这里，即使是这种最无耻的狡辩也是不可能得逞的，因为这段引文对“优先级”和“结合性”的讲解也是错的！

关于优先级，作者是这样写的：

“在表达式求值时，先按运算符的优先级别顺序执行，例如先乘除后加减。如表达式 a-b*c，b 的左侧为减号，右侧为乘号，而乘号的优先级高于减号，因此，相当于 a-(b*c)。”

这段文字的错误在于作者把运算符的优先级和表达式求值的执行次序混为了一谈。尤其是“先乘除后加减”更是对优先级的曲解。实际上，优先级和表达式求值的执行次序基本上是不相干的两回事情。

优先级的意义在于表明表达式的含义（结合性也是），而求值的执行次序则是编译器的自选动作。只要不违反表达式的含义，编译器可以按照自己的爱好安排求值次序，编译器也没有义务告诉你它是按照什么次序求值的。这种自选动作叫 implementation-defined behavior。

举例来说,对于表达式“ 1+2-3*4 ” ，“*”的优先级高于“-”，其意义在于表明减去的是（3*4）而不是减 3。只要你清楚地表明了这一点，你的任务就完成了。至于计算次序，编译器至少有两种选择：

1） 1+2-3*4 => 1+2-12 => 3-12 => -9

2） 1+2-3*4 => 3-3*4 => 3-12 => -9

按照外交部的惯用句型来说就是，怎么选择求值次序是编译器自己的事，程序员无权干涉（甚至无权了解，好奇也不行，编译器没义务告诉你）。 程序员的责任在于把表达式的含义写清楚，写正确；编译器的义务仅仅在于给出正确的结果而已。至于算出结果的过程（次序），对不起，关您什么事啊？您管的也太宽了吧。

总之，“优先级”的意思就是优先级高的运算符先选“对象”——运算对象。谭大爷明显是不明白这点小道理，可笑地把优先级和求值次序“绑定”到了一起，一相情愿地拉郎配。

如果优先级高的运算符选完了“对象”，剩下的运算符优先级相同，怎么办呢？

答案也简单，按照结合性挑选“对象”，如果是从左到右的结合方向，就左边的先挑，依次向右进行；如果是从右到左，则顺序相反。例如：

1+2-3*4

由于“*”的优先级最高，所以先挑运算对象，表达式的含义为

1+2-(3*4)

剩下的两个运算符“+”和“-”的优先级相同，所以看结合性，这两个运算符的结合性是从左到右，因此左面的先挑，表达式的含义可以进一步明确为

(1+2) - (3*4)

最后，可以确定“-”的运算对象分别为 (1+2) 和 (3*4)。

这就是优先级和结合性的全部含义。如此而已，非常简单。再来看看老谭是怎么讲的。

“如果在一个运算对象两侧的运算符的优先级别相同”，看到这句我差点没吐出来。老谭居然给来了个“如果”“运算对象”“两侧”，这跟优先级、结合性有什么关系吗？如果运算符在运算对象的同一侧怎么办？比如 - - i （不是"--",两个“-”之间有空格），这应该怎么办呢？再比如 a[3]，那个 3 应该算是在运算符的哪一侧呢？

所以看老谭的书，如果你不能发现他的愚蠢，那就只剩下了两种可能：1.你自己愚蠢；2.你正在变得愚蠢。

再往下看：

“算术运算符的结合方向都是'自左至右”，即先左后右'”，这句话本身就是错误的，前面已经提到过。

“因此 b 先与减号相结合”，这句更可笑，纯粹是望文生义式的妄断臆测。老谭把左结合性理解成了“运算对象先与左面的运算符结合”，简直是荒唐透顶。结合性，即使从字面上来说，也不是简单地指操作数与左面或右面的运算符相“结合”。简单举个例子就说明老谭的荒谬，"[]"运算符的结合性是从左至右，

那么表达式 a [b[0]] 中 ，无论是 a 或是 b，怎么与左面的运算符结合？所以“运算对象先与左面的运算符结合”这样的昏话绝对不可能是荒谬，只可能是荒谬透顶。

“即右结合性（例如，赋值运算符，若有 a=b=c，按从右到左的顺序，先把变量 c 的值赋给变量 b，然后变量 b 的值赋给 a）。”，这里也是错的。根据结合性，a=b=c 的含义是 a=(b=c) ，也就是说是把 b=c 的值赋给 a，而不是老谭所说的“然后变量 b 的值赋给 a”。

“关于“结合性”的概念在其他一些高级语言中是没有的，是 C 语言的特点之一”，这是井底之蛙坐井观天式思考的结果，根本不用反驳。

“希望能弄清楚”，希望很好，问题是您老人家自己弄清楚没有啊？您总这么瞎忽悠，让学习者怎么弄清楚啊？

“附录 D 列出了所有运算符以及它们的优先级别和结合性。”，第一，附录 D 并没有列出“所有”的运算符，第二，列出的部分不少是错误的。

P54

4.不同类型数据的混合运算

……如果一个运算符的两侧的数据类型不同，则先自动进行类型转换，使两者具有同一类型，然后进行运算。

评：若有

int a[1];

那么对于表达式

*(a + 0)

显然“+”“两侧的数据类型不同”。

老谭能否给解释一下，如何“自动进行类型转换”？使两者具有了什么样的“同一类型”？

P54

(1)+、-、*、/运算的两个数中有一个数为 float 或 double 型，结果是 double 型，因为系统将所有的 float 型数据都先转换为 double 型，然后进行运算。

评：一句不长的话错了两次。

1.“+、-、*、/运算的两个数中有一个数为 float 或 double 型，结果是 double 型”

老谭既然在前面吹嘘这本书是“按照 C 语言的新标准 C99 进行介绍”

难道不知道 C 语言有复数类型吗？

如果“+、-、*、/运算的两个数中有一个数为 float 或 double 型”，另一个是复数类型，“结果是 double 型”吗？

就算你根本不懂得 C99

虚张声势地出来蒙人

long double 类型可是几十年前就有了

如果“+、-、*、/运算的两个数中有一个数为 float 或 double 型”，另一个是 long double，“结果是 double 型”吗？

2.“系统将所有的 float 型数据都先转换为 double 型，然后进行运算。”，简直是胡扯

P54

(2)如果 int 型与 float 或 double 型数据进行运算，先把 int 型和 float 型数据转换为 double 型，然后进行运算。结果是 double 型

评：第一次读完这个(2)，我愣住了，简直不敢相信
再仔细看一遍，发现这个(2)真的是很(2)
因为什么呢
因为(1)已经说过“+、-、*、/运算的两个数中有一个数为 float 或 double 型”
现在(2)居然又开始大谈特谈“如果 int 型与 float 或 double 型数据进行运算”
车轱辘话来回说是老谭的一个特点
但刚说完就 repeat 一遍还是罕见的
中等程度的神经错乱都写不出来这种东西

而且，char 、long 等与 float 或 double 型数据进行运算的情况，阙如

这段中的严重错误就不谈了
与错乱相比，错误简直不算什么了不起的事情

P54

(3)字符(char)型数据与整型数据进行运算，就是把字符的 ASCII 代码与整型数据进行运算。

评：（暂且不谈 C 语言的执行环境未必使用 ASCII 码的问题）
这里的“整型数据”是个莫名其妙的概念

在该书的 43 页写到

整型类型	基本整型 （int）
	短整型 （short int）
	长整型 （long int）
	*双长整型 （long long int）
	字符型 （char）
	*布尔型 （bool） [注：这里的 bool 也是错误的]

所谓“字符(char)型数据与整型数据进行运算”
说的到底是哪种"整型类型"呢？
况且这根本就是一个不必要的条件限制
难道 char 类型数据 与 double 类型数据 做加法运算时那个 char 类型数据不是字符编码吗？
它不是字符的编码还能是什么别的东西吗?

P54

(3)字符(char)型数据与整型数据进行运算，就是把字符的 ASCII 代码与整型数据进行运算。如：12 + 'A'，

由于……
评：前面说的是“字符(char)型数据与整型数据进行运算”
立刻就拿出个两个 int 类型数据的加法来举例
很明显
老谭根本不知道 'A' 的类型究竟是什么

P54

字符数据可以直接与整型数据进行运算。
评：不知所云。
什么叫“直接”进行运算？
是否还有“间接”进行运算？

P54

如果字符型数据与实型数据进行运算，则将字符的 ASCII 代码转换为 double 型数据，然后进行运算。
评：1. “实型数据”，应为（实）浮点型数据。因为在 C 语言中 real types 是整数类型与（实）浮点类型的合称。
2.“字符的 ASCII” ，运行环境中的字符不一定是用 ASCII 码表示的。况且，即使运行环境也使用 ASCII 码，字符数据中存储的也不一定是 ASCII 码。比如，字符数据可能是一个负值
3.“转换为 double 型数据” ，完全无视了 integer promotions。而且最终可能转换成三种而不是只有 double 这一种。
紧接着
以上的转换是编译系统自动完成的，用户不必过问。
不知所云。

P54~55

分析下面的表达式，假设已指定 i 为整型变量，值为 3，f 为 float 型变量，值为 2.5，d 为 double 型变量，值为 7.5 。
10+'a'+i*f-d/3
编译时，从左到右扫描，运算次序为：
①进行 10+'a'的运算，……
②由于“*”比“+”优先级高，先进行 i*f 的运算。先将 i 和 f 都转成 double 型，……
③整数 107 与 i*f 的积相加。
④进行 d/3 的运算……
⑤将 10+'a'+i*f 的结果与 d/3 的商 2.5 相减，……
评：最雷人的是：“编译时……运算次序为”
编译时就运算，那运行时计算机干些什么呢？

“②由于“*”比“+”优先级高，先进行 i*f 的运算”
可是“进行 10+'a'的运算”为什么先做了呢，

难道“*”比第 1 个“+”优先级低、比第二个“+”优先级高吗？

“先将 i 和 f 都转成 double 型”
错的。

小学语文问题：
“i*f 的积”，
“将 10+'a'+i*f 的结果与 d/3 的商 2.5 相减”

P55

例 3.3 给定一个大写字母，要求用小写字母输出。
……，字符数据以 ASCII 码存储在内存的
……

```
char c1,c2;
c1='A';
c2=c1+32;
printf("%c\n",c2);
printf("%d\n",c2);
```

……
评：1.题目中的“要求用小写字母输出”莫名其妙。语文问题
2.“字符数据以 ASCII 码存储在内存的”，语文问题，而且观点错误。
3.c2=c1+32; 那个 32 是再拙劣不过的写法
4.既然“要求用小写字母输出”，printf("%d\n",c2);属于画蛇添足。程序完成不应该完成的功能，也是一种错误。

P55

一个字符数据既可以以字符形式输出，也可以以整数形式输出
评：但没有讲其中的道理
更可笑的是图 3.13
竟然表明用%c 和%d 可以用来输出一个字节的数据(01100001)
属于外行的似是而非
对学习者有极大的误导

P56

5.强制类型转换运算符
……

```
(double)a          (将 a 转换成 double 类型)
```

评：C 语言居然有这功能？

P56

……例如：

```
a=(int)x
```

如果已定义 x 为 float 型变量，a 为整型变量，进行强制类型运算(int)x 后得到一个 int 类型的临时值，它的值等于 x 的整数部分，把它赋给 a，注意 x 的值和类型都未变化，仍为 float 型。该临时值在赋值之后就不再存在了。
评：可怜的老谭，看来根本就不理解表达式的意义。

居然必须通过臆造一个“临时值”才能理解类型转换表达式
处于还没透彻理解表达式意义的水平下，让他写 C 语言书实在难为他了
“注意 x 的值和类型都未变化”，简直是废话，x 的类型怎么可能变化呢？任何情况下也没这个可能啊
“它的值等于 x 的整数部分”，是错误的，至少是不严谨的
“该临时值在赋值之后就不再存在了。”，对无中生有的“临时值”所做的无中生有的臆测。

P56

如%运算符要求其两侧均为整型量，若 x 为 float 型，则 x%3 不合法，必须用(int)x%3。……因此先进行(int)x 的运算，得到一个整型的中间变量。
评：“中间变量”属于老谭的“发明”
但老谭显然忘了他自己在前面是怎么说的了，
在 41 页：“变量代表一个有名字的、具有特定属性的一个存储单元”、“变量必须先定义，后使用”……
于是自己了打自己一耳光

3.3C 语句

P57

语句的作用是向计算机系统发出操作指令，要求执行相应的操作
评：表达式才是要求操作

P57

声明部分不是语句，只是对有关数据的说明
评：声明不是对数据的说明

P58

（2）函数调用语句。……
（3）表达式语句
评：尽管老谭辩解说这是为了“理解”和“使用”
但实际上这表明他自己的不理解和不会使用
不得不按照 BASCI 或 FORTRAN 的方式来理解 C
从书中的代码来看，除了构成表达式语句，他确实不会使用函数调用表达式

P59

在 C 程序中，最常用的语句是：赋值语句和输入输出语句。其中最基本的是赋值语句。程序中的计算功能

大部分是由赋值语句实现的，几乎每一个有实用价值的程序都包括赋值语句。有的程序中的大部分语句都是赋值语句。

评：我简直无语
更雷人的是
书的另一处居然写着：
“C 语言本身不提供输入输出语句”（p12）
让我们究竟相信你哪个说法呢

在 C 程序中，最常用的语句是：赋值语句和输入输出语句。其中最基本的是赋值语句。程序中的计算功能大部分是由赋值语句实现的，几乎每一个有实用价值的程序都包括赋值语句。有的程序中的大部分语句都是赋值语句。
评：笑喷了
这是史上最强悍最露骨的无知无畏——什么他都敢说，而且都是错话和废话

P59

printf("a=%f\tb=%f\t%f\n",a,b,c);
评：

P60

赋值符号=就是赋值运算符，它的作用是将一个数据赋给一个变量。如 a=3 的作用是执行一次赋值操作（或称赋值运算）。把常量 3 赋给变量 a。也可以将一个表达式的值赋给一个变量。
评：

P60

a+=3 等价于 a=a+3
评：不很严谨
不过指望老谭能理解二者的差异是不可能的事情
算了

P60

①a+=b (其中 a 为变量，b 为表达式)
……
注意：如果 b 是包含若干项的表达式，则相当于它有括号。例如，……
评：例如
a+=3,2
就相当于

a+=(3,2) 吧？

结论：老谭根本不懂得表达式
不懂表达式根本连 C 的门都还没摸到
奇怪的是
居然有那么多人自称跟着老谭入了门

P61

凡是二元（二目）运算符，都可以与赋值符一起组合成复合赋值符。
评：好一个“凡是”啊
这口气听起来就像权威
不过
“==”应该是“二元（二目）运算符”吧
请问老谭
“=== ”是什么“复合赋值符”呢

P61

C 语言采用这种复合运算符，一是为了简化程序，使程序精炼，二是为了提高编译效率，能产生质量较高的目标代码。
评：什么叫敢想敢说？
这就是

P61

左值的意思是它可以出现在赋值运算符的左侧，它的值是可以改变的
评：记得 supermegaboy 就这个问题给老谭上过一课
看来老谭还是没懂
有一点进步
已经不说左值只能是变量而不能是表达式了

P61

执行表达式“a=(b=5)”,就是执行 b=5 和 a=b 两个赋值表达式。
评：

凡是二元（二目）运算符，都可以与赋值符一起组合成复合赋值符。

评：是没睡醒还是在信口开河。
如此说来

“=”是二元运算符，它可以与赋值符一起组合成复合赋值符“==”  、

P62

赋值表达式也可以包括复合的赋值运算符。例如：
a+=a-=a*a
也是一个赋值表达式。如果 a 的初值为 12，此赋值表达式的求解步骤如下：
①先进行“a-=a*a”的运算，它相当于 a=a-a*a，a 的值为 12-144=132。
④ 再进行“a+=-132”的运算，相当于 a=a+(-132)，a 的值为-132-132=-264
评：这个例子表明老谭不知道写代码的最基本准则
这种错误在大学教材里存在了二十年
是一种国耻

也是程序员之耻
人们有理由问
这个国家的程序员们究竟懂不懂 C 语言

P62

printf("%d",a=b);
如果 b 的值为 3，则输出 a 的值(也是表达式 a=b 的值)为 3。
评：这句话对初学者的误导几乎是致命的。

P63

（4）字符型数据赋给整型变量时，将字符的 ASCII 代码赋给整型变量。如：
i='A';;
评：1.字符型数据存储的未必是 ASCII 码
2.'A'根本不是字符型数据
3.“;;”

P63

（5）将一个占字节多的整型数据赋给一个占字节少的整型变量或字符变量（例如把占 4 个字节的 int 型数据赋给占 2 个字节的 short 变量或占一个字节的 char 变量）时，只将其低字节原封不动地送到被赋值的变量（即发生“截断”）。
评：这是错误的
估计这是老谭根据某个编译器下的运行结果自己总结出来的

P64

例如：

if((a=b)>0) max=a;

评：在这个地方之前根本就没介绍过“>”运算，更没介绍过 if 语句。这例子很差劲

P64

先进行赋值运算(将 b 的值赋给 a)，然后判断 a 是否大于 0

评：对“if((a=b)>0) max=a; ”的解说

是错误的。

解说所描述的是 a=b,a>0 这个表达式而不是(a=b)>0 这个表达式

P64

6.变量赋初值

……

也可以被定义变量的一部分赋初值。

评：语文问题

冷一看还以为老谭在说结构体呢

什么叫“变量的一部分”？

P65

一般变量初始化不是在编译阶段完成的(只有在静态存储变量和外部变量的初始化是在编译阶段完成的)，而是在程序运行时执行本函数时赋予初值的，相当于执行一个赋值语句。
评：编译时变量存在吗？

> “只有在静态存储变量和外部变量的初始化是在编译阶段完成的”，语文问题

3.4 数据的输入输出

p66

该 scanf 函数表示从终端输入的 3 个数据……。双撇号内用%lf 格式声明，表示输入的是双精度型实数。
评：scanf 函数的输入来自标准输入设备

> “表示输入的是双精度型实数”是很严重的误导。这种似是而非的“省劲”说法，会给后面的学习带来很多困惑和麻烦。
> 标准输入设备提供的只是一些字符而已，根本就没有什么双精度实数。

P67

（1）所谓输入输出是以计算机主机为主体而言的。从计算机向输出设备（如显示器、打印机等）输出数据称为输出，从输入设备（如键盘、磁盘、光盘、扫描仪等）向计算机输入数据称为输入，如见图 3.17 所示。

评：总体来说是废话

给人的感觉是输入输出设备明显已经不是计算机的组成部分了，不知道冯.诺依曼是否知道他的体系已经被老谭给强力摧毁

P67

（2）C 语言本身不提供输入输出语句，

评：我就不明白老谭为什么反复说这句话，从开头到这里至少说四五次了。是不是没话找话呢

P67

例如 printf 函数和 scanf 函数。读者在使用它们时，千万不要误认为它们是 C 语言提供的“输入输出语句”

评：除了老谭自己的书上，我从来没见过把 printf 函数和 scanf 函数调用叫做输入输出语句的

P67

C 提供的标准库函数以库的形式在 C 的编译系统中提供，它们不是 C 语言文本中的组成部分。

评：前半句就是个病句，搞不清到底是谁“提供”

后半句不知所云。什么叫“C 语言文本”

P67

不把输入输出作为 C 语句的目的是使 C 语言编译系统简单精炼，因为将语句翻译成二进制的指令是在编译阶段完成的，没有输入输出语句就可以避免在编译阶段处理与硬件的有关问题，可以使编译系统简化，而且通用性强，可移植性好，在各种型号的计算机和不同的编译环境下都能适用，便于在各种计算机上实现。

各种 C 编译系统提供的系统函数库是各软件公司编译的，它包括了 C 语言建议的全部标准函数。

评：武断

P67

各种 C 编译系统提供的系统函数库是各软件公司编译的，它包括了 C 语言建议的全部标准函数，还根据用户的需要补充一些常用的函数，已对它们进行了编译，成为目标文件（.obj 文件）。它们在程序连接阶段与

源程序经编译得到的目标文件（.obj 文件）相连接，生成一个可执行的目标程序（.exe 文件）。如果在源程序中有 printf 函数，在编译时并不把它翻译成目标指令，而是在连接阶段与系统函数库相连接后，在执行阶段中调用函数库中的 printf 函数。

评：1.头一次听说库是 .obj 文件，不知道老谭是否在编译软件中发现了 printf.obj 啊？

2.“在编译时并不把它翻译成目标指令”，那还叫编译吗？不翻译难道保留源代码？

3.最雷人的是“在执行阶段中调用函数库中的 printf 函数”，老谭自己知不知道自己在说什么啊？对自己根本不了解的东西难道可以这样信口开河吗？执行阶段怎么调用库？难道发行可执行文件必须像 VB 那样搭配一个库？

P68

#include 指令都放在程序文件的开头
评：未必

P69

printf 函数(格式输出函数)用来向终端(或系统隐含指定的输出设备)输出若干个任意类型的数据。
评：这个属于昏话。
首先“终端”这词是模糊不清的、不准确的
“系统隐含指定的输出设备”更是不知所云的昏话
至于“若干个任意类型的数据”更是胡扯

printf()函数的功能是在标准输出设备上输出若干字符

P69

“输出表列”是程序需要输出的一些数据，可以是常量、变量或表达式。
评：这个说法很滑稽
常量、变量都是表达式
老谭居然整出了个“常量、变量或表达式”
和“香蕉、苹果或水果”如出一辙
这东西作为小学作文也不及格吧

此外“表列”这个词也很奇怪
注意了一下
这个怪异的说法并非笔误而是在多处一致地这样写

P70

如果整数比较大，则把它的最后一个字节的信息以字符形式输出。如：
int a=377;
printf("%c",a);
评：这个说法不准确，实际上是通过类型转换

P70

（1）d 格式符
……在输出时，按十进制整型数据的实际长度输出，正数的符号不输出。
评：1.既然谈到了数据的类型，就谈不到什么“十进制”
2.正数的符号可以输出

P73

（2）o 格式符。以八进制整数形式输出。将内存单元中的各位（0 或 1）按八进制形式输出，因此输出的数值不带符号，即将符号位也一起作为八进制数的一部分输出。例如：
int a=-1;
printf(“%d\t%o\n”,a,a);
运行时输出：
-1　　37777777777

评：用%o 输出 int 类型数据是未定义行为

P73~74

（5）g 格式符。用来输出浮点数，系统自动选 f 格式或 e 格式输出，选择其中长度较短的格式，不输出无意义的 0。如：
double a=12345678954321
printf("%f\t%e\t%g\n",a,a,a);
的输出结果为：
12345678954321.000000　　1.234568e+013　　1.23457e+013
可以从以上看到用%f 格式输出占 20 列，用%e 格式输出占 13 列，故%g 采用%e 格式输出。
评：问题是“double a=12345678954321”根本无法通过编译
结果究竟是怎么得到的？

即使按“double a=12345678954321;”来看，这个赋初值也是很成问题的

况且用%f 格式输出占的是 21 列

%g 格式的输出也不是按照%e 格式

P74

表 3.6 printf 函数中用到的格式符

……

d,i　　以带符号的十进制形式输出整数（正数不输出符号）

o　　以八进制无符号形式输出整数（不输出前导符 0）

x,X　　以十六进制无符号形式输出整数（不输出前导符 0x）

评：不是这样的

P74

表 3.6 printf 函数中用到的格式符

……

g,G　　选用%f 或%e 格式中输出宽度较短的一种格式，不输出无意义的 0。用 G 时若以指数形式输出，则指数以大写表示。

评：这个也是错的。

P74

表 3.7 printf 函数中用到的格式附加字符

……

评：1.不全

2.其中的 m,n 是什么以及如何用根本就没说清楚，并且有其他错误。

P75

scanf("a=%f,b=%f,c=%f",&a,&b,&c);

评：这种写法要多蠢有多蠢

简直是自虐

P76

表 3.8 scanf 函数中所用到的格式字符

u　　用来输入无符号的十进制数

o　　用来输入无符号的八进制数

x,X　　用来输入无符号的十六进制数

评：这几种格式都容许输入有符号整数

P76

例如，若 a 和 b 为整型变量，如果写成
scanf("%f%f%f",a,b,c);
是不对的。应将“a,b,c”改成“&a,&b,&c”。许多初学者常犯此错误。
评：那样是不对
　　老谭改的就对吗？

　　前面说“若 a 和 b 为整型变量”
　　后边居然莫名其妙地出来个 c
　　而且整型使用%f
　　这不是粗制滥造是什么呢

P76

如果有
scanf("a=%f,b=%f,c=%f",&a,&b,&c);
在输入数据时，应在对应的位置上输入同样的字符。即输入
a=1,b=3,c=2
评：scanf("a=%f,b=%f,c=%f",&a,&b,&c);
　　如此愚蠢的代码怎么可以写在教科书里
　　而且煞有介事地进行解释呢

P77

如果 scanf 函数为
scanf("a=%f　b=%f　c=%f",&a,&b,&c);
由于在两个%f 间有两个空格，因此在输入数据时，两个数据间应有两个或更多的空格字符。）
评：第一，两个数据间应有两个或更多的不一定是空格字符
　　第二，这种写法同样很蠢，两个%f 之间加空格干什么？

P77

(3)在用“%c”格式声明输入字符时，空格字符和“转义字符”中的字符都作为有效字符输入
评：“转义字符”中的字符
　　这个太搞了

P78

putchar(c)的作用是输出字符变量 c 的值，显然它是一个字符
评：显然老谭不知道那个 c 不是 char 类型

P78

例 3.8 先后输出 BOY 三个字符。
解题思路:定义 3 个字符变量，……

```
#include <stdio.h>
int main()
{
char a='B',b='O',c='Y';
putchar(a);
putchar(b);
putchar(c);
putchar('\n');
return 0;
}
```

评：唯一值得表扬的是 putchar('\n');
输出三个字符哪里需要用的着费那么大劲啊
printf("BOY\n");
定义变量很傻

P79

因此将一个字符赋给字符变量和将字符的 ASCII 代码赋给字符变量作用是完全相同的（但应注意，整型数据应在 0~127 的范围内）
评：完全不懂得赋值是怎么回事
所谓“将一个字符赋给字符变量”“赋给字符变量”是完全不通的说法
实际是根本不存在的情况
0~127 这个条件根本不需要
那个“整型数据”不清楚是从哪里窜出来的，前文中根本没有

P79

说明：putchar(c)中的 c 可以是字符常量、整型常量、字符变量或整型变量（其值在字符的 ASCII 代码范围内）
评：c 就是一个 int 类型数据
“其值在字符的 ASCII 代码范围内”也不对

P79

一般是显示器的键盘
评：看来我真 out 了

还真没见过这种东东 

P79

例 3.9 从键盘输入 BOY 三个字符，然后把它们输出到屏幕。
评：问题的提法非常古怪无聊
如果输入 DOG 是不是还得另外编个程序

P79

用 putchar 函数既可以输出能在显示器屏幕上显示的字符，也可以输出屏幕控制字符。
评：老谭总是自创一些莫名其妙的概念，这表明他自己概念不清
一大群幼稚的初学者跟着莫名其妙地理解，各有各的糊涂理解。有人把这叫“通俗易懂”

P79

字符类型也属于整数类型。
评：严重表扬！老谭说了一句正确的话
相对以前各版，这是一个巨大的进步

P80

这些字符先暂存在键盘的缓冲器中，
评：前所未闻

P82

可以用 printf 函数和 scanf 函数输入或输出字符。
请比较这两个方法的特点，在特定情况下用哪一种方法为宜。
评：语文问题

第 4 章 选择结构程序设计

4.1 选择结构和条件判断

P85

输入一个数，要求输出其绝对值。
评：不明确的程序功能要求

P86

例 4.1 ……
……//disc 是判别式 sqrt(b*b-4ac)
disc = b * b - 4 * a * c;
……
评：此外此例代码风格欠佳。

（1） 为提高精度以及避免在编译时出现“警告”，将所有变量定义为双精度浮点型。
评：如果是为了“避免”“警告”“将所有变量定义为双精度浮点型”表明老谭不会使用 float 类型而不得不废掉 C 语言的一种基本数据类型。
这是一种误导。选择数据类型不能根据（因为不懂得如何使用而引起的）“警告”
谭的做法是因噎废食，道无知以误人

4.1 小节的另一个缺欠是，在没介绍关系运算和 if 语句的条件下用它们写代码。

4.1 用 if 语句实现选择结构

P87

从例 4.1 可以看到：在 C 语言中选择结构主要是用 if 语句实现的。
评：从一个例子就能看出“在 C 语言中”如何如何
老谭的因果关系一向非常山奔海立，让人可惊可愕

为了实现互换，必须借助于第 3 个变量。
评：“必须”两个字过于武断
不借助第 3 个变量也可以交换变量的值

例 4.2 输入两个实数，按代数值由小到大的顺序输出这两个数。
例 4.3 输入 3 个数 a、b、c，要求按由小到大的顺序输出。

评：前一个是合格的题目
后一个不合格:"3 个数"性质不明确;a、b、c 更是多此一举

4.2 用 if 语句实现选择结构
评：这一小节最重要的缺失是没有介绍 if 语句的具体执行过程
实际上
if(表达式)语句
计算机首先计算“表达式”的值，然后根据这个值是否不为 0 选择是否执行语句
这种次序观念对于初学者来说极其重要

P89

4.2.2 if 语句的一般形式
……
if(表达式)语句 1
[else 语句 2]
if 语句中的“表达式”可以是关系表达式、逻辑表达式，甚至是数值表达式。其中最直观、最容易理解的是关系表达式。
评：这恍然大悟式的惊呼说明了老谭对 C 的理解程度
然而这里列举的并不全面（其实除了 void 类型所有表达式都可以）
而且对比不当
试问什么是“数值表达式”？
关系表达式、逻辑表达式是不是数值表达式？
如果承认关系表达式、逻辑表达式也是数值表达式
那么何谈“甚至”呢？

而“最直观、最容易理解”其实恰恰说明老谭自己对其他表达式感到难以理解，至于他能否恰当地应用，至少我是怀疑的

P90

……
else cost=0
评：讽刺的是在这页上写着“每个内嵌语句的末尾都应当有分号，因为分号是语句中的必要成分。”

(1) 整个 if 语句可以写在多行上，也可以写在一行上，如：
if(x>0)y=1 ;else y=-1;
但是，为了程序的清晰，提倡写成锯齿形式。
评：这是无厘头的废话
除了预处理命令，整个源程序都可以写在一行上，难道也告诉读者?

C 源程序的写法是统一的，if 语句并不存在任何特殊性

每个内嵌语句的末尾都应当有分号，因为分号是语句中的必要成分。
评：没这个事

```
if(1){
}
else{
}
```
就一个分号都没有

联系到该书把语句分为 控制语句、函数调用语句、表达式语句、空语句、复合语句，以及“最基本的语句——赋值语句”这样的说法
不难发现该书作者对“语句”始终缺乏准确、完整的概念

P91

（6）在 if 语句中要对给定的条件进行检查，判定所给定的条件是否成立。判定的结果是一个逻辑值“是”或“否”。
评：本来以为 89 页的那句带着“甚至”的惊呼“ if 语句中的“表达式”可以是关系表达式、逻辑表达式，甚至是数值表达式。”表明老谭的 C 语言开始入门了。现在看来，他还是在 BASIC 的“是”、“否”泥潭里面折腾着呢。
不难发现他的自相矛盾
也终于理解他前面为什么要说“甚至是数值表达式”了，这是一种初学者新发现式的惊奇。

事实上，以
if(表达式) 语句
为例

其中的“表达式”本来就是一个数值表达式，如果这个表达式的值不为 0 则执行“语句”，否则不执行“语句”
如此而已

根本就没有什么“条件”、“检查”，C 也没有“是”、“否”这样的概念

又如：判断“a>b”条件是否满足，当 a>b 时，就称条件“a>b”为“真”，如果“a≤b”，则不满足“a>b”条件，就称此时条件“a>b”为假。————
评：这讲的不是 C 语言
在 C 语言中“>”是一种运算符
a>b 的值可能为 1，也可能为 0

4.3 关系运算符合关系表达式

P91

其中的“>”是一个比较符
评：老谭总喜欢发明这些古怪且没有什么意义的“新术语”

例如，a>3 是一个关系表达式，大于号是一个关系运算符，如果 a 的值为 5，则满足给定的“a>3”条件，因此关系表达式的值为“真”(即“条件满足”)；如果 a 的值为 2，不满足“a>3”条件，则称关系表达式的值为“假”
评：似是而非的误导。
http://bbs.chinaunix.net/thread-2297714-1-2.html
这个帖子说明了这种误导的后果

实际上“真”、“假”这些概念根本不属于 C 语言
总是借助多余的概念来理解 C 所表明的是并不真正理解 C

在 C 语言中，>是一种运算符
5>3 的运算结果为 1
2>3 的运算结果为 0
就这么简单。根本不需要那些画蛇添足的“真”、“假”

P92

用关系运算符将两个数值或数值表达式连接起来的式子，称关系表达式。
评：“称关系表达式”，这语文水平就不评说了
表述也极不严谨
试问“3&4”、“5|6”算不算数值表达式
用关系运算符连接起来——“ 3&4<5|6 ”,是关系表达式吗？

4.4 逻辑运算符和逻辑表达式

P92~93

4.4 逻辑运算符和逻辑表达式
有时要求判断的条件不是一个简单的条件，而是由几个给定简单条件组成的复合条件。如：“如果星期六不下雨，我去公园玩”。这就是由两个简单条件组成的复合条件，需要判定两个条件：（1）是否星期六；（2）是否下雨。只有这两个条件都满足，才去公园玩。又如“参加少年运动会的年龄限制为 13~17 岁”，

这就需要检查两个条件：（1）年龄 age≥13（2）年龄 age≤17。这个组合条件条件是不能够用一个关系表达式来表示的，要用两个表达式的组合来表示，即：age≥13 AND age≤17。用一个逻辑运算符 AND 连接 age≥13 和 age≤17。两个关系表达式组成一个复合条件，“AND”的含义是“与”，即“二者同时满足”。age≥13 AND age≤17 表示 age≥13 和 age≤17 同时满足。这个复合的关系表达式“age≥13 AND age≤17”就是一个逻辑表达式。其他逻辑表达式可以有：

x>0 AND y>0　　(同时满足 x>0 和 y>0)

age<12 OR age>65　　(年龄 age 小于 12 的儿童或大于 65 的老人)

上面第 1 个逻辑表达式的含义是：只有 x>0 和 y>0 都为真时，逻辑表达式 x>0 AND y>0　才为真。上面第 2 个逻辑表达式的含义是：age<12 或 age>65 至少有一个为真时，逻辑表达式 age<12 OR age>65 为真。OR 是“或”的意思，即“有一即可”，在两个条件中有一个满足即可。AND 和 OR 是逻辑运算符。

用逻辑运算符将关系表达式或其他逻辑量连接起来的式子就是逻辑表达式。

评：云山雾罩，不知所云

任何合格的 C 程序员都不可能认为这是在讲 C 语言

尤其是“AND 和 OR 是逻辑运算符”这句，属于根本性的概念错误

在 C 语言中 AND 和 OR 是标识符不是运算符（在 C 语言中唯一可以做运算符的标识符是 sizeof）

连“标识符”、“运算符”这样最最的基本概念都拎不清，居然写教材?C 语言界没人啦

"用逻辑运算符将关系表达式或其他逻辑量连接起来的式子就是逻辑表达式"的错误性质见 1180 楼

P93

有 3 种逻辑运算符：与(AND)，或(OR)，非(NOT)。在 BASIC 和 Pascal 等语言中可以在程序中直接用 AND，OR，NOT 作为逻辑运算符。

评：逻辑运算有很多种，至少还有个 XOR,怎么可能只有 3 种逻辑运算符呢？

在 BASIC 中还有 Eqv,Imp

BASIC 也不是直接用 AND，OR，NOT 作为逻辑运算符，而是用的 And,Or,Not，只不过 BASIC 不区分大小写罢了

再则，BASIC 的逻辑运算符的运算含义也绝对不同于 C 的逻辑运算符的运算意义，根本不能当作同一种东西一概而论

在 C 语言中不能在程序中直接用 AND，OR，NOT 作为逻辑运算符，而是用其他符号代替。见表 4 .1。

评：这圈子绕的！从上海到苏州，却非要舍近求远地先到欧洲兜一圈再到苏州

讲了一火车废话错话，有意义的内容用几个字就能概括——C 有三个逻辑运算符：&&,||、!。

表 4.1 C 逻辑运算符及其含义

运算符	含义	举例	说明
&&	逻辑与	a&&b	如果 a 和 b 都为真，则结果为真，否则为假
||	逻辑或	a || b	如果 a 和 b 有一个以上为真，则结果为真，二者都为假时，结果为假
!	逻辑非	！a	如果 a 为假时，则！a 为真，如果 a 为真，否则！a 为假

评：1.C 语言没有“真”“假”这种东西

2.对&& ||运算的解释是错误的，是一种似是而非的歪曲

3.只介绍了优先级，对结合性只字未提。

总之，学习者不可能从中真正理解这 3 个运算

&&和||运算涉及到 sequence point，不懂得这个概念，就不能说是真正懂得了这两种运算

P94

5>3&&8<4-!0

表达式自左至右扫描处理求解。首先处理“5>3”（因为关系运算符优先于逻辑运算符&&）。在关系运算符>两侧的 5 和 3 作为数值参加关系运算，“5>3”的值为 1（代表真），再进行“1&&8<4-!0”的运算，8 的左侧为“&&”，右侧为“<”，根据优先规则，应先进行“<”的运算，即先进行“8<4-!0”的运算。现在 4 的左侧为“<”，右侧为“-”运算符，而“-”优先于“<”,因此应先进行“4-!0”的运算，由于“!”的级别最高，因此先进行“!0”的运算，得到结果 1.然后进行“4-1”的运算，得到结果 3，再进行“8<3”的运算，得 0，最后进行“1&&0”的运算，得 0.

评：自相矛盾之处在于

一会说“自左至右扫描处理求解”

一会又根据优先级确定运算的先后

"8 的左侧为“&&”，右侧为“<”，根据优先规则，应先进行“<”的运算"更荒唐，大概老谭忘记了还有优先级更高的“!”

这两种说法都是站不住脚的，也都是错误的

C 语言编译系统在表示逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”

评：是什么话

编译系统怎么表示运算结果？运算是在编译时进行的吗？如果不是，编译系统怎么可能表示运算结果呢

（1）若 a=4，则!a 的值为 0.……

（2）若 a=4,b=5,则 a&&b 的值为 1.……

（3）a 和 b 值分别为 4 和 5，a||b 的值为 1.

……

评：这就和他的代码中，哪怕是常量也一定要赋值给一个变量再运算，不管有没有必要，是一致的

实际上

（1）无非说的是 !4 的值为 0

（2）无非说的是 4&&5 的值为 1

（3）无非说的是 4||5 的值为 1

搞不清楚他画蛇添足地弄出来 a、b 作甚

P95

在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。举例如下。

评：“执行”“运算符”
听着比日本话还别扭：运算符执行地有
作者是中国人吗

(1)a&&b&&c。只有 a 为真（非 0）时，才需要判别 b 的值。只有当 a 和 b 都为真的情况下才需要判别 c 的值。如果 a 为假，就不必判别 b 和 c（此时整个表达式已确定为假）。如果 a 为真，b 为假，不判别 c，见图 4.7。
评：原来这就是所谓的“并不是所有的逻辑运算符都被执行”啊
这里的例子说明的恰恰是两个“&&”运算都执行了
任何一个没执行都不可能得到最后的结果
没被执行的是求 b 和 c 的值

既然关系表达式和逻辑表达式的值是 0 和 1，而且在判断一个量是否为“真”时，以 0 代表“假”，以非 0 代表“真”。那么就可以理解为什么在 if 语句中表达式可以是任何数值表达式。
评：不存在这种因果关系。把“关系表达式和逻辑表达式的值是 0 和 1”扯进来更是思路不清的表现
试想，关系表达式和逻辑表达式的值是 0 和 1 和 if 语句的表达式有什么必然的联系呢
所以这是一段似是而非的论述

此外“if 语句中表达式可以是任何数值表达式”是错误的
这个表达式只能是标量(scalar)类型

P97

如果读者使用的 C++的编译器（如 Visual C++）,则可以使用逻辑型变量。但要把文件名后缀改为.cpp，作为 C++程序，
评：搞不清楚这是想做什么
C、C++分不清楚在谭书中是一贯的
著名的把赋值表达式当作左值的笑话就是这种情况

4.5 逻辑运算符和逻辑表达式

P97

```
if(a>b)
    max=a;
else
    max=b;
……
```

C 提供条件运算符和条件表达式来处理这类问题。

评：牵强附会
C 语言的条件运算符并非是为这类问题而存在的
这种说法是误导，会限制学习者正确地应用条件运算符

“?"是条件运算符。
评：有人不主张挑这种鸡毛蒜皮的小错误
但这种小错误如此之多又说明什么呢

P98

a>b?(max=a) : (max=b);
a>b?printf("%d",a):printf("%d",b);
评：真应了那句“再好的语言也挡不住有人写出垃圾代码”？

P99

ch=(ch>='A'&&ch<='Z')?(ch+32):ch;
评：风格、效率、可移植性都成问题
效率指的是和什么比的？能具体说一下不？

4.6 选择结构的嵌套

P100

为了避免二义性的混淆，最好使内嵌语句也包含 else 部分，这样 if 的数目和 else 的数目相同，从内层到外层一一对应，不致出错。
评：这招灵吗？我很怀疑
不管有没有必要都把 else 写上？

P100~101

```
        -1  （x<0 ）
y =     0   （x=0 ）
        1   （x>0 ）
```

……
程序 1：

```
#include <stdio.h>
int main()
{
   int x,y;
```

```
    scanf("%d",&x);
    if(x<0)
        y=-1;
    else
        if(x==0)y=0;
        else y=1;
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

……

程序 2：

```
#include <stdio.h>
int main()
{
    intx,y;
    scanf("%d",&x);
    if(x>=0)
        if(x>0)y=1;
        else       y=0;
    else           y=-1;
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

评：这也太雷人了吧

一个如此简单的问题

居然弄了个那么复杂的嵌套结构

舍简就繁

程序 2 居然有运行结果

P101~102

为了使逻辑关系清晰，避免出错，一般把内嵌的 if 语句放在外层的 else 子句中（如程序 1 那样），这样由于有外层的 else 相隔，内嵌的 else 不会被误认为和外层的 if 配对，而只能与内嵌的 if 配对，这样就不会搞混。

评：这是因噎废食而想到的削足适履性质的差劲办法

代码的结构应该服从思想，应该是思想的自然展开和表达

先把思想装进形式上的套子里写不出优质的代码

4.7 用 switch 语句实现多分支选择结构

P103

4.7 用 switch 语句实现多分支选择结构
switch 语句的一般形式如下：

```
switch(表达式)
  {
    case 常量 1：语句 1
    case 常量 2：语句 2
    ……
    case 常量 n：语句 n
    default：语句 n+1
  }
```

……
switch 语句下面的花括号内是一个复合语句。
评：switch 语句的一般形式并非是
switch(表达式) 复合语句

switch 后面括号内的“表达式”，其值的类型应为整数类型（包括字符型）
评：1 整数类型本来就包括字符类型
2 谭书 43 页数据类型总表中根本没有“整数类型”，不清楚谭书中的“整数类型”究竟是指的什么

case 后面跟一个常量(或常量表达式)
评：不对！只能是整数常量表达式

每个 case 标号出现的次序不影响执行结果
评：显然不对

P104

在 case 子句中虽然包含了一个以上执行语句，但可以不必用花括号括起来，会自动顺序执行本 case 标号后面所有的语句。当然加上花括号也可以。
评：到底想表达什么？

例 4.7
评：毫无意义的例题
1.根本就不是完整的代码
2.用到了根本没介绍的知识（函数定义等）
3.风格拙劣，写出的代码中也有很多错误,例如 intx
问题本身就描述的不完整，讲解部分完全是空对空的纸上谈兵

4.8 选择结构程序综合举例

P105

4.8 选择结构程序综合举例

评：居然一口气把判断闰年的问题写了四遍，不可思议

完全看不出那样做有什么意义

没有分析各种写法的利弊得失

几个写法都不咋地

尤其是第四个，简直是为赋新词强说愁——仅仅是为了用一下 C99 的_Bool 类型，其余的都和第一个写法完全一样。写完之后无法编译，又建议读者把.c 改成.cpp，简直是笑话。C++和 C 根本就是两回事，披着羊皮的狼不还是狼吗

P106

```
#include <stdio.h>
int main()
{
   int year,leap;
   printf("enter year:");
   scanf("%d",&year);
   if(year%4==0)
      {
      if(year%100==0)
           {
            if(year%400==0)
               leap=1;
            else
               leap=0;
           }
      else
         leap=1;
      }
   else
      leap=0;
   if(leap)
         printf("%d is ",year);
   else
         printf("%d is not ",year);
   printf("a leap year.\n");
   return 0;
```

}

评：这段代码没有错误，但风格很差，而且啰嗦

1.leap=1;　leap=0;　是很差劲的写法，可读性差且容易出错。应该使用符号常量

2.第 7~20 行，啰嗦累赘。若在此之前给 leap 赋值，代码和逻辑则都要简洁得多。

3.第 21~25 行，更是似是而非。若不希望写重复的代码，至少应该把"%d is "部分写在 if 语句外面，即：

```
printf("%d is ",year);
if(leap==0)
    printf("not ");
printf("a leap year.\n");
```

但这也不是最好的写法，实际上还有更好的写法。

下面是我改写的代码

```
#include <stdio.h>
#define LEAP        1
#define NOT_A_LEAP 0
int main( void )
{
   int year,leap = NOT_A_LEAP ;
   printf("enter year:");
   scanf("%d",&year);
   if( year % 4 == 0 )
       if( year % 100 == 0 ){
           if( year % 400 == 0 )
               leap = LEAP ;
       }
       else {
           leap = LEAP ;
       }

   printf( "%d is %sa leap year.\n" , year , ( leap == LEAP )?"":"not ");

   return 0;
}
```

P108~109

例 4.9 求 ax^2+bx+c=0 方程的解。

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

```
#include <math.h>
int main( )
{
   double a,b,c,disc,x1,x2,realpart,imagpart;
   scanf("%lf,%lf,%lf",&a,&b,&c);
   printf("The equation");
   if(fabs(a)<1e-6)
      printf("is not a quadratic\n");
   else
   {
      disc=b*b-4*a*c;
      if(fabs(disc)<1e-6)
         printf("has two equal roots:%8.4f\n",-b/(2*a));
      else
         if(disc>1e-6)
         {
            x1=(-b+sqrt(disc))/(2*a);
            x2=(-b-sqrt(disc))/(2*a);
            printf("has distinct real roots:%8.4f and %8.4f\n",x1,x2);
         }
         else
         {
            realpart=-b/(2*b);
            imagpart=sqrt(-disc)/(2*a);
            printf("has complex roots:\n");
            printf("%8.4f+%8.4fi\n",realpart,imagpart);
            printf("%8.4f-%8.4fi\n",realpart,imagpart);
         }
   }
   return 0;
}
```

评：书上说输入 1，2，1
输出为 The equationhas two equal roots: -1.0000
这没什么问题

可是输入 1e-8,2e-8,1e-8 时
输出居然是 The equation is not a quadratic 就说不过去了吧？

P109~110

例 4.10 运输公司对用户计算费用。路程（skm）越远。每吨·千米运费越低。标准如下

	s<250	没有折扣
250<	s<500	2%折扣

500< s<1000 5%折扣
1000< s<2000 8%折扣
2000< s<3000 10%折扣
3000≤ s 15%折扣

……

（代码略）

评：此题目根本就不应该用 swtitch 语句解决

P111

float p,w,d,f;

（3）变量名尽量采用“见名知意”的原则，……，在本书的例题程序，由于是练习程序，并且考虑到多数读者的习惯和方便，尽量不采用较长的变量名，而用单词的首字母或缩写作为变量名。在读者今后编程时，可根据实际情况决定。

评：实际是在用垃圾风格熏陶读者，使读者耳濡目染地养成坏习惯

口口声声“考虑到多数读者的习惯和方便”是文过饰非,为自己的垃圾风格辩解

良好的风格习惯不是一下子就能养成的

相反，养成了坏习惯就很难改掉

（4）第 6 行“printf("please enter price,weight,discount:");”的作用是向用户提示应输入什么数据，以方便用户使用，避免出错，形成友好界面。建议读者在编程序（尤其是供别人使用的应用程序）也这样做，在 scanf 函数语句输入数据前，用 printf 函数语句输出必要的“提示信息”。

评：1.身教胜于言教，除了这个例题，老谭自己做到没有？赤裸裸的 scanf 到处都是，教别人去做，有点不给力吧？

2.代码中写的是

printf("please enter price,weight,discount:"); //提示输入的数据

scanf("%f,%f,%d",&p,&w,&s) ; //输入单价、重量、距离

英语不好，硬是没看懂

3.“ scanf 函数语句”，“printf 函数语句”，估计是 C99 的新概念

第 5 章 循环结构程序设计

P116

循环体如果包含一个以上的语句，应该用花括号括起来，作为复合语句出现。如果不加花括号，则 while 语句的范围只到 while 后面第一个分号处。

评：while(EXP1)

if(EXP2)

S1 ;
else
S2 ;
怎么说？

sum=sum+i;
评：太像 BASIC 了
应该写成 sum += i ;

5.3 用 do…while 语句实现循环

P117

do
语句
while(表达式);
其中的“语句”就是循环体。它的执行过程可以用图 5.4 表示。
评：图 5.4 的流程图居然没有入口
太有“创意”了

例 5.2 用 do…while 语句求 1+2+3+……100，即 $\sum_{n=1}^{100} n$
评：这个题目用 do…while 语句来解决属于头脑不清醒，思维呈发散性混乱状态

我也同意应该用 for 语句。
编程应该用恰当的语句描述算法，而不是扭曲思想去适应语句。编程不是数学上的一题多解，不是花样越多越好，这方面软件史上有过深刻的教训

P118

#include <stdio.h>int main()

对同一个问题可以用 while 语句处理，也可以用 do...while 语句处理。do...while 语句结构可以转换成 while 结构。
评：严重的误导
结构换来换去只说明思路上模糊不清

在一般情况下，用 while 语句和用 do…while 语句处理同一问题时，若二者的循环体部分是一样的，那么结果也一样。

评：这是瞪着眼睛说胡话
完全不具备起码的逻辑思维能力

P118~119

例 5.3 while 和 do……while 循环比较
评：无厘头式的比较
结论毫无意义

5.4 用 for 语句实现循环

P120

而且 for 语句更为灵活，不仅可以用于循环次数已经确定的情况，还可以用于循环次数不确定而只给出循环结束条件的情况。它完全可以代替 while 语句。
评：令人啼笑皆非
既生瑜何生亮
老谭的意思是 K&R 发明 C 时既然设置了 for 语句再设置 while 语句实在是多余

P121

```
for(i=1;i<=100;i++)
                    sum=sum+i;
```

(2)"表达式 1"可以省略……

```
    i=1;
      for(;i<=100;i++)sum=sum+i;
```

评：因为不懂
所以把恶劣的编程风格作为知识讲授

for 语句的一般形式
for(表达式 1;表达式 2;表达式 3)语句
可以改写为 while 循环的形式：
表达式 1;
while 表达式 2
{
语句
表达式 3;
}
二者无条件等价。
评：1.while 语句居然没有()

2.老谭大概忘记 C 语言还有个 continue 语句吧

P122

(3)"表达式 2"也可以省略……

for(i=1;;i++)sum=sum+i;

评：阉割 for 语句的“大师”

整个 121，122，123 页都是在做这种无聊的阉割

P123

可见 for 语句比 while 语句功能强，

评：胡扯

这表明老谭根本不懂得各种循环语句究竟应该怎么应用

while(1)rintf("%d\n",i);

for(i=0,j=100;i<=j;i++,j++)k=i+j;

表达式 1 和表达式 3 都是逗号表达式，各包含两个赋值表达式，

（9）表达式 2 一般是关系表达式（如 i<=100）或逻辑表达式（如 a<b&&x<y），但也可以是数值表达式或字符表达式，

评：典型的谭体句型

仿谭体造句：一般是土豆或洋葱，但也可以是植物或辣椒

P124

```
for(;(c=getchar())!='\n' ; )
    printf("%c",c);
```

for 语句中只有表达式 2，而无表达式 1 和表达式 3.其作用是每读入一个字符后立即输出该字符，直到输入一个“换行”字符。

评：1.实现那个 for 语句的功能，应该

```
while( ( c = getchar() ) != '\n' )

            putchar(c);
```

2.功能描述错误

所描述的功能实际是

```
do
    putchar( c=getchar() );
while( c != '\n' );
```

C 语言的 for 语句比其他语言（如 FORTRAN,Pascal）中的 for 语句功能强得多。

评：无病呻吟

比较的标准是什么？

搞不清想要传递给读者什么信息

C 的 for 语句比 C++的 for 语句功能也强得多？

5.5 循环的嵌套

P124~125

5.5 循环的嵌套

评：毫无意义地罗列了 6 种循环嵌套的结构

还写错了一种

```
do
  {...
      do
        {...}
      while()
  }while()
```

这个反映了老谭基本功

5.6 几种循环的比较

P125

3 种循环都可以用来处理同一问题，一般情况下它们可以互相替代。

评：严重的误导

for 循环可以在表达式 3 中包含使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式 3 中。因此 for 语句的功能更强，凡用 while 循环能完成的，用 for 循环都能完成。

评：胡扯+误导

5.7 改变循环执行的状态

P126

例 5.4 在全系 1000 学生中，征集慈善募捐，当总数达到 10 万元时就结束，统计此时捐款的人数，以及平均每人捐款的数目。

评：嗯！

够黑！够狠！

和咱们的 XX 有得一拼

从题目中根本看不出来若是总数达不到怎么办

看来是志在必得

收钱的事，完不成怎么可以？！

num=7

aver= 14669.71

评：好在学生中有几个大款

问题是

哪怕小学生也不会算出平均每人捐款的数目是 14669.71 吧？

P127

break 语句的一般形式为

break

评：还真是从没见过这种语句

P128~129

例 5.6 输出以下 4*5 的矩阵。

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20

……

编写程序：

```
#include <stdio.h>
int main( void )
{
    int i,j,n=0;
```

```
    for(i=1;i<=4;i++)
       for(j=1;j<=5;j++,n++)
       { if(n%5==0)printf("\n");
          printf("%d\t",i*j);
       }
    printf("\n");
    return 0;
}
```

评：这个代码写得非常愚蠢，功能上也有缺陷。
在教科书中传授这种代码是一种犯罪行为
这种愚蠢的代码能把初学者变成脑残和弱智

在 129 页愚蠢代码的基础上又毫无意义地分别加上了 continue 和 break
使得愚蠢毫无意义地复杂化，然后让读者分析
复杂的愚蠢代码是最难分析的
而且有害无益

5.8 循环程序举例

P131

例 5.7 用π /4≈1-1/3+1/5-1/7…公式求的近似值，直到发现某一项的绝对值小于 10^6 为止（该项不累加）。
评：任何一项都小于 10^6

P132

```
#include <stdio.h>
#include <math.h>
int main()
{
int sign=1;
double pi=0.0,n=1.0,term=1.0;
while(fabs(term)>=1e-6)
{
   pi=pi+term;
   n=n+2;
   sign=-sign;
   term=sign/n;
}
pi=pi*4;
printf("pi=%10.8f\n",pi);
```

return 0;
}

评：那个 fabs()函数调用毫无必要
n 为 double 类型 sign 为 int 类型也莫名其妙
此外 pi=pi+term 这样的写法纯粹是 BASIC 语言的写法
这个代码至少应该改成这样

```
#include <stdio.h>
int main( void )
{
double pi=0.,n=1.,term=1., sign=1.;
while( term >= 1e-6 )
{
   pi += sign * term;
   n += 2. ;
   sign = - sign ;
   term = 1. / n ;
}
pi *= 4. ;
printf("pi=%10.8f\n",pi);
return 0;
}
```

在 C 库函数中，有两个求绝对值的函数
评：错
C99 中不是这样
这再次表明老谭这本书“按 C 语言的新标准 C99 进行介绍”是骗人的

P133

经过对程序的补充和运行，可以知道在 while(fabs(t)>=1e- 6)时，执行 50 万次，当 while(fabs(t)>=1e- 8)时，执行循环体 5000 万次、两者相差 100 倍，在分别运行以上两种情况下得程序时，可以明显地感觉到后者运行的时间长很多。
评：一堆废话

对于
π /4≈1-1/3+1/5-1/7…
最后一项<1e-6 要执行多少次，小学生心算都能算出来，可老谭居然要“经过对程序的补充和运行”才知道
fabs(t)：哪冒出来的“t”？
运行时间的估计居然靠实际“感觉”，不知道老谭怎么感觉出相差 100 倍的

P134

例 5.8 求 Fibonacci 数列的前 40 个数。

```
#include <stdio.h>
int main()
   {
    int f1=1,f2=1,f3;
    int i;
    printf("%12d\n%12d\n",f1,f2);
    for(i=1;i<=38;i++)
    {
       f3=f1+ f2;
       printf("%12d\n",f3);
       f1=f2;
       f2=f3;
    }
   return 0;
}
```

评：for 语句中的 “38”非常蹩脚
属于莫名其妙的常数

P135

```
#include <stdio.h>
int main()
   {
    int f1=1,f2=1;
    int i;

    for(i=1;i<=20;i++)
    {
       printf("%12d%12d",f1,f2);
       if(i%2==0)printf("\n");
       f1=f1+f2;
       f2=f2+f1;
    }
   return 0;
}
```

评：这个代码中的那个 20 很烂
if(i%2==0)printf("\n"); 十分蹩脚，风格也极其差劲

例 5.9 输入一个大于 3 的整数 n,判定它是否为素数（prime,又称质数）

评：居然有这样的程序，只能判断大于 3 的整数是不是素数
　　这东西交给客户时怎么说呢？
　　我这个程序能判断 3 以上的素数
　　3 以下的还得您自己亲自判断

```
#include <stdio.h>
int main()
{int n,i;
   printf("please enter a integer number,n=?");
   scanf("%d",&n);
   for(i=2;i<=n-1;i++)
      if(n%i==0)break;
      if(i<n)printf("%d is not a prime number.\n",n);
      else printf("%d is a prime number.\n",n);
      return 0;
}
```

评：这风格！山奔海立的，自成一派

```
        if(i<n)printf("%d is not a prime number.\n",n);
         else printf("%d is a prime number.\n",n);
```

　　写的无比笨拙

P137

图 5.19

k< √n

评：应为 k= √n

```
#include <stdio.h>
#include <math.h>
int main()
{int n,i,k;
   printf("please enter a integer number,n=?");
   scanf("%d",&n);
   k=sqrt(n);
   for(i=2;i<=k;i++)
      if(n%i==0)break;
   if(i<=k)printf("%d is not a prime number.\n",n);
   else printf("%d is a prime number.\n",n);
      return 0;
}
```

评：k=sqrt(n);试图求出 n 的平方根的整数部分
但这点是得不到保证的
另一个问题是
题目要求“输入一个大于 3 的整数 n”
但无论输入的整数是否大于 3
程序都会给出结果
输入一个负数的话还会亲眼目睹程序崩溃的壮烈景观

P137~138

例 5.10 求 100~200 间的全部素数。

```
#include <stdio.h>
#include <math.h>
int main()
   {int n,k,i,m=0;
      for(n=101;n<=200;n=n+2)
         {k=sqrt(n);
          for(i=2;i<=k;i++)
             if(n%i==0)break;
          if(i>=k+1)
             {printf("%d",n);
              m=m+1;
             }
          if(m%10==0)printf("\n");
         }
      printf("\n");
      return 0;
}
```

评：1.逻辑不清，到底怎样算“100~200 间”，“n=101;n<=200”无论怎么解释都是错的
2.k=sqrt(n);，见 1363 楼
3.if(i>=k+1)，很不清醒的画蛇添足，实际应该是 if(i>k)
4.if(m%10==0)printf("\n"); 不仅笨拙，而且如果 n 的初值不是素数的话会输出若干丑陋的空行
5.风格很烂

```
if(m%10==0)printf("\n");
   if(n%i==0)break;
   {int n,k,i,m=0;
   {k=sqrt(n);
```

都属于很烂的风格

```
n=n+2
m=m+1
```

这种写法简直就不是 C 语言，而是 BASIC 等语言的写法

P138

(1) 根据常识，偶数不是素数，所以不必对偶数进行判断，只对奇数进行检查。故循环变量 n 从 101 开始，每次增值 2。

评：这里至少有三个问题

1.200 也是偶数，为什么要 n<=200,和这段解释自相矛盾。

2.这种代码根本没有通用性，如果是求另一个范围内的素数，代码需要改动的地方很多。

3.“根据常识，偶数不是素数”，这是常识性的错误。小学生都知道错在哪里

(2)从附录 F 可以看到：sqrt 是求平方根的函数，它要求参数为双精度数。在执行时会自动将整数 n 转换为双精度数。求出的函数值也是双精度数，再把它赋值给整型变量 k，系统会自动小数部分舍弃，只把整数部分赋给 k。在进行编译时，系统给出警告，提醒用户有可能出现误差。只要用户确认没有问题，可以不理会它。

评：1.用整数类型做对应原型中类型为 double 的实参，不规矩（当然不是不可以）

2.“系统会自动小数部分舍弃”，语文问题

3.用户不可能确认没有问题

4.教人不理会警告，是一种教唆行为。优秀的程序员不会容许任何警告，哪怕是无害的情况。更何况这里的警告是一个让人放心不下的警告。

P138~139

例 5.11 译密码……

将字母 A 变成字母 E,a 变成 e,即变成其后的第 4 个字母，W 变成 A，X 变成 B,Y 变成 C，Z 变成 D，……#include <stdio.h>

```
int main()
  {char c;
    c=getchar();
    while(c!='\n')
      {if((c>='a'&&c<='z')||(c>='A'&&c<='Z'))
        {if( c>='W'&&c<='Z'||c>='w'&&c<='z')c=c-22;
                                    //如果是 26 个字母中最后 4 个字母之一就使 c-22
          else c=c+4;
      }
       printf("%c",c);
       c=getchar();
      }
   printf("\n");
   return 0;
  }
```

评：风格太烂

P139~140

程序改进：

```
#include <stdio.h>
int main()
   {char c;
      while((c=getchar())!='\n')
         {if((c>='A'&&c<='Z')||(c>='a'&&c<='z'))
            { c=c+4;
               if( c>='Z'&&c<='Z'+4|| c>'z')
                  c=c-26;
            }
          printf("%c",c);
         }
     printf("\n");
     return 0;
   }
```

评：确实有点改进

可惜改错了

第 6 章 利用数组处理批量数据

6.1 怎样定义和引用一维数组

P142

由于计算机键盘只能输入有限的单个字符而无法表示上下标，C 语言就规定用方括号中的数字来表示下标，

评：胡扯

P143

如果在被调用的函数（不包括主函数）中定义数组，其长度可以是变量或非常量表达式。如:

```
void fun(int n)
{
int a[2*n];     //合法，n 的值从实参传来
……
}
```

评：胡扯

如“int a[n];”是不合法的。也就是说，C 语言不允许对数组的大小作动态定义，即数组的大小不依赖于程序运行过程中变量的值。

评：可笑的是这段文字与 1403 楼引用的文字出现在同一页上
　　还没翻页就开始自己打自己耳光

int a[10];
它表示定义了一个整型数组，数组名为 a，此数组有 10 个整型元素。

评：根据该书 43 页对数据类型的划分
　　整型数据至少包括 6 种以上的数据类型
　　所以，所谓的“数组有 10 个整型元素”中的整型元素属于指代不明

定义一维数组的一般形式为
类型符 数组名[常量表达式];

评：无论从 C90 还是 C99 标准来说都是错误的

　　从 C90 来说，“常量表达式”是不准确的，不是任何常量表达式都可以
　　从 C99 来说，并没有要求一定是“常量表达式”

常量表达式中可以包括常量和符号常量

评：标准的谭氏废话句型
　　这话放小学生作文里是要得红叉叉的

不能包括变量，如“int a[n];”是不合法的。也就是说，C 语言不允许对数组的大小作动态定义，即数组的大小不依赖于程序运行过程中变量的值。例如下面这样定义数组是不行的：

```
int n;
scanf("%d",&n) ;    //企图在程序中临时输入数组的大小
int a[n];
```

评：已经说了是“常量表达式”，“不能包括变量”是废话
　　此外，老谭号称他这本书依据 C99 写的，如“int a[n];”是不合法的 。这句是完全错误的。

　　这段代码在支持 C99 的编译器下没有问题。
　　自相矛盾的是，在同一页，老谭又写到

如果在被调用的函数（不包括主函数）中定义数组，其长度可以是变量或非常量表达式。如:

```
void fun(int n)
{
int a[2*n];    //合法，n 的值从实参传来
……
}
```

评：刚说完[]内只能是“常量表达式”，一转眼又说 int a[2*n];　　//合法
　　这种自相矛盾带给初学者的不止是错误还有混乱和无所适从
　　如果按 C90 这段论说是绝对错误的
　　如果按 C99 来说这段说法依然是错误的，因为 main()与其他函数相比除了是程序开始执行的位置之外

没有什么特殊的，“不包括主函数”的说法完全是不懂装懂的信口开河，没有任何根据

在调用 func 函数时，形参 n 从实参得到值。这种情况称为“可变长数组”，允许在每次调用 func 函数时，n 有不同的值。
评：又在胡扯，
“可变长数组”根本不是像他说的那样

但是在执行函数时，n 的值是不变的
评：n 是变量，怎么就不变了呢

P144

6.1.2 怎样引用一维数组

6.1.2 怎样引用一维数组
在定义数组并对其中的各元素赋值后，就可以引用数组中的元素。
评：赋值后就可以引用
赋值时呢？
难道不是引用？

例如下面的赋值表达式包含了对数组元素的引用：
：a[0]=a[5]+a[7]-a[2*3]
每一个数组元素都代表一个数值。
评：”：” 印刷错误

每一个数组元素都代表一个数值：错！这个表达式中的数组元素并非都代表数值

例 6.1
```
int i,a[10];
for(i=0;i<=9;i++)
    a [ i ] = i ;
```
评：i<=9 是业余写法

P147

例 6.3 有 10 个地区的面积，要求对它们按由小到大的顺序排列。
评：这算题吗？
条件本身就不全，怎么可能完成呢？

在题目条件不充分的条件下给出代码
是在鼓励程序员胡乱猜测程序功能并示范如何进行胡猜
而程序员擅自胡乱猜测程序功能是与软件工程的思想和基本原则背道而驰格格不入的
所以就不能理解老谭这书究竟是在教别人写程序
还是在诱导教唆学习者违背并践踏写程序的基本原则

6.2 怎样定义和引用二维数组

P149

二维数组常称为矩阵(matrix)。
评：概念混乱的惊人
一个是程序设计语言中的概念
另一个是数学中概念

二维数组定义的一般形式为
类型说明符 数组名[常量表达式][常量表达式];
评：两个“常量表达式”给人貌似相同的错觉
另外“常量表达式”这个描述也不准确，不是所有的常量表达式都可以

P150

二维数组元素的表示形式为
数组名[下标][下标]
评：两个“下标”没有区别，不妥

数组元素可以出现在表达式中，也可以被赋值
评：谭氏废话
相当于什么也没说
其实只要知道数组元素本身也是表达式
那些都是不言而喻的

P153

例 6.4

```
#include <stdio.h>
int main()
{
int a[2][3]={{1,2,3},{4,5,6}};
……
for(i=0;i<=1;i++)
```

```
{
   for(j=0;j<=2;j++)
      {
       printf("%5d",a [ i ][j]);
      ……
      }
}
……
return 0;
}
```

评：这是一种似是而非的外行写法

会给初学者带来很坏的影响

至少应该写成

```
for(i=0;i<2;i++)
{
   for(j=0;j<3;j++)
      {
       printf("%5d",a[ i ][j]);
      ……
      }
}
```

更好的写法是用 sizeof 运算求出数组的尺寸

```
printf("max=%d\nrow=%d\ncolum=%d\n",max,row,column);
```

评：密密麻麻

风格太差

6.3 字符数组

P154

前已介绍：字符型数据是以字符的 ASCII 代码存储在存储单元中的，一般占一个字节。由于 ASCII 代码也属于整数形式，因此在 C99 标准中，把字符类型归纳为整型类型中的一种。评：短短一段话有多处错误

1. “字符型数据是以字符的 ASCII 代码存储在存储单元中的”，未必
2. “一般占一个字节”，没什么一般特殊而言，char 类型就是占一个字节
3. “由于 ASCII 代码也属于整数形式，因此在 C99 标准中，把字符类型归纳为整型类型中的一种。”，压根就没有这种因为所以。

   此外 C89 中字符数据就属于整数类型
4. “整型”，这个概念和书里出现的其他“整型”概念不一致。

字符串是存放在字符型数组中的。
评："ABC"
这样的字符串存放在哪个数组中？

P154~155

用来存放字符数据的数组是字符数组。
……
也可以用整型数组它存放字符数据，例如
int c[10]; //合法，但浪费存储空间
c[0]='a';
评：1.自相矛盾。按照第一句的说法, c 也是字符数组
2.第二句话是病句
3.int c[10];未必属于浪费存储空间

P156

例 6.6 输出一个已知的字符串。
例 6.7 输出一个菱形图。
评：谭书的很多例题都是这样没头没脑
如果不看输出结果
你根本不知道他这题目究竟想做什么

例 6.6 输出一个已知的字符串。
解题思路：先定义一个字符数组，并用“初始化列表”对其赋以初值。然后逐个输出此字符数组中的字符。
评：“解题思路”倒是更像一个具有具体要求的题目
而“输出一个已知的字符串”根本不成其为题目
矛盾的是“解题思路”和输出字符串根本扯不上边
如果输出一个字符串
puts()和 printf()都很容易完成
根本用不着舍近求远地去定义字符数组
况且解题思路中没有丝毫字符串的影子
如果看下代码
就会发现代码和题目之间驴唇不对马嘴
编写程序：

```
#include <stdio.h>
int main()
{ char c[15]={'I',' ','a','m',' ','a',' ','s','t','u','d','e','n','t','.'};
    int i;
    for(i=0;i<15;i++)
        printf("%c",c[ i ]);
```

```
    printf("\n");
    return 0;
}
```

例 6.7 输出一个菱形图。

解题思路：先画出一个如图 6.12 所示的平面菱形图案。

评：题目本身没头没脑

没有人能解这样的题目

图 6.12 所画的也不是菱形图案

编写程序：

……

```
char diamond[][5]={{' ',' ','*'},{' ','*',' ','*'},{'*',' ',' ',' ','*'},
                                                {' ','*',' ','*'},{' ',' ','*'}}
```

……

评：初始化的风格很差。

P156~157

例 6.6 就是用一个一维的字符数组来存放字符串"I am a student."的

评：然而 例 6.6 写的却是

```
char c[15]={'I',' ','a','m',' ','a',' ','s','t','u','d','e','n','t','.'};
```

P157

为了测定字符串的实际长度，C 语言规定了一个“字符串结束标志”。

评：C 语言确实规定了一个“字符串结束标志”，但并不是“为了测定字符串的实际长度”。所谓“为了测定字符串的实际长度”，属于老谭自己的臆测

ASCII 码为 0 的字符不是一个可以显示的字符，而是一个“空操作符”

评：“字符”怎么成了“操作符”？

概念严重不清

printf("How do you do?\n");

在执行此语句时系统怎么知道应该输出到哪里为止呢？实际上，在向内存中存储时，系统自动在最后一个字符'\n'的后面加了一个'\0',……

评：又见“系统”

“系统”是老谭最得心应手的挡箭牌

说不清楚的事情统统都推给含混不清的“系统”

相信老谭自己也解释不清这两个系统究竟是什么，是不是同一个意思

char c[]="I am happy";

这里不像例 6.6 那样用单个字符作为字符数组的初值，而是用一个字符串(注意字符串的两端是用双撇号而不是单撇号括起来的)作为初值。

评：1.单个字符不可能作为字符数组的初值

2.“用单个字符作为字符数组的初值”与同页上“例 6.6 就是用一个一维的字符数组来存放字符串"I am a student."的”自相矛盾

3.用字符串作为数组的初值同样说不通

p160

char str1[5],str2[5],str3[5];
scanf("%s%s%s",str1,str2,str3);
输入数据：
How are you?
由于有空格字符分隔，作为 3 个字符串输入。在输入完后，str1,str2,str3 数组的状态如下：
H o w \0 \0
a r e \0 \0
y o u ? \0
数组中未被赋值的元素的值自动置'\0'。
评：荒唐透顶
这又是老谭自己的臆测

char str[13];
scanf("%s",str);
如果输入以下 12 个字符：
How are you?
由于系统把空格作为输入的字符串之间的分隔符，因此只将空格前得字符“How”送到 str 中。由于把“How”作为一个字符串处理，故在其后加'\0'。str 数组的状态为
H o w \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
评：不能因为新手无知就这样糊弄他们吧

可以用下面的输出语句得到数组的起始地址。
printf("%o",c);
评：老谭在前面反复强调“C 语言本身不提供输入输出语句”（例如第 12 页）
然后又反复地自己打自己耳光（例如第 59 页：最常用的语句是：赋值语句和输入输出语句）
这里居然又出现了“输出语句”
如此地颠三倒四
令人简直无语

其次，用%o 格式输出 c 是错误的

2、即使是在 C 语言中，事实上也容许非 C 风格字符串（的存储实现），虽然并非 C 语言标准定义的——例如 BSTR。
评：如果是这样，前提是必须给出“字符串”的一般性定义
但老谭的书中并没有给出“字符串”的一般定义
倒是说过“字符串（以'\0'结束的字符序列）”这样的话（161 页）
所以可见谭书所说的字符串并非一般意义上的、非 C 标准定义的字符串

p161

1.puts 函数——输出字符串的函数
其一般形式为
puts(字符数组)
评：把 puts 的参数说成“字符数组”是不对的
（考虑到这里还没有学习指针，把 gets 的参数说成是字符数组还马马虎虎）

由于可以用 printf 函数输出字符串，因此 puts 函数用得不多。
评：这个属于井蛙之见
也表明谭不懂得在何种场合如何恰当地使用 puts()函数

用 puts 函数输出的字符串中可以包含转义字符。例如：

```
char str[]={"China\nBeijing"};
puts(str);
```

评：废话
字符串中的字符可以用转义字符的形式写出
跟 puts()函数有什么关系

那个例子中的 str 是多余的
其实可以直接写

```
puts("China\nBeijing");
```

看来老谭真的以为 puts()函数的参数只能是“字符数组”

p162

3.strcat 函数——字符串连接函数
其一般形式为
strcat(字符数组 1,字符数组 2)
评：实际上这两个参数都必须是字符串
　　第二个参数不必是保存在字符数组中的字符串

7.strlwr 函数——转换为小写的函数
其一般形式为
strlwr(字符串)
strlwr 是 STRing LoWeRcase(字符串小写)的缩写。函数的作用是将字符串中的大写字母换成小写字母。
评：不知道老谭哪儿搞来的这东西
　　这个函数根本就不是标准函数

　　从功能上来说
　　这个函数的参数也不可能是“字符串”，只能是存放在数组中的字符串

p163

5)可以用 strncpy 函数将字符串 2 中前面 n 个字符复制到字符数组 1 中去。例如：
strncpy(str1,str2,2);
作用是将 str2 中最前面两个字符复制到 str1 中，取代 str1 中原有的最前面 2 个字符。但复制的字符个数 n 不应多于 str1 中原有的字符(不包括'\0')。
评：对照一下 C 标准的说法：
　　The strncpy function copies not more than n characters (characters that follow a null character are not copied) from the array pointed to by s2 to the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. If the array pointed to by s2 is a string that is shorter than n characters, null characters are appended to the copy in the array pointed to by s1, until n characters in all have been written.
　　不难发现谭的说法漏洞百出

p164

7.strlwr 函数——转换为小写的函数
其一般形式为
strlwr(字符串)
strlwr 是 STRing LoWeRcase(字符串小写)的缩写。函数的作用是将字符串中的大写字母换成小写字母。
评：不知道老谭哪儿搞来的这东西
这个函数根本就不是标准函数

从功能上来说
这个函数的参数也不可能是“字符串”，只能是存放在数组中的字符串

p165

库函数并非 C 语言本身的组成部分，而是 C 语言编译系统为方便用户使用而提供的公共函数。
评：看来 C 标准完全可以把库函数部分删除了
这会节约很多纸张

p165~166

例 6.8 输入一行字符，统计其中有多少个单词，单词之间用空格分隔开。

```
#include <stdio.h>
int main()
   {
     char string[81];
     int i,num=0,word=0;
     char c;
     gets(string);
     for(i=0;(c=string[ i ])!='\0';i++)
          if(c==' ')word=0;
          else if(word==0)
                    {word=1;
                       num++;
                    }
     printf("There are %d words in this line.\n",num);
     return 0;
}
```

评：这是老谭书中最好的代码
模仿的是 K&R 的代码
不过学的不怎么样
毛病很多
首先，c 变量是多余的
81 这个常数也很怪异
第三，老谭在 165 页说过使用 gets 应当在程序开头用 #include <string.h>
但在这里却带头违背
第四 与该代码对应的“N-S 流程图”（图 6.18）不伦不类且有非常明显的几处错误

欣赏一下 K&R 的代码
不难发现老谭 1538 楼的代码如何点金成铁

```
#include <stdio.h>
#define IN   1
#define OUT 0
main()
{
   int c,nl,nw,nc,state;
   state = OUT ;
   nl = nw = nc = 0 ;
   while((c = getchar())!= EOF){
         ++nc;
         if( c == '\n')
              ++nl;
```

```
        if( c ==' ' || c =='\n' || c =='\t')
            state = OUT ;
        else if(state == OUT){
            state = IN ;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl , nw , nc );
}
```

K&R 用的标识符是 state,老谭用的是 word
K&R 用了两个漂亮的#define , 老谭用的是丑陋的 0,1
风格方面
K&R ：

```
        else if(state == OUT){
            state = IN ;
            ++nw;
        }
```

老谭：

```
        else if(word==0)
                {word=1;
                   num++;
                }
```

p167

C 语言把赋值运算作为表达式
评：晕

例 6.9
图 6.19
评：这个图是错误的
　　而且和题目不相符

解题思路：……然后经过 3 次两两比较……
评：代码中并没有经过 3 次两两比较

p168

printf("\nthe largest string is:\n%s\n",string);

评：代码丑了点

太爱用 printf 了

其实可以写为

puts("\nthe largest string is:");

puts(string);

运行结果：

Holland

China

America

the largest string is:"

Holland

评：然而说明该例题的图 6.19 中却是

China

Japan

India

没想到结果居然冲出了亚洲走向了世界

第 7 章 用函数实现模块化程序设计

7.1 为什么要用函数

p170

7.1 为什么要用函数
评：讲老实话，我不清楚什么叫“模块化程序设计”
老谭也没讲清楚究竟什么叫“模块化程序设计”

看来以其昏昏使人昭昭是不行的

注意：函数就是功能。
评：这是个很滑稽的解释
把英汉词典上的两个义项硬说成是一个
如果这个逻辑成立
那么也完全可以说
（教科书的）谬误(error)就是罪孽(error)或犯罪(error)

图 7.1 是一个程序中函数调用的示意图。
评：只是一个三叉树而已
无法示意函数调用
函数调用可能比这复杂得多

p171

```
#include <stdio.h>
int main()
{ void print_star();
   void print_message();
   print_star();
   print_message();
   print_star();
}
void print_star()
{
     printf("********************\n");
}

void print_message()
{printf("How do you do!\n");
}
```

运行结果：

```
********************
      How do you do!
********************
```

评：把函数原型写在 main()之内
　　是误人子弟的写法

　　此外函数原型写的要么可以说是错误的，要么就是毫无意义

　　运行结果是伪造的

在定义这两个函数时指定函数的类型为 void，意为函数无类型。

评：函数无类型的说法极其荒谬
　　C 语言中不存在无类型的函数

　　指定函数的类型为 void
　　应为指定函数返回值的类型为 void

一个源程序文件可以为多个 C 程序共用

评：令人啼笑皆非的说法

p172

C 程序的执行是从 main 函数开始的，如果在 main 函数中调用其他函数，在调用后流程返回到 main 函数，在 main 函数中结束整个程序的运行。

评：这段文字有些莫名其妙,即使从汉语语法的角度来说也有些成问题。成问题的地方在于“在调用后流程返回到 main 函数”不清楚是前面“如果”的一个结果还是对前面“如果”的一个补充。

抛开这种语文方面的缺点不谈，这段文字给人的印象就是“C 程序从 main 函数开始执行，在 main 函数结束”。然而事实真的如此吗？

C 程序的运行离不开一定的环境（Environment），这种环境叫做执行环境（Execution environment）。

运行环境有两种：独立环境（Freestanding environment）和宿主环境(Hosted environment)。

所谓独立环境是指程序并非借助操作系统来运行的，宿主环境则是指程序是在操作系统的控制下执行的。

在这两种环境下，程序开始运行（Program startup）的标志是程序的某个指定的函数开始被调用。

在独立环境中，首先被调用的函数的名字并不一定是 main，而是由编译器自行确定的，这叫做由实现定义（Implementation-defined）。甚至这个首先被调用的函数的类型也同样是由实现定义的。

只有在宿主环境下，C 程序的运行才是从 main()开始的。

因此，“C 程序的执行是从 main 函数开始的”这句话仅仅是在一定的条件下才成立，是片面的。

至于程序在哪个函数结束，C 语言从来没有任何规定。程序可能在 main()中结束，也可能在其他函数中结束。C 语言标准库中有很多函数都与结束程序有关，这些函数的函数原型在 stdlib.h 中描述，例如

```
void abort ( void ) ;
void exit ( int ) ;
```

下面的代码就是一个并非在 main()中结束程序的简单示例：

```
#include “stdlib.h”
void fun ( void ) ;
int main ( void ){
fun();
return 0;
}
void fun ( void ){
exit(1);
}
```

它是在 fun()函数中结束的。

p172

无参函数可以带回也可以不带回函数值，但一般以不带回函数值得居多。
评：毫无意义的废话

所有的函数都有函数值，只是有的函数的返回值是 void 类型

②有参函数。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。
评：思维混乱
居然从有参函数扯到了函数返回值
有参和返回值没关系

例 1.3 的 max 函数定义为 int 型。
评：函数 为 int 型
完全不理解数据类型的含义

7.2 怎样定义函数

p172

7.2.1 为什么要定义函数

C 语言要求，在程序中用到的所有函数，必须“先定义，后使用”。例如想用 max 函数求两个数中的大者，必须事先按规范对它进行定义，指定它的名字、函数返回值类型、函数实现的功能以及参数的个数与类型，将这些信息通知编译系统。这样，在程序执行 max 时，编译系统就会按照定义时所指定的功能执行。如果事先不定义，编译系统怎么能知道 max 是什么、要实现什么功能呢！

评：什么叫“先”、什么叫“后”？
如果是指编辑时的时间顺序，毫无疑问这种说法是错误的。因为先写调用，最后完成再函数定义的情况比比皆是
如果是指源代码的空间次序，这种说法也是错误的。因为在多个源文件情况下，根本就谈不上先后
所以，所谓“先定义，后使用”是一句似是而非的废话

“想用 max 函数求两个数中的大者”，这也是模棱两可的话，什么叫“两个数中的大者”，“两个数”也是含糊不清的概念

“指定它的名字、函数返回值类型、函数实现的功能以及参数的个数与类型，将这些信息通知编译系统”：搞不清这是在说函数定义还是在说函数声明

“在程序执行 max 时，编译系统就会按照定义时所指定的功能执行”，“编译系统”居然也能“执行”程序了，不管你们信不信，反正我就这么写了

“如果事先不定义，编译系统怎么能知道 max 是什么、要实现什么功能呢！”：编译系统可以通过函数原型知道 max 是什么，编译系统并不需要 max 要实现什么功能

p173

对于 C 编译系统提供的库函数，是由于编译系统事先定义好的，库文件中包括了对各种函数的定义。
评：编译系统居然还能“定义”函数
　　库文件中居然包括函数定义

　　究竟什么叫“定义函数”？
　　在 p172 页谭是这样写的

定义函数应包括以下几个内容：
（1）指定函数的名字，……
……
（4）指定函数应当完成什么样的操作
评：编译系统如何指定函数的名字的呢？
　　……
　　又是如何指定函数完成的操作的呢？

　　哪个库文件中写着函数的定义？

1.定义无参函数
……定义无参函数的一般形式为
类型名 函数名()
{
函数体
}
评：1.这是早就过时的写法，应该避免使用
　　2.函数体包括{}部分

```
int max(int x,int y)
{int z;
   z=x>y?x:y;
   return(z);
}
```
评：不多定义一个多余的变量心里就没底

p174

3.定义空函数
……
类型名 函数名()
{}
评：1.过时的写法
2.分类不当，结构混乱，造成误导

该小节的结构是

7.2.2 定义函数的方法

1.定义无参函数
……
2.定义有参函数
……
3.定义空函数
……
评：给读者造成函数有无参函数、有参函数、空函数三类函数的错觉

7.3 调用函数

p174~175

7.3.1 函数调用的形式

……

按函数调用在程序中出现的形式和位置，可以有以下 3 种函数调用方式。

1.函数调用语句

……

2.函数表达式

……

3.函数参数

……

评：完全看不出有必要这样划分

三种形式中的函数调用没有任何本质区别

除非对语句、表达式或实参的概念缺乏准确的理解，否则不可能想到这样划分

因为在三种方式中函数调用都是表达式

p175

调用函数并不一定要求包括分号（如 print_star();），只有作为函数调用语句才需要有分号

评：前面说“不一定要求包括分号”，可“如”的却是有分号，混乱不清自相矛盾，即使作为小学生作文也是不合格的

“只有作为函数调用语句才需要有分号”更是胡扯

for(i=0;i<strlen(s);i++)这个怎么算？

实际上函数调用和分号扯不上半点关系

老谭把他们放在一起说纯属概念混乱纠缠不清

实际参数可以是常量、变量或表达式。

评：就如同说

应该多吃青菜、白菜和蔬菜

可以看出老谭连什么是表达式都不清楚

p176

在调用函数过程中发生的实参与形参间的数据传递，常称为“虚实结合”

评：这回没用 BASIC 冒充 C
改用 FORTRAN 冒充了
为此还特意捏造了一个“虚拟参数”的概念

p177

实际参数可以是常量、变量或表达式。
评：参见 1763 楼
(就如同说
应该多吃青菜、白菜和蔬菜
可以看出老谭连什么是表达式都不清楚)

如果说不懂指针就不完全懂得 C 的话
那么不懂表达式就完全不懂得 C 语言

很不幸
老谭属于后者

如果实参为 int 型而形参 x 为 float 型，或者相反，则按不同类型数值的赋值规则进行转换
评：这个是有前提的
按照老谭写函数声明的风格就不会

字符型与 int 可以互相通用
评：这个是胡扯
不同的字符类型都不能通用
遑论字符类型与 int 类型

调用结束，形参单元被释放。注意，实参单元仍保留并维持原值，没有改变。
实参和形参在内存中占有不同的存储单元
评：压根就不存在实参单元这种东西

p178

赋值语句把这个函数值赋给变量。
评：然而在该书的第 12 页、第 67 页
　　老谭煞有介事地铁口直断

C 语言本身不提供输入输出语句。
评：自己扇自己耳光
　　一点感觉都没有吗
　　境界啊

```
max(int x,int y)
{
   return ((x>y)?x:y);
}
```

评：然而在同一页上居然写着
注意：在定义函数时要指定函数的类型。

例 7.3 将例 7.2 稍作改动，将在 max 函数中定义的变量 z 改为 float 型。函数返回值的类型与指定的函数类型不同，分析其处理方法。
评：例 7.2 的要求是“输入两个整数，要求输出其中的大者。要求用函数找到大者。”
　　相应的 max()函数定义是

```
int max(int x,int y)
{
    int z;
    z=x>y?x:y;
    return(z) ;
}
```

　　怎么可以胡乱“将 max 函数中定义的变量 z 改为 float 型”呢？
　　难道可以不顾要求
　　仅仅为了要说明一下“函数返回值与指定的函数类型不同，按照赋值规则处理”（注“函数类型”也是个错误的术语）
　　就胡乱修改代码？

```
int max(float x,float y)
{
    float z;
    z=x>y?x:y;
    return(z) ;
}
```

评：这改成什么了呢？函数的功能不伦不类简直无法言说

这是一种很奇怪的无病呻吟
不是因为痒了而去挠
而是为了挠一下故意装得有点痒
更何况 return 的表达式与函数返回值的类型不一致本来就是一种很差劲的代码风格

p179

（4）对于不带回值的函数，应当用定义函数为“void”类型（或称“空类型”）。……此时在函数体中不得出现 return 语句。

评：应当用定义函数为“void”类型 ： 这不是话

此时在函数体中不得出现 return 语句：胡扯

float z; //z 为实型变量

评：“实型”是个含义不清的概念

7.4 对被调用函数的声明和函数原型

p180

如果不包含“stdio.h”文件中的信息，就无法使用输入输出库中的函数。同样，使用数学库中的函数。应该用#include <math.h>。

评：这给读者一种错觉

以为编译器提供的库有多个

如果使用用户自己定义的函数，而该函数的位置在调用它的函数（即主调函数）的后面（在同一个文件中），应该在主调函数中对被调用的函数作声明（declaration）。

评：把函数类型声明放在函数体内是一种拙劣的风格

会造成函数体内代码的污染

并且可能需要多次进行不必要的函数类型声明

“函数的位置”是含混不清的说法

应该是函数定义的位置

p181

如果没有对函数 add 的声明，当编译到程序第 7 行时，编译系统无法确定 add 是不是函数名，
评：编译器可以根据运算符()来确定 add 是一个函数
　　尽管应该写函数声明，但不是因为不写编译系统就无法确定 add 是不是函数名

如果不作检查，在运行时才发现实参与形参的类型或个数不一致，出现运行错误。大体上可猜出这是一个输出学号，性别和成绩的函数。
评：运行时可以产生运行错误
　　但不可能发现实参与形参的类型或个数不一致

使用函数原型作声明是 C 的一个重要特点。
评：这个又是在信口开河了
　　C 语言最初并没函数原型这个概念
　　这是向 C++学来的
　　正因为如此
　　这方面无论在理论还是实践方面目前都有些混乱
　　老谭自己不就是把函数声明写得乱七八糟吗
　　一会用函数原型
　　一会用过时的函数声明
　　甚至把许多过时的东西写在书里吗

用函数原型来声明函数，能减少编写程序时可能出现的错误。由于函数声明的位置与函数调用语句的位置比较近，因此在写程序时便于就近参照函数原型来书写函数调用，不易出错。
评：没头没脑
　　怎么就“由于”了呢？

　　函数原型是给谁看的？难道是给程序员吗

p182

对函数的“定义”和“声明”不是一回事。函数的定义是指对函数功能的确立，包括指定函数名，函数值类型、形参及其类型以及函数体等，它是一个完整的、独立的函数单位。而函数的声明的作用则是把函数的名字，函数类型以及形参的类型、个数和顺序通知编译系统，以便在调用该函数时进行对照检查（例如，函数名是否正确，实参与形参的类型和个数是否一致），它不包括函数体。
评：显著的错误是检查“函数名是否正确”，函数声明不检查这个

> 对函数的“定义”和“声明”不是一回事：实际上函数定义也具有函数声明的作用。因此就说明函数名性质这个角度来说，函数的定义也是函数声明的一种
> 如果从狭义的角度来理解的话，即“定义”和“声明”不是一回事，那么后面的叙述——“函数的声明的作用则是把函数的名字，函数类型以及形参的类型、个数和顺序通知编译系统”就是错误的
> 这里混淆了函数声明与函数原型的概念
>
> 此外，“函数值类型”这个说法是可接受的，但后面又说“函数类型”则是错误的。“函数类型”和“函数值类型”根本就不是一回事

用了有意义的参数名有利于理解程序，如：
void print(int num,char XXX,float score);　　(注：XXX 处的原文可能很黄，发不出去，只好以 XXX 代替)
大体上可猜出这是一个输出学号，性别和成绩的函数。
评：这种猜测不但是危险的、盲目的
也是违背软件工程基本原则的
放纵这种猜测
必然导致 BUG 丛生

而且函数原型的目的并不是让人照着写函数调用
其意义主要在于给编译器看

7.5 函数的嵌套调用

p183

7.5 函数的嵌套调用

……

(5)执行 b 函数，如果再无其他嵌套的函数，则完成 b 函数的全部操作；

(6)返回到 a 函数中调用 b 函数的位置；

(7)继续执行 a 函数中尚未执行的部分，直到 a 函数结束；

评：实际上应该是遇到 return 返回或直到函数结束返回

p184

程序分析：可以清楚地看到，在主函数中要调用 max4 函数，因此在主函数的开头要对 max4 函数作声明。……

评：的确“可以清楚地看到”

然而可以更清楚看到的是这种写法把各个函数污染得乱七八糟

程序改进：

（1）……

```
int max2(int a,int b)
{ return (a>b)?a:b; }
```

评：看不懂这究竟在是“改进”还是“改退”

（2）在 max4 函数中，3 个调用 max2 的语句（如 m=max2(a,b);）可以用以下一行代替：

```
m=max2(max2(max2(a,b),c),d);
```

甚至可以取消变量 m,max4 可写成

```
int max4(int a,int b,int c,int d)
{ int max2(int a,int b);
   return max2(max2(max2(a,b),c),d);
}
```

评：max2(max2(max2(a,b),c),d) 显然不如 max2 (max2(a,b) , max2(c,d))

而且这一切有什么必要放在 max4()里完成呢？

放在 main()里难道不是“甚至可以取消”函数 max4()吗？

函数 max4()难道不是无病呻吟吗

不仅 max4()是无病呻吟

max 变量，m 变量也是如此

通过此例，可以知道，不仅要写出正确的程序，还要学习怎样使程序更加精炼、专业和易读。

评：通过此例，只能看到如何使程序更繁复、业余和晦涩

精炼、专业和易读的代码是这样的

```
#include <stdio.h>
int max(int ,int );
int main( void )
{
   int integer_1 , integer_2 , integer_3 , integer_4 ;
   printf( "输入 4 个整数:" ); //假洋鬼子才写 interger
   scanf(    "%d%d%d%d" ,
```

```
                &integer_1,&integer_2,&integer_3,&integer_4);
    printf( "max=%d\n" ,
                max( max(integer_1,integer_2) , max(integer_3,integer_4) ) );
    return 0;
}
int max( int i , int j)
{
     return ( i > j )? i: j ;
}
```

7.6 函数的递归调用

C 语言的特点之一就在于允许函数的递归调用。

评：这是在没话找话吧

很多语言都“允许函数的递归调用”

这怎么成了“C 语言的特点”了呢

p185

```
int f(int x)
{
   int y,z;
   z=f(y);
   return (2*z);
}
```

评：这个例子要多烂有多烂

简直是教唆读者如何写烂代码：

1. x 没用
2. 使用垃圾值 y
3. 无限递归
4. return 一句是无效代码

p186

```
int age(int n)
{
   int c;
   if(n==1)
      c=10;
   else
     c=age(n-1)+2;
   return(c);
}
```

评：return(c);　中的()很搞笑

那个 c 是多余的

可能有人觉得是为了单一出口

我个人觉得那很做作

并没有收获到实惠

p188

```
#include <stdio.h>
int main()
{ int fac(int n);
    int n;
    int y;
    printf("input an integer number:");
    scanf("%d",&n);
    y=fac(n);
    printf("%d=%d\n",n,y);
    return 0;
}
int fac(int n)
{
   int f;
   if(n<0)
      printf("n<0,data error!");
   else if(n==0||n==1)
      f=1;
   else f=fac(n-1)*n;
   return(f);
}
```

评：n<0 的判断本来应该是 main()的任务
　　却硬塞到了 fac()内
　　得到的是一个毫无意义的怪胎
　　如果真的输入一个负数
　　会得到更怪的怪胎

　　input an integer number:-1
　　n<0,data error!-1!=65

　　除了可以让人们观瞻一下代码可以丑陋到何种程度

这个 fac()没有任何价值

更可笑的是老谭居然宣称他用这个代码计算出了 31！

……当 n=31 时，运行正常，输出为

input an integer number:31

31!=738197504

评：用脚后跟也应该能想到 738197504 不是 31！的阶乘吧

p191

```
void hanoi(int n,char one,char two,char three)   //定义 hanoi 函数
      //将 n 个盘子从 one 座借助 two 座，移到 three 座
{
   ……
}
```

评：这是一种令人作呕的注释，污染代码

另：该页上的 main()缺少 return 0;语句

7.7 数组作为函数参数

p192

实参可以是常量、变量或表达式。
评：逻辑错乱
　　茄子、辣椒或蔬菜

凡是变量可以出现的地方，都可以用数组元素代替
评：int i;
　　这个 i 变量出现的地方呢？

数组名也可以作实参和形参
评：数组名根本不能作形参

p193

数组元素可以用作函数实参，不能用作形参
评：这个没什么问题
问题在于下面的“因为”

因为形参是在函数被调用时临时分配存储单元的，不可能为一个数组元素单独分配存储单元
评：这就是在胡扯了
1.auto 变量都是临时的，不仅仅是形参
2.什么时候也不可能为数组元素单独分配存储单元,这跟“形参是在函数被调用时临时分配存储单元的”有个屁关系

在用数组元素作函数实参时，把实参的值传给形参，是“**值传递**”方式
评：就如同在说
今天是中秋，早上太阳是从东边出来的
实际上 C 语言只有值传递
和用什么作实参没有关系

例 7.9 输入 10 个数，要求输出其中值最大的元素和该数是第几个数，
评：题目本身就很蠢
“输入 10 个数”，但没明确什么样的数
“元素”是代码层面的术语，却和问题的描述混为一谈

```
#include <stdio.h>
int main()
    {int max(int x,int y);
     int a[10],m,n,i;
     printf("enter 10 integer numbers:");
     for(i=0;i<10;i++)            //输入 10 个数给 a[0]~a[10]
        scanf("%d",&a[ i]);
     printf("\n");
     for(i=1,m=a[0],n=0;i<10;i++)
     {
         if(max(m,a[ i])>m)
         {m=max(m,a[ i]);
          n=i;
         }
     }
     printf("The largest number is %d\nit is the %dth number.\n",m,n+1);
    }

int max(int x,int y)
```

```
{
    return(x>y?x:y);
}
```

评：对愚蠢的完美诠释

1

根本不需要数组

2

输入 10 个数给 a[0]~a[10]

3

printf("\n");

4

```
            if(max(m,a[ i])>m)
            {m=max(m,a[ i]);
              n=i;
            }
```

调用 max()两次

这里其实可以简洁地写为

```
   for( i = 1 , n = 0 ; i < 10 ; i++ )
        if(max(a[n] , a[ i]) > a[n] )
            n = i ;
```

当然这种做法还是比较啰嗦

5

缺少 return 0

p194

当然，本题可以不用 max 函数求两个数中的最大数，而在主函数中直接用 if(m>a[i])来判断和处理。本题的目的是介绍如何用数组元素作为函数实参

评：用愚蠢的数据结构和愚蠢的算法介绍数组元素作为函数实参
读者最多只能学会怎样愚蠢地用数组元素作实参

用数组元素作实参时，向形参变量传递的是数组元素的值，而用数组名作函数实参时，向形参（数组名或指针变量）传递的是数组首元素的地址。

评：这个是误导
不管什么作实参，传的都是“值”，数组名作实参也不例外
此外形参不可能是数组名

```
#include <stdio.h>
int main( )
{ float average(float array[10]);
    float score[10],aver;
    int i;
    printf("input 10 scores:\n");
    for(i=0;i<10;i++)
      scanf("%f",&score[i]);
    printf("\n");
    aver=average(score);
    printf("average score is%5.2f\n",aver);
    return 0;
}

float average(float array[10])
{int i;
float aver,sum=array[0];
for(i=1;i<10;i++)
     sum=sum+array[i];
aver=sum/10;
return(aver);
```

}

评：float average(float array[10])
中的 10 非常愚蠢

p195

用数组名作函数参数，应该在主调函数和被调函数分别定义数组，例中 array 是形参数组名，score 是实参数组名，分别在其所在函数中定义，不能只在一方定义。

评：没弄清什么叫定义数组

在定义 average 函数时，声明数组的大小为 10，但在实际上，指定其大小是不起任何作用的，
评：不起作用为什么要写？
压根就不该写

形参数组可以不指定大小，在定义数组时在数组名后面跟一个空的方括号，如：
float average(float array[])

评：不指定大小是对的
但不指定数组的尺寸则是误导
写出的函数是半身不遂的函数

例 7.11 有两个班级，分别有 35 名和 30 名学生，调用一个 average 函数，分别求这两个班的学生的平均成绩。
评：调用一个 average 函数，分别求这两个班的学生的平均成绩：病句

实际的代码居然是

float score1[5]=……
float score2[10]=……

巨无聊

拙劣的题目可以伤害正常的逻辑思维能力

p196

用数组名作函数实参，不是把数组元素的值传递给形参，而是把实参数组的首元素的地址传递给形参数组，这样两个数组就共占同一段内存单元。
评：“两个数组”，是一种捏造

用数组名作函数实参，不是把数组元素的值传递给形参，而是把实参数组的首元素的地址传递给形参数组，这样两个数组就共占同一段内存单元。
评：“两个数组”，是一种捏造

p197

可以用多维数组名作为函数的实参和形参，在被调用函数中对形参数组定义时可以指定每一维的大小，也可以省略第一维的大小说明。例如：

int array[3][10];

或

int array[][10];

评：写那个 3 是误导，因为根本没必要

此外，形参声明是根本不可能带那个“;”的，除非是古代的 C 语言

p198

由于形参数组与实参数组类型相同
评：这个错的没边了
　　它们的类型压根就不一样

在第 2 维大小相同的前提下，形参数组的第 1 维可以与实参数组不同。例如，实参数组定义为
　int score[5][10];
而形参数组定义为
　int score[][10];
或
　int score[8][10];均可以。
评：这是把无聊当知识卖弄
　　写那个 8 毫无意义

这时形参数组和实参数组都是由相同类型和大小的一维数组组成的
评：形参根本就不可能是数组（即使形式上是，但那叫不完整类型）
　　更不可能由一维数组组成

int max_value(int array[][4])
……
评：这是在教人写半身不遂的函数
　　这种函数没有价值和适应性
　　因为它本身依赖于一个莫名其妙的 Magic number

7.8 局部变量和全局变量

p199

定义变量可能有 3 种情况：
（1）在函数开头定义；
（2）在函数内的复合语句内定义；
（3）在函数的外部定义。
评：错
　　这表明老谭所号称的这本书是“按照 C 语言的新标准 C99 进行介绍”是蒙人的
　　老谭连 C99 门都没摸到

　　实际上只有两种可能：函数内和函数外

P200

(4)在一个函数内部，可以在复合语句中定义变量，这些变量只在本复合语句中有效。这种复合语句也称为“分程序”或“程序块”。
在函数内定义的变量是局部变量，在函数之外定义的变量是全局变量。
评：“**分程序**”或“**程序块**”是老谭捏造的新概念
第一不必要，因为有复合语句的概念，没必要再给它两个别名
第二不合理，因为源程序的下一层是函数定义，再下一层才是复合语句的层次，所以说“复合语句”是分程序非常荒谬
这种复合语句也称为“**分程序**”或“**程序块**”。：语文问题。既然是“这种”，就意味着还有“那种”，实际上根本没有。

P201

int p=1,q=5;　　//定义外部变量。
int c1,c2;　　//定义外部变量。
p,q,c1,c2 都是全局变量，
评：概念不统一

设置全局变量的作用是增加了函数间数据联系的渠道。
评：这个说法违背结构化程序设计原则，是误导

如果在一个函数中改变了全局变量的值，就能影响到其他函数中全局变量的值。
评：“其他函数中全局变量”，这叫什么话？

由于函数的调用只能带回一个函数返回值，因此有时可以利用全局变量来对增加函数间的联系渠道，通过函数调用能得到一个以上的值。
评：这个外行，竟然用这种垃圾理论来蒙学生
完全不顾结构化程序设计最基本的原则

在 C 程序设计人员中有一个习惯（但非规定），将全局变量名的第个字母用大写表示。
评：有这事儿吗？
从没听说过
老谭捏造也不是第一次了

例 7.14 有一个一维数组，内放 10 个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分。
评：题目要求很变态

解题思路：调用一个函数可以得到一个函数返回值，现在希望通过函数调用能得到 3 个结果。可以利用全局变量来达到此目的。
评：解题思路更变态

P202

```
#include <stdio.h>
float Max=0,Min=0;
int main()
{ float average(float array[],int n);
   float ave,score[10];
   int i;
   printf("Please enter 10 scores:");
   for(i=0;i<10;i++)
        scanf("%f",&score[i]);
   ave=average(score,10);
   printf("max=%6.2f\nmin=%6.2f\naverage=%6.2f\n",Max,Min,ave);
   return 0;
}
float average(float array[],int n)
{ int i;
   float aver,sum=array[0];
   Max=Min=array[0];
   for(i=1;i<n;i++)
      {if(array[i]>Max)Max=array[i];
       else if(array[i]<Min)Min=array[i];
       sum=sum+array[i];
      }
   aver=sum/n;
   return(aver);
}
```

评：代码则令人作呕
这是在教唆别人学坏
那两个外部变量要多恶心有多恶心

由于 Max 和 Min 是全局变量
评：C 语言只有外部变量，没有全局变量这个概念

建议不在必要时不要使用全局变量，原因如下：
①全局变量在程序的全部执行过程中都占用存储单元，而不是仅在需要时才开辟单元。
评：1
全局变量为外部变量之误
2
在程序的全部执行过程中都占用存储单元，并非不应该使用外部变量的原因

在某些语言里，全局变量的意思是只要定义了，其他模块就可见

在 C 语言中没有这样的东西

外部变量有两种 extern 或 static
前一种其他模块可以用，但在使用前应该声明（这个应该是和全局变量的一个最主要的区别）
后一种其他模块不可以用

使用外部变量，最主要的问题是破坏了良好的程序结构，使程序的各个部分联系更紧密了

如果全部执行过程都占有单元也算一个缺陷的话，那 static 局部变量怎么说，也不应该？有些变量确实需要一直占用内存

P203

②它使函数的通用性降低了
评：应该是这样的函数没有通用性

例 7.15
评：对全局变量（外部变量）b 的作用范围的标示是错误的

7.9 变量的存储方式和生存期

P204

变量可以分为全局变量和局部变量。
评：全局变量是个既不正确也很糟糕的概念
在有的语言里
全局变量的意思是只要定义，就全局可见
但 C 语言中外部变量并没有这样的性质
首先，外部变量的作用域只是定义点到所在源文件(或编译单元结束)
其他源文件使用前必须声明而不是可以直接拿过来使用
而且有的外部变量（static）其他源文件根本不能使用
即使在所在文件，外部变量也可能需要声明才能使用（定义在使用点后面的情形）
“全局”这两个字含糊掩盖了以上所提到的外部变量的精确含义
对于初学者来说尤其如此

有的变量则是在调用其所在的函数时才临时分配内存单元，而在函数调用结束后该存储单元就马上释放了，变量不存在了。
评：这个说法是错误的
局部非 static 变量不是在调用其所在函数后存在的
应该是程序执行到其所在块声明点之后才存在的
释放也是如此

变量的存储有两种不同的方式：静态存储方式和动态存储方式
评：这个基本属于捏造
变量的存储方式没什么静态动态之分
变量的生命期有 static storage duration 和 automatic storage duration

静态存储方式是指在程序运行期间由系统分配的存储空间的方式，而动态存储方式则是在程序运行期间根据需要进行动态的分配空间的方式
评：“静态存储方式是指在程序运行期间由系统分配的存储空间的方式”
这个是错误的，static storage duration 的变量在程序运行前就存在了
这里所谓的“由系统分配”是在打马虎眼，搞不清这是什么系统
如果是编译系统，那么后者也是由系统分配

先看一下内存中供用户使用的存储空间的情况。这个存储空间可以分为 3 部分：
（1）程序区。
（2）静态存储区。
（3）动态存储区。
评：这个说法没有任何根据，更荒谬的是

在动态存储区存放以下数据：
（1）函数形式参数。在调用函数时给形参分配空间。
（2）函数中定义的没有用关键字 static 声明的变量，即自动变量。
（3）函数调用时的现场保护和返回值等。
评：（1）、（2）都是错误的，因为还有 register
（3）是一种没有根据的猜测或根据个别实现的情况作出的不完全归纳

P205

即长如果用户不指定
静态的（statis）
根据变量的存储类别，可以知道变量的作用域和生存期。
评：变量的存储类别与变量的作用域没有半毛钱关系

C 的存储类别包括 4 种
评：实际上是 5 种

函数中的局部变量，如果不专门声明为 static(静态)存储类别，都是动态地分配存储空间的，数据存储在动态存储区中。
评：这个说法是错误的
“动态存储区”是老谭捏造的一块“内存”区域
而非 static 的局部变量，未必放在内存之中

函数中的形参和在函数中定义的局部变量（包括在复合语句中定义的局部变量）……在函数调用结束时就

自动释放这些存储空间。
评：块作用域的 auto 局部变量不是在函数结束时释放，而是在块结束时释放

关键字“auto”可以省略，不写 auto 则隐含指定为“自动存储类别”
评：错
应该是没声明存储类别的局部变量为自动存储类别

P206

static c=3;
评：老谭号称是按照 C99 写的
但这种写法在 C99 是禁止的
一句话，挂羊头卖狗肉

对应的运行结果也是货不对板

```
7
8
9
-
```

评：最后“-”莫名其妙

P207

什么情况下需要用局部静态变量呢？……例如可以用下面的方法求 n!。
例 7.17 输出 1 到 5 的阶乘值。

```
#include <stdio.h>
int main( void )
{int fac(int n);
   int i;
   for(i=1;i<=5;i++)
      printf("%d!=%d\n" ,i,fac(i) );
   return 0;
}
int fac(int n)
{static int f=1;
   f=f*n;
   return(f);
}
```

评：误导
这不是正确使用 static 局部变量的方法
因为这个 fac()函数和那啥一样
只能一次性使用

甚至连一次性都算不上
如果题目要求求 1、3、5 的阶乘
立刻就会发现 fac()的残疾

如果函数中的变量只被引用而不改变值，则定义为静态局部变量（同时初始化）比较方便，以免每次调用时重新赋值。

评：既然不改变，为什么不用常量？外部变量等技术手段？
究竟何时使用局部静态变量以及如何使用，看来老谭根本不清楚
所以无论给出的例子还是讲解都是错误的

用静态存储要多占内存(长期占用不释放，而不是像动态存储那样一个单元可以先后为多个变量使用，节约内存)……而且降低了程序的可读性……

评：降低程序可读性是无稽之谈
动态存储节约内存也是无稽之谈

评：注意这里的静态存储与外部还是局部变量无关
局部变量也可以是静态生命期
可笑的是你把静态存储理解为非局部变量

P208

寄存器存储在 CPU 中的寄存器中

评：这根本就不是话

全局变量都是存放在静态存储区中的。因此它们的生存期是固定的，存在于程序的整个运行过程。

评：1.全局变量是外部变量之误
2.外部变量存放在哪里跟程序员无关，这是编译器作者的事情
3.外部变量的生存期和它存放在哪里没有因果关系
总之一堆错话与废话
实际上只要一句话就足够了：外部变量具有静态存储持续期（static storage duration），在程序运行过程中一直存在。

一般来说，外部变量是在函数外部定义的全局变量。

评：这个胡扯太过分了
如果“外部变量是在函数外部定义的全局变量”
那么老谭能否说说什么是在函数内部定义的全局变量

一般来说，外部变量是在函数外部定义的全局变量。它的作用域是从变量的定义处开始，到本程序文件的末尾。

评：变量的作用域并不是从变量的定义处开始
而是从变量的声明处开始

但有时设计人员希望能扩展外部变量的作用域。

评：不存在所谓作用域扩展

作用域是声明决定的

1. 在一个文件内扩展外部变量的作用域

评：这是一个伪命题

在 C 里根本就不存在扩展变量作用域

变量的作用域决定于声明的位置

如果外部变量不在文件的开头定义，其有效的作用范围只限于定义处到文件结束。

评：1.外部变量的作用域取决于声明的位置

2.那个“如果”莫名其妙，外部变量的作用域无论你“如果”还是不“如果”都是从声明处到文件的结束

P208~212

7.9.3 全局变量的存储类别。

1.在一个文件内扩展外部变量的作用域

2.将外部变量的作用域扩展到其他文件

3.将外部变量的作用域限制在本文件中

评：大标题是“存储类别”

下面三个下标题却都是讲“作用域”

思维混乱竟至于此

P209

```
#include <stdio.h>
int main()
{int max();
extern int A,B,C;  //把外部变量 A,B,C 的作用域扩展到从此处开始
printf("Please enter three integer numbers:");
scanf("%d%d%d",&A,&B,&C);
printf("max is %d\n",max());
return 0;
}

int A,B,C;

int max()
{int m;
m=A>B?A:B;
if(C>m)m=C;
return(m);
}
```

评：所谓“把外部变量 A,B,C 的作用域扩展到从此处开始”是误导，实际上是在 main()中声明这三个变量，以便在该函数内使用

代码风格拙劣，思想垃圾，简直是教唆别人如何写垃圾代码

由于 A,B,C 是外部变量，所以在调用 max 函数时用不到参数传递。在 max 函数中可直接使用外部变量 A,B,C 的值。

评：这是在教读者如何用最荒唐的方法使用外部变量

用 extern 声明外部变量时，类型名可以写也可以不写。例如，“extern int A,B,C;”也可以写成“extern A,B,C;”。因为它不是定义变量，可以不指定类型，只须写出外部变量名即可。

评：这是在信口开河，胡说八道

P210

extern Num

评：错误性质同前

例 7.19 给定 b 的值，输入 a 和 m，求 a*b 和 a^m 的值。

评：这也能叫题目？

文件 file1.c

```
int A;
int main()
{int power(int);
   int b=3,c,d,m;
   printf("enter the number a and its power m:\n");
   scanf("%d,%d",&A,&m);
   c=A*b;
printf("%d*%d=%d\n",A,b,c);
d=power(m);
printf("%d**%d=%d\n",A,m,d);
return 0;
}
```

文件 file2.c

```
extern A;
int power(int n)
{int i,y=1;
    for(i=1;i<=n;i++)
        y*=A;
    return(y);
}
```

评：extern A 的错误不算

这个例题成功地应用外部变量把代码写得无比垃圾

没有足够多的愚蠢是无论如何也写不出如此垃圾的代码的

首先 3 是常量，却画蛇添足地赋值给 b
A,b,c,d 无比垃圾式的命名
毫无必要的变量 c,d 及赋值
毫无通用性的函数 power()
怪异的 return(y);
业余的循环 for(i=1;i<=n;i++)

实际上，在编译时遇到 extern 时，先在本文件中找外部变量的定义，如果找到，就在本文件扩展作用域；如果找不到，就在连接时从其他文件中找外部变量的定义。如果从其他文件中找到了，就将作用域扩展到本文件；如果再找不到就按出错处理。
评：“先后理解成在限定一个翻译单元内的代码顺序解析能说得通。当然先定义后使用的说法是错的，应该是先声明后使用。”
这个完全同意。只能在同一个翻译单元内谈先后
而且应该是先声明后使用而不是先定义后使用
不止如此，至少还是有遗漏的。

```
int(*foo(void))[3]
{
/*return...*/
}
```

这里的“类型”是都能写在函数名左边的“类型名”吗？
谢谢补充

P212

对一个数据的定义，需要指定两种属性：数据类型和存储类别，分别使用两个关键字。
评：错乱不堪
数据包括常量与变量
常量即不存在定义的问题
也无需指定存储类别
对于变量来说
存储类别也不一定通过关键字指定
"分别使用两个关键字"更是胡扯
至少还有两种情况：一个关键字不足以说明类型，仅用关键字不足以说明类型

可以用 extern 声明已定义的外部变量，例如：
extern b;
评：这个错误已经出现至少十几次了

自动变量，即动态局部变量（离开函数，值就消失）
寄存器变量（离开函数，值就消失）
评：这里有两个错误
1.不是值消失，而是对象消失

2.“离开函数”这个说法是错误的，应该是离开语句块

从作用域角度分，有局部变量和全局变量。它们采用的存储类别如下：
评：1.全局变量是个错误的概念
2.存储类别是个含糊不清的概念

寄存器变量（离开函数，值就消失）
评：“函数”、“离开”和“值”这三个概念都有错误
分别应该是“语句块”、“结束”和“数据对象”

静态外部变量
全局变量
外部变量
评：首先“全局变量”是无中生有
其次把全局变量分为静态外部变量和外部变量很错乱
第三和 200 页老谭给出的定义自相矛盾

从变量存在的时间（生存期）来区分，有动态存储和静态存储两种类型。
评：“动态存储”的说法值得商榷
根据 2006 楼 幻の上帝 的帖子
在 C99 中
There are three storage durations: static, automatic, and allocated.
如果“动态”是来自 automatic 的话，这个翻译明显不当
而若根据《C 语言参考手册》
storage duration 可分为 static extend,local extend 和 dynamic extend
这里的 dynamic 倒是可以译为“动态”，但其含义和老谭所说的截然不同

P213

(本函数内有效)
评：这个错误出现多次

静态外部变量(本文件内有效)
评：这个说法是错误的
在声明点之前无效

外部变量(用 extern 声明后，其他文件可引用)
评：明显的词不达意，该回去重修小学语文

图 7.20 是生存期的示意图
评：图 7.20 中，把 f2()函数中的局部静态变量 c 的生存期从 f2()被调用开始算起，是错误的

如果一个变量在某个文件或函数范围内是有效的，就称该范围为该变量的作用域，在此作用域内可以引用

该变量，在专业书中称变量在此作用域内“可见”
评：这个是似是而非、不求甚解的蒙人的说法
作用域和可见性是两回事
在作用域内未必可见

表 7.2 表示各种类型变量的作用域和存在性的情况。
评：表 7.2 是非常荒谬的一张表格
比如
认为外部变量的作用域涵盖函数内与函数外
实际上函数内与函数外都可能不是外部变量的作用域
因为外部变量的作用域从声明处开始到源文件结束或到所在块的结束处
因为外部变量的作用域从定义处开始到源文件结束
定义前面的部分可能不属于其作用域
再比如
以函数内外来描述变量的存在性也很荒谬

7.10 关于变量的声明和定义

P214

在第 2 章介绍了如何定义一个变量。
评：第 2 章并没有介绍如何定义变量

从第 2 章已经知道，一个函数一般由两部分组成：声明部分和执行语句。
评：1
第 2 章并没有介绍函数由声明部分和执行语句两部分组成
2
函数也并不是由声明部分和执行语句两部分组成

函数的声明是函数的原型
评：这个恰恰说反了
函数的原型是函数的声明(的一种)

对被调用函数的声明是放在主调函数的声明部分中的
评：语法上没有错误
但实际上是作茧自缚
不仅可能不必要地多次声明
而且污染主调函数的代码

对“int a;” 而言，它既是声明，又是定义；而对“extern a;”而言，它是声明而不是定义。
评：1.
如果
int a;

的位置在{}之内，可以说“它既是声明，又是定义”
如果在{}之外，它可能不是定义
2.
extern a;
这种写法本身是错误的
在 C89 中只有 int 类型可以这样写
在 C99 中不容许省略 int
而且，不可能仅仅根据具有 extern 这个 storage-class specifiers 这一点就断定它一定是声明而非定义

```
int main()
{
   extern A;        //是声明，不是定义，声明将已定义的外部变量 A 的作用域扩展到此
   .
   .
   .
   return 0;
}

int A;                  //是定义，定义 A 为整型外部变量
```

评：extern A;　　//是声明，不是定义，声明将已定义的外部变量 A 的作用域扩展到此：
这个确实是声明，不是定义。但并不是“将已定义的外部变量 A 的作用域扩展到此”，这个标识符 A 本身有自己的作用域，即声明点到}。这个声明同时说明了 A 的定义在别处。此外，这个声明应该写明 A 的类型

int A;　　//是定义，定义 A 为整型外部变量：
这个说法是武断的。这个可能是定义也可能不是
例如
file1.c

```
int main( void )
{
extern int A;
printf("%d\n",A);
return 0;
}
int A;//这个不是定义
```

file2.c

```
int A=3;//这个才是定义
.
```

在同一文件中，可以有多次对同一外部变量的声明，它的位置可以在函数之内（哪个函数要用就在哪个函数中声明），也可以在函数之外（在外部变量的定义点之前）。
评：话确实可以这样说
但其出发点则是愚蠢的

尤其是“哪个函数要用就在哪个函数中声明”

7.10 内部函数和外部函数

P215

在定义内部函数时，在函数名和函数类型的前面加 static，即：
static 类型名 函数名(形参表);
评：这不是定义

通常把只能由本文件使用的函数和外部变量放在文件的开头，前面都冠以 static 使之局部化，其他文件不能引用。
评：毫无道理
莫名其妙
这不是“通常”的做法，而是“反常”的做法

P216

例 7.20 有一个字符串，内有若干个字符，现输入一个字符，要求程序将字符串中该字符删去。用外部函数实现。
评：“内有若干个字符”这种废话绝对是谭氏体
字符串里除了有若干个字符还能有什么呢

P216~217

file1.c(文件 1)

```
#include <stdio.h>
int main()
{
   extern void enter_string(char str[]);
   extern void delet_string(char str[],char ch);
   extern void print_string(char str[]);
   char c,str[80];
   enter_string(str);
   scanf("%c",&c);
   delet_string(str,c);
   print_string(str);
   return 0;
}
```

file2.c(文件 2)

```
void enter_string(char str[80])
```

```
{
    gets(str);
}
```

file3.c(文件 3)

```
void delet_string(char str[],char ch)
{int i,j;
for(i=j=0;str[ i]!='\0';i++)
        if(str[ i]!=ch)
           str[j++]=str[ i];
str[j]='\0';
}
```

file4.c(文件 4)

```
void    print_string(char str[])
{
    printf("%s\n",str);
}。
```

评：这段源程序有下面这些毛病

把

```
extern void enter_string(char str[]);
extern void delet_string(char str[],char ch);
extern void print_string(char str[]);
```

写在 main()中

污染了 main()

而且这样很难保证函数类型声明的一致性

P217

一般都省写 extern，例如例 8.20 程序中 main 函数的第一个函数声明可写成

```
void enter_string(char str[]);
```

评：“一般都省写 extern”是误导，C 语言鼓励程序员明确

此处是第 7 章，居然扯出个“例 8.20”

第 8 章 善于利用指针

P220

将地址形象化地称为“指针”

评：这是胡扯
多数人指针学得稀里糊涂不是因为别的
就是因为他们把指针和地址混为一谈

地址指向该变量单元。
评：关于地址，老谭是这样说的“内存区的每一个字节有一个编号，这就是‘地址’”

然而，变量单元所占的可能不只一个字节
所以地址指向变量单元的说法是荒谬的

P220~221

由于通过地址能找到所需的变量单元，因此说，地址指向该变量单元。
评：车轱辘话来会说
此外通过地址并不能找到所需的变量单元

P221

一个变量的地址称为该变量的“指针”
评：这是自己扇自己耳光
前面老谭说“内存区的每一个字节有一个编号，这就是‘地址’”
变量可能占几个字节，哪来的地址？

这句话的错误还体现在
指针未必和变量相关

如果有一个变量专门用来存放另一个变量的地址（即指针），则它称为“指针变量”
评：扯淡
再次混淆指针与地址这两个概念
“专门用来存放另一个变量的地址”，井蛙之见
double (*p)(double)=sin;
p 算不算是指针变量？
如果不算它算什么？
如果算，它存放了哪个变量的地址啦？

指针变量就是地址变量
评：C 语言中压根就没有“地址变量”这种东西

指针变量的值是地址（即指针）
评：再次混淆指针与地址这两个概念
这是把读者往沟里带

P222

第 8 行输出*pointer_1 和*pointer_2 的值。其中的“*”表示“指向”
评：“*”表示“指向”是胡扯
　　在这里这是一个运算符

P223

定义指针变量的一般形式为
类型名 *指针变量名;
评：void (*(*p) (void *(*) (char *, long , long))) (char *, long , long);
　　这个不知道老谭如何解释

int **pointer_1,*pointer_2;
左端的 int 是在定义指针变量时必须指定的“基类型”。
评：“基类型”是臆造出的概念

P224

(int *),(float *),(char *)是 3 种不同的类型
评：(int *),(float *),(char *) 根本就不是类型

指针变量中只能存放地址（指针），不要将一个整数赋给一个指针变量。如：
*pointer_1=100;
…系统…判为非法。
评：莫名其妙
　　这根本不是“将一个整数赋给一个指针变量”
　　也根本不会“判为非法”

在引用指针变量时，可能有 3 种情况：
（1）给指针变量赋值。……
（2）引用指针变量指向的变量。
（3）引用指针变量的值。如：
printf("%o",p);
评：搞笑。岂止有 3 种情况
　　printf("%o",p); 是野路子写法， "%o"是转换输出 unsigned 类型数据的

P225

例 8.2 输入 a 和 b 两个整数，按先大后小的顺序输出 a 和 b。

#include <stdio.h>

```
int main()
{int *p1,*p2,*p,a,b;
printf("please enter two integer numbers:");
scanf("%d,%d",&a,&b);
p1=&a;
p2=&b;
if(a<b)
    {p=p1;p1=p2;p2=p;}
printf("a=%d,b=%d\n",a,b);
printf("max=%d,min=%d\n",*p1,*p2);
return 0;
}
```

评：交换两个指针完全是脱裤子放屁

```
#include <stdio.h>

int main( void )
{
    int *p1,*p2,a,b;

    printf("please enter two integer numbers:");
    scanf("%d%d",&a,&b);

    if(a<b)
    {
        p1 = &a;
        p2 = &b;
    }
    else
    {
        p2 = &a;
        p1 = &b;
    }

    printf("a=%d,b=%d\n",a,b);
    printf("max=%d,min=%d\n",*p1,*p2);

    return 0;
}
```

P226

例 8.3 题目要求同 8.2……

```
#include <stdio.h>

int main()
{void swap(int *p1,int * p2);
   int a,b;
   int *pointer_1,*pointer_2;
   printf("please enter a and b:");
   scanf("%d,%d",&a,&b);
   pointer_1=&a;
   pointer_2=&b;
   if(a<b)swap(pointer_1,pointer_2);
   printf("max=%d,min=%d\n",a,b);
   return 0;
}
void swap(int *p1,int * p2)
{int temp;
   temp=*p1;
   *p1=*p2;
   *p2=temp;
}
```

评：画蛇添足，脱裤子放屁

8.2.4 指针变量作为函数参数

评：标题就有问题

狭隘

P227

这个改变不是通过将形参值传回实参来实现的。

评：这是废话

任何时候都不可能“将形参值传回实参”

P228

为了使在函数中改变了的变量能被主调函数 main 所用，不能采取上述把要改变值的变量作为实参的办法，而应该用指针变量作为函数参数，在函数执行过程中使指针变量所指向的变量值发生变化，函数调用结束后，这些变量值的变化依然保留下来，这样就实现了“通过调用函数使变量的值发生变化，在主调函数（如 main 函数）可以使用这些改变了的值”的目的。

评：这文字也太垃圾了吧

这就是通俗易懂？
我是看不懂

P228~229

例 8.4 对输入的两个整数按大小顺序输出。

```
int main()
{void swap(int *p1,int * p2);
int a,b;
int *pointer_1,*pointer_2;
printf("please enter a and b:");
scanf("%d,%d",&a,&b);
pointer_1=&a;
pointer_2=&b;
if(a<b)swap(pointer_1,pointer_2);
printf("max=%d,min=%d\n",a,b);
return 0;
}
void swap(int *p1,int * p2)
{int *p;
p=p1;
p1=p2;
p2=p;
}
```

评：老谭无中生有地批评这个代码企图通过“形参 p1 与 p2 将它们的值（是地址）传回实参 pointer_1 和 pointer_2”，“但是，这是办不到的”，因而“在输入 5,9 之后程序的实际输出为 max=5,min=9”
这是在放空炮
因为即使形参的值能够传回实参，这个程序“在输入 5,9 之后程序的实际输出”依然“为 max=5,min=9 ”

P228~229

函数的调用可以(而且只可以)得到一个返回值（即函数值），而使用指针变量作参数，可以得到多个变化了的值。
评：关公战秦琼
驴唇不对马嘴

P230

```
int a,b,c,*p1,*p2,*p3
```

评：p1,p2,p3 三个变量定义得毫无意义

P231

所谓数组元素的指针就是数组元素的地址。
评：再次混淆指针与地址的概念
　　把读者往沟里带

可以用一个指针变量指向一个数组元素
评：无的放矢的废话
引用数组元素可以用下标法，也可以使用指针法……使用指针法能使目标程序质量高（占内存少，运行速度快）
评：这是没有任何依据的以讹传讹

在 C 语言中，数组名（不包括形参数组名，形参数组并不占据实际的内存单元）代表数组中首元素（即序号为 0 的元素）的地址
评：大谬特谬
　　数组名和指针是两回事
　　此外，所谓的形参数组根本就不存在

注意：数组名不代表整个数组，只代表数组首元素的地址。
评：吐血
　　纯粹是狗屁不通

前已反复说明指针就是地址。
评：说指针就是地址说明不懂指针
　　在教科书里这样说应该被枪毙

P232

两个指针相减，如 p1-p2(只有 p1 和 p2 都指向同一数组中的元素时才有意义)。
评：这个说法是错误的

p+1 所代表的地址实际上是(p+1)*d,d 是一个数组元素所占的字节
评：小学数学没学好

这里需要注意的是 a 代表数组首元素的地址，a+1 也是地址，它的计算方法同 p+1，即它的实际地址为(a+1)*d。
评：首先，a 并不代表数组首元素的地址
　　其次，a+1 也不是地址。在 C 语言中压根就没有地址这种数据类型
　　至于(a+1)*d 已经荒谬的没边了

[]实际上是变址运算符
评：这是无中生有的捏造

把 C 给变质了

p2 指向实型数组元素
评：C 语言里压根就没有“实型”

P233

引用一个数组元素，可以用下面两种方法：
（1）下标法，如 a[i]形式；
（2）指针法，如*(a+i)或*(p+i)。其中 a 是数组名，p 是指向数组元素的指针变量，其初值 p=a。
评：这种浮于表面而不及实质的、愚蠢而又自作聪明的不全面且无意义的归纳会使读者永远无法理解什么是指针
估计老谭自己也是稀里糊涂，要是看到 p[i]恐怕会吓一跳吧

printf("%\n");
评：这是神马东东
估计又是一个未定义行为

P233~234

(1)
```
int a[10];
int i;
for(i=0;i<10;i++)
   printf("%d",a[ i ]);
```

(2)
```
int a[10];
int i;
for(i=0;i<10;i++)
   printf("%d",*(a +i));
```

(3)
```
int a[10];
int *p;
for(p=a;p<(a+10);p++)
   printf("%d", *p );
```
第(3)种方法比第(1)和第(2)种方法快
评：我认为这种说法根据不足
而且也没什么意义

P235

指针变量 p 可以指向数组以后的存储单元。
评：严重的误导
　　后果很严重
　　最多可以指向数组之后的第一个数组元素类型的对象

P236

……定义数组时指定它包含 10 个元素……如果在程序中引用数组元素 a[10]……但 C 编译程序并不认此为非法。系统把它按*(a+10)处理
评：引用 a[10]是未定义行为
　　“系统把它按*(a+10)处理”的说法是武断的，毫无依据

*p++;
由于++和*同优先级
评：++和*优先级并不相同

P237

将++和--运算符用于指针变量十分游戏，可以使指针变量自动向前或向后移动
评：“自动”二字滑稽
　　什么叫“自动”移动
　　是不是还有“被动”移动啊

```
p=a;
while(p<a+100)
printf("%d",*p++);
```
评：风格拙劣

```
p=a;
while(p<a+100)
　　{printf("%d",*p);p++;}
```
评：风格拙劣

P238

arr 为形参数组名

评：形参根本就不可能是数组

常用这种方法通过调用一个函数来改变实参数组的值。
评：函数永远不可能改变实参的值
"数组的值"也是莫名其妙的说法
什么叫数组的值？
数组的值用什么表示
如果说数组名表示数组的值的话，那么数组的值根本不可能以任何方式被改变
如果说数组所占据内存的内容代表数组的值，那么数组绝对不可能是实参

P239

表 8.1 以变量名和数组名作为函数参数的比较

参数类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否改变实参的值	不能改变实参变量的值	能改变实参数组的值

评：这是脱离本质看现象
实际上实参是一个表达式，不能分为变量名和数组名
形参也不存在数组名，形参只是一个变量
无论实参如何传递的都是一个值
通过函数调用绝对不可能改变实参的值

评：那段代码还不算最坏的
下面这段居然能出现在教科书上实在是令人拍案惊奇

```
void fun(arr[    ],int n)
{ printf("%d\n",*arr);
    arr=arr+3;
    printf("%d\n",*arr);
}
```

在用数组名作为函数实参时，既然实际上相应的形参是指针变量，为什么还允许使用形参数组的形式呢？这是因为在 C 语言中用下标法和指针法都可以访问一个数组
评：问题本身荒唐
回答逻辑错乱
这个原因应该问问 K&R
老谭不要越位瞎说一通

形参数组与实参数组共占同一段内存。
评：荒谬
形参根本就不可能是数组

在函数调用进行虚实结合后
评：C 语言根本就没有虚实结合这一说

将 a[int(n-1)/2]与 a[n-int((n-1)/2)-1]对换
评：这是什么污七八糟的烂东西
编译都不可能通过

例 8.8 将数组 a 中的 n 个整数按相反顺序存放，……
再将 a[i] 与 a[j]对换，直到 i=(n-1)/2 为止
评：“直到 i=(n-1)/2 为止”在有些情况下多做了一次无用的交换

实参用数组名 a，形参可以用数组名，也可以用指针变量名。
评：这个说法的荒唐之处在于

```
int i;

f(&i)

f(int a[])
{
……
}
```

所以所谓的形参数组，根本就是无中生有的说法

P239~240

例 8.8 将数组 a 中的 n 个整数按相反顺序存放，……

```
#include <stdio.h>
int main( void )
{void inv(int x[],int n);
int i, a[10]={3,7,9,11,0,6,7,5,4,2};
printf("The original array:\n");
for(i=0;i<10;i++)
    printf("%d ",a[ i ]);
printf("\n");
inv(a,10);
printf("The array has been vnverted:\n");
for(i=0;i<10;i++)
    printf("%d ",a[ i ]);
printf("\n");
return 0;
}

void inv(int x[ ],int n)                                    //形参 x 是数组名
{int temp,i,j,m=(n-1)/2;
for(i=0;i<=m;i++)
```

```
    {j=n-1-i;
     temp=x[ i];x[ i]=x[j];x[j]=temp;
    }
return ;
}
```

评：可呕可吐之处颇多

inv()函数调淹没在脑满肠肥的琐碎且无聊的细节之中

main()中存在两段相同的代码

m 变量明显多余

"形参 x 是数组名"的说法是胡扯

……

```
void inv(int x[ ],int n)
{int temp,i,j,m=(n-1)/2;
for(i=0;i<=m;i++)
    {j=n-1-i;
     temp=x[ i];x[ i]=x[j];x[j]=temp;
    }
return ;
}
```

评：以 n 为 3 时走查一下

i	j	m
0	2	1
1	1	1

不难发现，这时 temp=x[i];x[i]=x[j];x[j]=temp; 是在毫无必要地 x[1]自己在和自己交换

这段代码至少应该写成

```
void inv(int x[ ],int n)
{
   int temp,i,j;
   for( i = 0 , j = n - 1 ; i < j ; i++ , j-- )
   {
        temp=x[i];
        x[i]=x[j];
        x[j]=temp;
   }
return ;
}
```

P241

修改程序：

```
#include <stdio.h>
int main( void )
{void inv(int x[],int n);
int i, a[10]={3,7,9,11,0,6,7,5,4,2};
printf("The original array:\n");
for(i=0;i<10;i++)
    printf("%d ",a[ i ]);
printf("\n");
inv(a,10);
printf("The array has been vnverted:\n");
for(i=0;i<10;i++)
    printf("%d ",a[ i ]);
printf("\n");
return 0;
}

void inv(int *x,int n)
{int *p,temp,*i,*j,m=(n-1)/2;
i=x;j=x+n-1;p=x+m;
for(;i<=p;i++,j--)
    {temp=*i;*i=*j;*j=temp;}
return ;
}
```

评：修改之后的代码更 dirty 更令人作呕

P241~242

如果有一个实参数组，要想在函数中改变此数组中的元素的值，实参与形参的对应关系有以下 4 种情况。

（1）形参和实参都用数组名，

（2）实参用数组名，形参指针变量。

（3）实参和形参都用指针变量

```
int main()
{int a[10],*p=a;

f(p,10);
}
void f(int *x,int n)
{
}
```

（4）实参为指针变量，形参为数组名

评：一堆废话，说不到点子上，只会把读者绕晕

实际上根本不存在形参数组

数组名作为实参本来就表示指针

p=a 属于脱裤子放屁

P242~243

例 8.9

```
#include <stdio.h>
int main( void )
{void inv(int *x,int n);
int i, a[10],*p=arr;
printf("The original array:\n");
for(i=0;i<10;i++,p++)
    scanf("%d ",p);
printf("\n");
p=arr;
inv(p,10);
printf("The array has been vnverted:\n");
for(p=arr;p<arr+10;p++)
    printf("%d ",*p);
printf("\n");
return 0;
}
void inv(int *x,int n)
{int *p,m,temp,*i,*j;
  m=(n-1)/2;
  i=x;j=x+n-1;p=x+m;
for(;i<=p;i++,j--)
    {temp=*i;*i=*j;*j=temp;}
return ;
}
```

评：一蟹不如一蟹

P243

```
#include <stdio.h>
int main( void )
{void inv(int *x,int n);
int i, *arr;
printf("The original array:\n");
```

```
for(i=0;i<10;i++)
    scanf("%d ",arr+i);
printf("\n");
inv(arr,10);
printf("The array has been vnverted:\n");
for(i=0;i<10;i++)
    printf("%d ",*(arr+i));
printf("\n");
return 0;
}
```

评：信口开河的弥天大谎
后面的解释更是胡说八道

```
f(x[],int n )
```

评：函数定义都写不全
老谭的基本功很成问题

P244

注意：如果指针变量作实参，必须先使指针变量有确定值，指向一个已定义的对象。
评：纯粹是废话
任何东西做实参都必须有确定的值，和实参是不是“指针变量”没有半毛钱狗屁关系

“已定义的对象”也属于似是而非、狗屁不通的说法

例 8.10 用指针方法对 10 个整数由大到小顺序排序。
评：“指针方法”是子虚乌有的

解题思路：在主函数中定义数组 a 存放 10 个整数，定义 int *型指针变量 p 指向 a[0],定义函数 sort 使数组 a 中的元素由大到小的顺序排列。在主函数中调用 sort 函数，用指针变量 p 作实参。sort 函数的形参用数组名。
评：这是脱裤子放屁，因为 a 本身就是指向 a[0]的指针
难道求 3.0 的平方根不直接写 sqrt(3.0)非要先
double d=3.0;
再 sqrt(d) 不成

```
#include <stdio.h>
int main( void )
{void sort(int x[],int n);
   int i, *p,a[10];
   p=a;
   printf("please enter 10 integer numbers:");
   for(i=0;i<10;i++)
```

```
        scanf("%d",p++);
    p=a;
    sort(p,10);
    for(p=a,i=0;i<10;i++)
         {printf("%d ",*p);
            p++;
         }
printf("\n");
return 0;
}

void sort(int x[],int n)
{int i,j,k,t;
    for(i=0;i<n-1;i++)
        {k=i;
          for(j=i+1;j<n;j++)
               if(x[j]>x[k])k=j;
                     if(k!=i)
               {t=x[i];x[i]=x[k];x[k]=t;}
        }
}
```

评：其蠢无比的代码

首先

```
            for(i=0;i<10;i++)
                scanf("%d",p++);
```

————谭浩强 ，《C 程序设计》（第四版），清华大学出版社， 2010 年 6 月，p244

本可以简单地写成

```
        for(;p<a+10;p++)
                scanf("%d",p);
```

其次

```
    p=a;
    sort(p,10);
```

————谭浩强 ，《C 程序设计》（第四版），清华大学出版社， 2010 年 6 月，p244

更蠢

本可以简单地

```
sort(a,10);

for(p=a,i=0;i<10;i++)
              { printf("%d ",*p);
                  p++;
```

}

————谭浩强 ，《C 程序设计》（第四版），清华大学出版社， 2010 年 6 月，p244

同样很蠢

表达的无非是

```
for(p=a;p<a+10;p++)
    printf("%d ",*p);
```

而已

```
void sort(int x[],int n)
{int i,j,k,t;
for(i=0;i<n-1;i++)
    {k=i;
      for(j=i+1;j<n;j++)
          if(x[j]>x[k])k=j;
              if(k!=i)
                  {t=x[i];x[i]=x[k];x[k]=t;}
    }
}
```

评：最傻的就是 if(k!=i) 的那个一本正经的缩进，滑稽

其他风格方面的残疾就不多说了

P245

```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
```

……

a 代表二维数组首元素的地址

评：这种说法是片面的

a 本身就是数组

但在有些情况下

a 是作为指针（而不是地址）来使用

a 代表的是首行(即序号为 0 的行)的首地址

评：首地址 这个概念本身就有问题，是含混不清的

P246

C 语言又规定了数组名代表数组首元素的地址

评：混淆指针和地址的概念，误导

C 语言并没有规定了数组名代表数组首元素的地址

&[0][1],&[0][2],&[0][3]

评：竟然有这种低级错误
难以想象这种错误是怎么犯的

有必要对 a[i]的性质作进一步说明。a[i]从形式上看是 a 数组中序号为 i 的元素。如果 a 是一维数组名，则 a[i]代表 a 数组序号为 i 的存储单元。a[i]是有物理地址的，是占存储单元的。但如果 a 是二维数组，则 a[i]是一维数组名，它只是一个地址，并不代表某一元素的值（如同一维数组名只是一个指针常量一样）。a,a+i,a[i],*(a+i),*(a+i)+j,a[i]+j 都是地址。而*（a[i]+j）和*(*(a+i)+j)是二维数组元素 a[i][j]的值，……
评：这段陈述错误太多了
几乎就没有正确的地方

a[i]从形式上看是 a 数组中序号为 i 的元素。

它就是 a 数组中序号为 i 的元素

如果 a 是一维数组名，则 a[i]代表 a 数组序号为 i 的存储单元。a[i]是有物理地址的，是占存储单元的。

a 是多维数组名时，a[i]同样代表 a 数组序号为 i 的存储单元
“物理地址”是莫名奇妙的概念

如果 a 是二维数组，则 a[i]是一维数组名，它只是一个地址，并不代表某一元素的值

“只是一个地址”是大错特错的说法
“不代表某一元素的值”与老谭前面讲的“首元素”是自相矛盾的
a[i] 同样可以代表某一元素的值，当然这要看怎么理解“元素”这个概念

a,a+i,a[i],*(a+i),*(a+i)+j,a[i]+j 都是地址。

片面
而*（a[i]+j）和*(*(a+i)+j)是二维数组元素 a[i][j]的值

未必

P247

表 8.2 二维数组 a 的有关指针
评：基本上全是错误的
错误在于片面地解释数组名
并把指针当作地址
全然忽视了数据类型

班长相当于列指针，排长相当于行指针
评：C 语言中压根就没有行指针、列指针这样的概念
行指针、列指针是没有定义的不严谨的概念

而不严谨对于程序设计来说
就是灾难

P248

a 和 a[0]的值虽然相同（等于 2000），但是由于指针的类型不同（相当于排长和班长面向的对象一样，a 指向一维数组 a[0],而[0]指向列元素 a[0][0]）。
评：“a 和 a[0]的值虽然相同（等于 2000），但是由于指针的类型不同。”
这根本就不是话，小学语文老师要打屁股的

“相当于排长和班长面向的对象一样”
更是莫名其妙前后自相矛盾

“[0]指向列元素 a[0][0]”
喝高了吧
什么是"列元素"
[0]是神马东西

再次强调：二维数组名（如 a）是指向行的。因此 a+1 中的“1”代表一行中全部元素所占的字节数
评：指向数据对象的指针+1 有统一、确定的定义
因此这里根本没有“因此”可言

一维数组名（如 a[0],a[1]）是指向列元素的。a[0]+1 中的 1 代表一个 a 元素所占的字节数。
评：“列元素”是个扯淡的概念
什么叫“列元素”？
一列元素吗？
对不起，压根没有指向这种东西的指针
如果“列元素”指的只是数组中某一个具体的元素
有什么必要再发明一个子虚乌有、多余且压根没有准确定义的“列元素”的概念呢

例如，a 和 a+1 是指向行的指针，在它们前面加一个*就是*a 和*(a+1)，它们就成为列的指针，分别指向 a 数组 0 行 0 列的元素和 1 行 0 列的元素。
评：这是胡扯
根本就不存在指向列的指针
因而*a 和*(a+1)不可能是什么列的指针
甚至*a 和*(a+1)是否是指针还是未定之数

反之，在指向列的指针前面加&，就成为指向行的指针。例如 a[0]是指向 0 行 0 列元素的指针，在它前面加一个&，得&a[0],由于 a[0]与*(a+0)等价，因此&a[0]与&*a 等价，也就是与 a 等价，它指向二维数组的 0 行。
评：这个更荒诞
试想
int a[3][4];

int *p = a[0];
p 算不算指向“列的指针”呢？
如果算的话
你就是把刘谦招来
他也无法把&p 变成指向行的指针
再比如
如果把“a[0]　”　说成“是指向 0 行 0 列元素的指针”
那么 a[0]+1 显然是指向 0 行 1 列元素的指针
你若敢在代码中斗胆写出 &(a[0]+1)
连编译都根本通不过

不要把&a[i]简单地理解为 a[i]元素的物理地址，因为并不存在 a[i]这样一个实际的数据存储单元
评：这里，“物理地址”是故弄玄虚，是不懂装懂地唬人
“并不存在 a[i]这样一个实际的数据存储单元”则是自相矛盾
因为前面有“a 数组包含 3 行，即 3 个元素：a[0],a[1],a[2]。”（p245）这样的叙述

不要把&a[i]简单地理解为 a[i]元素的物理地址，因为并不存在 a[i]这样一个实际的数据存储单元。它只是一种地址的计算方法，能得到第 i 行的首地址
评：在 C 语言中并不存在地址计算
a[i] 得到的也并非第 i 行的首地址
而且首地址这个概念本身就是错误的

&a[i]或 a+i 指向行，而 a[i]或*(a+i)指向列
评：扯淡
什么叫“指向列”
这是根本不可能的

而对二维数组，a+i 不是指向具体存储单元而指向行。
评：内存中压根就没有“行”这种东西
指针只可能指向存储单元（除非 void *）

例 8.11
int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
printf("%d,%d\n",a,*a);
评：%d 转换格式用于转换 int 类型数据而不是转换指针

P249

例 8.12 有一个 3*4 的二维数组，要求用指向元素的指针变量输出二维数组各元素的值。
```
#include <stdio.h>
int main()
{int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int *p;
```

```
    for(p=a[0];p<a[0]+12;p++)
        {if((p-a[0])%4==0)printf("\n");
          printf("%4d",*p);
    }
    printf("\n");
    return 0;
  }
```

评：12：Magic Number

if((p-a[0])%4==0)printf("\n");:喧宾夺主。而且没有完成功能“输出 4 个数据后换行”(p250)，不得不在后面补了一个 printf("\n"); 。并且它在输出数据之前多输出了一个换行

printf("%4d",*p);：潜在的 BUG，之所以没暴露出来是因为数组元素的值太小而已

P251~252

```
#include <stdio.h>
int main()
{ int a[4]={1,3,5,7};
    int (*p)[4];
    p=&a;     //p 指向一维数组
  printf("%d\n",(*p)[3]);
    return 0;
}
```

注意第 5 行不应写成“p=a;”，因为这样写表示 p 的值是&a[0],指向 a[0]。

评：第 5 行确实不应写成“p=a;”

但是这样写并不表示 p 指向 a[0]

事实上尽管写 p=a;有些瑕疵，但这样 p 依然是指向数组的

p=&a;   //p 指向一维数组

这本身没有什么错误

但是和老谭在 p231 页所讲的

数组名不代表整个数组，只代表数组首元素的地址。

完全是自相矛盾的

P252

由于 p 是指向一维数组的指针变量，因此 p 加 1，就指向下一行

评：正确的说法是

由于 p 是指向一维数组的指针,因此 p 加 1，就指向下一个一维数组

它已经转化为指向列元素的指针了。

评：估计只有 SB 才能领会列元素这种 SB 概念

一维数组名可以作为函数参数，多维数组名也可以作为函数参数。用指针变量作形参，以接受实参数组名传递来的地址。可以有两种方法：①用指向变量的指针变量；②用指向一维数组的指针变量。

评：逻辑错乱

首先：“一维数组名可以作为函数参数，多维数组名也可以作为函数参数”这个说法是错误的，数组名只能做实参。形参不可能是数组名，即使用[]修饰，那也是对指针的一种变相说明

其次：“用指针变量作形参，以接受实参数组名传递来的地址”以接受实参数组名传递来的指针

第三：形参是何种类型只能依据实参的类型，不可能有两种方法

第四：前面说“多维数组名也可以作为函数参数”，后面却说“用指向一维数组的指针变量”，显而易见是错的

P252~253

例 8.14 有一个班，3 个学生，各学 4 门课，计算总平均分数以及第 n 个学生的成绩。

```
#include <stdio.h>
int main( )
{void average(float *p,int n);
   void search(float (*p)[4],int n);
   float score[3][4]={{65,67,70,60},{80,87,90,81},{90,99,100,98}};
   average( * score,12);
   search(score,2);
   return 0;
}

void average(float *p,int n)
   {float *p_end;
     float sum=0,aver;
     p_end=p+n-1;
     for(;p<=p_end;p++)
        sum=sum+(*p);
     aver=sum/n;
     printf("average=%5.2f\n",aver);
   }

void search(float (*p)[4],int n)
{int i;
   printf("The score of No.%d are:\n",n);
   for(i=0;i<4;i++)
      printf("%5.2f ",*(*(p+n)+i));
   printf("\n");
}
```

评：先，问题和代码根本对不上号

问题是“第 n 个学生的成绩”
但代码没有反应出这一点
毛病出在“问题”上
另外“计算总平均分数以及第 n 个学生的成绩”在语文上是不通的
第 n 个学生的成绩根本不是计算出来的
“有一个班，3 个学生，各学 4 门课”根本就不能算是问题的条件
相当于什么都没说
这种似是而非的问题对学习程序设计危害极大
正确地提出问题比解决问题重要得多

void average()这个函数的功能根本就不是“计算总平均分数”，而是计算之后输出。
把多个功能揉在一个函数里，不符合编程之道

average(* score,12);
这个调用写得也不好
那个 12 显得莫名其妙
不如写 average(* score, sizeof score / sizeof **score);
甚至不如写
average(* score,3*4);

用函数 search 找出并输出第 i 个学生的成绩
评：在解题思路中
题目本身不明确的要求被篡改为
这种胡乱篡改需求（尽管需求本身也不明确且有错误）的做法是最要不得的
这给学习者带来的坏影响比把 C 讲错了还严重
况且，这里的 i 也是莫名其妙不知所云的东东

谭的代码一向存在在 main()中写不该写代码的毛病
但这次走向了另一个极端
把该在 main()中写的内容写到了其他函数中

void search(float (*p)[4],int n)
中的 printf("The score of No.%d are:\n",n);
以及
void average(float *p,int n)
中的 printf("average=%5.2f\n",aver);
都是这样

此外 seach()的功能也根本不是在“查找”什么，它只是输出了一个一维数组的各个元素的值而已

P254

注意：实参与形参如果是指针类型，应当注意它们的类型必须一致。
评：这是废话
实参与形参本来就应该类型一致
怎么能扯出个“如果”来呢

不应把 int *型的指针（即元素的地址）传给 int (*)[4]型（指向一维数组）的指针变量，反之亦然。正如不应把“班长”传给“排长”一样，应当是“门当户对”
评：这是什么不伦不类的比喻啊
令人无语

search(score,2); //用 score（即 score[0]的起始地址作为实参）
评：“score[0]的起始地址作为实参”这种说法相当于把 score 贬低成了 void *
约等于什么都没说
因为 score ，score[0]，score[0][0]的起始地址都是一个

search(*score,2); //用*score（即 core[0]的地址作为实参
评：确实不对
但老谭说的理由也不对

虽然 score 和*score 都是地址，但后者的类型与形参 p 的类型不匹配。
评：胡扯
地址哪来的类型可言
C 语言中压根就没有地址这种类型

例 8.15 在上题的基础上，查找有一门以上课程不及格的学生，输出他们全部课程的成绩。
评：上个题本身就有很多毛病（参见 2538 楼）
居然还要在“在上题的基础上”

这里"一门以上"就是混沌不清的说法

```
#include <stdio.h>
int main()
{void search(float (*p)[4],int n);
float score[3][4]={{65,57,70,60},{58,87,90,81},{90,99,100,98}};

search(score,3);
return 0;
}

void search(float (*p)[4],int n)
```

```
{int i,j,flag;
for(j=0;j<n;j++)
      {flag=0;
       for(i=0;i<4;i++)
          if(*(*(p+j)+i)<60)flag=1;
             if(flag==1)
                    {printf("No.%d fails,his scores are:\n",j+1);
                    for(i=0;i<4;i++)
                           printf("%5.2f ",*(*(p+j)+i));
                           printf("\n");
                  }
       }
}
```

评：主要毛病

1.flag 用得很垃圾

2.main()里什么也没有，把所有的事情都推给了 search()函数

与其这样还不如把所有的代码都在 main()中写

3.search()函数名不符实，功能混乱不清

这种写法

就像用把所有的东西都塞到柜子里去的办法收拾房间一样

表面看 main()里很清爽

但骨子里依然是混乱不堪

P255

通过指针变量存取数组元素速度快，程序简明。用指针变量作形参，所处理的数组大小可以改变。因此数组与指针常常是紧密联系的，使用熟练的话可以使程序质量提高，编写程序灵活方便。

评：“通过指针变量存取数组元素速度快，程序简明。”:说速度快没有根据，未必程序简明。例 8.15 是直接的例子，速度不快而且不简明

“用指针变量作形参，所处理的数组大小可以改变。”：这句更荒唐，C 语言里没有什么东西在存在滞后大小可以改变

“因此数组与指针常常是紧密联系的”：平白无故地哪儿冒出个“因此”呢？从上文看不出任何“因”。逻辑混乱

在 C 程序中，字符数组是存放在字符数组中的

评：这个是无稽之谈

试问"ABC" 存在在哪个数组中？数组的名字是什么？

想引用一个字符串，可以用以下两种方法。

（1）用字符数组存放一个字符串，[quote]可以通过数组名和下标引用字符串中的一个字符，也可以通过数组名和格式声明"%s"输出该字符串。

评：文不对题，莫名其妙

用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量。

————谭浩强 ，《C 程序设计》（第四版），清华大学出版社， 2010 年 6 月，p256
狭隘
字符指针变量指向字符串的说法也不够准确

P256

C 语言对字符串常量是按字符数组处理的，在内存中开辟了一个字符数组用来存放该字符串常量，但是这个字符数组是没有名字的，因此不能通过数组名来引用，只能通过指针变量来引用。
评：“只能”是错误的

char string[]="I love China";
printf("%s\n",string);
评：这种写法很笨拙且低效
实际上可以简单地
puts(string);

P257

printf("%s\n",string);
%s 是输出字符串时所用的格式符，在输出项中给出字符指针变量名 string，则系统会输出 string 所指向的字符串第 1 个字符，然后自动使 string 加 1，使之指向下一个字符……如此直到遇到字符串结束标志'\0'为止。
评：非常通俗易懂
然而却是错的
错误的东西就是再通俗易懂又有什么意义呢
“自动使 string 加 1，使之指向下一个字符”极其荒谬
此外，前面说%s 是格式声明，这里又说是格式符
自相矛盾
“系统”二字一向是老谭打马虎眼时的常用词

通过字符数组名或字符指针变量可以输出一个字符串。
评：不通过这两者也可以
谭的归纳没有说到点子上
因而是不全面的和肤浅的

P257~258

int a[10];
……

printf("%d\n",a);
是不行的，它输出的是数组首元素的地址。
评：确实不行
但 printf("%d\n",a);本身也是不对的，并非是“它输出的是数组首元素的地址”

P258

例 8.18 将字符串 a 复制为字符串 b，然后输出字符串 b。
评：莫名其妙
题就是错的
汉语“复制为”的含义之一是“复制成”
显然这是错的

这个问题的正确提法是把 a 指向的字符串复制到 b 指向的位置

字符串 a、字符串 b 这种说法的错误在于“a”和“b”都 不是字符串
在 C 语言中，除非 string literal 可以直接称呼
比如 字符串"ABC"
用指针称呼只能说某指针指向的字符串
或从某个位置开始的字符串
用数组称呼只能说某数组中存储的字符串

```
#include <stdio.h>
int main( void )
{char a[]="I am a student.",b[20];
int i;
for(i=0;*(a+i)!='\0';i++)
      *(b+i)=*(a+i);
*(b+i)='\0';
printf("string a is:%s\n",a);
printf("string b is:");
for(i=0;b[i]!='\0';i++)
    printf("%c",b[i]);
printf("\n");
return 0;
}
```

评：拷贝字符串，从没见过这么笨的法子
for(i=0;*(a+i)!='\0';i++)
(b+i)=(a+i);
*(b+i)='\0';
原因：选择错误的语句
这里应该用 while 语句

这个题目应该这样写

```
#include <stdio.h>
void str_copy( char * , char const * );

int main( void )
{
    char tgt[80];

    str_copy( tgt , "I am a student." );//假设不会越界

    puts(tgt);

    return 0;
}

void str_copy( char * to, char const *from )
{
    while( (*to++=*from++)!='\0' )
        ;
}
```

程序中 a 和 b 都定义为字符数组，今通过地址访问其数组元素。

评：错。是通过指针

P258~259

例 8.19 用指针变量来处理 8.18 问题。

解题思路：定义两个指针变量 p1 和 p2,分别指向字符数组 a 和 b。改变指针变量 p1 和 p2 的值，使它们循序指向数组中的各元素，进行对应元素的复制。

评：老谭根本就没搞清楚“指向”两个字的含义

前面说指向数组

后面又说指向元素

自相矛盾

P259

如果想把一个字符串从一个函数“传递”到另一个函数，可以用地址传递的办法，即用字符数组名作参数，也可以用字符指针变量作参数。在被调用的函数中可以改变字符串的内容，在主调函数中可以引用改变后的字符串。

评：“用字符数组名作参数”和“用字符指针变量作参数”没有任何本质区别

其本质都是传递一个指针（而不是“地址传递”）

至于“在被调用的函数中可以改变字符串的内容”，这是显而易见的事情，因为传递的是指针
而“在主调函数中可以引用改变后的字符串”则是废话一句

8.4.2 字符指针作函数参数

评：字符指针作为函数参数没有什么特殊性
之所以把这个作为小节标题
是因为前面根本没有讲清楚数组名的本质
也没有讲清楚“数组形式”的形参的本质

函数的形参和实参可以分别用字符数组名或字符指针变量。

评：即是废话又是错话
首先
实参和形参是两回事
形参：
在“形式上”可以是数组名，但这不是数组类型，而是不完整类型，本质上是一个指针，把形参写成数组名有一些缺点，而把形参理解为数组名只有坏处没有好处

实参：实参不仅仅限于可以用“字符数组名或字符指针变量”

P259~260

```
#include <stdio.h>
int main( void )
{void copy_string(char from[],char to[]);
char a[]="I am a teacher.";
char b[]="You are a student.";
printf("string a=%s\nstring b=%s\n",a,b);
printf("copy string a to string b:\n");
copy_string(a,b);
printf("string a=%s\nstring b=%s\n",a,b);
return 0;
}
void copy_string(char from[],char to[])
{int i=0;
   while(from[i]!='\0')
      {to[i]=from[i];i++;}
   to[i]='\0';
}
```

评：总算是用函数了
不过那个 i 明显是多余了
拷贝的实现也过于笨拙

但是 260 页的运行结果明显是伪造的（中间多一个空行）

根据代码无法得到这样的结果

P260~261

（2）用字符型指针变量作实参

```
#include <stdio.h>
int main( void )
{void copy_string(char from[],char to[]);
char a[]="I am a teacher.";
char b[]="You are a student.";
char *from=a,*to=b;
printf("string a=%s\nstring b=%s\n",a,b);
printf("copy string a to string b:\n");
copy_string(a,b);
printf("string a=%s\nstring b=%s\n",a,b);
return 0;
}
void copy_string(char from[],char to[])
{int i=0;
   while(from[i]!='\0')
      {to[i]=from[i];i++;}
   to[i]='\0';
}
```

评：一蟹不如一蟹

char *from=a,*to=b;　无比荒唐

把 a,b 的值存放在变量里毫无必要

P261

```
#include <stdio.h>
int main( void )
{void copy_string(char *from,char *to);
char *a="I am a teacher.";
      char b[]="You are a student.";
char *p=b;
printf("string a=%s\nstring b=%s\n",a,b);
printf("copy string a to string b:\n");
copy_string(a,b);
printf("string a=%s\nstring b=%s\n",a,b);
return 0;
}
```

```
void copy_string(char *from,char *to)
{
    for(;*from!='\0';from++,to++)
        {*to=*from;}
    *to='\0';
}
```

评：一样的东西反复写
有什么意义！
而且风格怪异
{*to=*from;}
{to[i]=from[i];i++;}
{to[i]=from[i];i++;}

都属于垃圾写法

P262

程序改进：

```
void copy_string(char *from,char *to)
        {while((*to=*from)!='\0')
          {to++;from++;}
        }
```

评：改进有限
{to++;from++;}依然怪异

```
{while((*to++=*from++)!='\0');}
```

评：这个马马虎虎
但是把“;”写在行尾是一种糟糕的风格

```
{while(*from!='\0')
          *to++=*from++;
    *to='\0';
}
```

评：没必要展示这些垃圾写法

```
{while(*from)
          *to++=*from++;
    *to='\0';
}
```

评：无语

P263

（6）函数体中也可以改为只用一个 for 语句：

```
for(;(*to++=*from++)!=0 ; ) ; );
```

评：连 for 语句都能写错？

```
void copy_string(char from[],char to[])
{ char *p1,*p2;
   p1=from;p2=to;
   while((*p2++=*p1++)!='\0');
}
```

以上各种用法，变化多端，使用十分灵活，程序精炼，比较专业，初学者看起来不太习惯

评：应该说是非常业余，专业人士看不惯

尤其是那个 p1,p2

画蛇添足

表 8.3 调用函数时实参与形参的对应关系

实参	形参	实参	形参
字符数组名	字符数组名	字符指针变量	字符指针变量
字符数组名	字符指针变量	字符指针变量	字符数组名

评：这个表毫无意义

只能给学习者以误导

实际上

形参只可能是“字符指针变量”

而实参则绝不限于“字符数组名”和“字符指针变量”

8.4.3 使用字符指针和字符数组的比较

评：所讲的并非是字符指针的特殊性而是指针的共性

所讲的字符数组的性质也是所有数组的共性（除了赋初值，字符数组有一点特殊）

把指针和数组的共性作为特殊的字符指针和字符数组的性质来讲解

只能说是在误导

P264

编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配一个存储单元

评：存储单元这种概念

在老谭那里不知道有多少种含义

在这里

同一句话中出现了两次

但含义完全不同

这是小学生都能看出的逻辑错误

P265

指针变量的值是可以改变的，而数组名代表一个固定的值（数组首元素的地址），不能改变。
评：片面

指针变量的值未必是可以改变的
数组名也未必代表一个固定的值

字符数组中各元素的值是可以改变的（可以对它们再赋值）
评：荒谬
const char a[]="HOUSE";
改改这个试试？

(7)引用数组元素。对字符数组可以用下标法(用数组名和下标)引用一个数组元素（如 a[5]），也可以用地址法(如*(a+5))引用数组元素。

评：毫无意义的废话

这个前面都讲过了

有什么必要再借字符数组和字符指针重讲一次呢

所谓“下标法”、“地址法”是可笑的说法

表明根本还不懂得什么是指针

如果定义了字符指针变量 p，并使它指向数组 a 的首元素，则可以用指针变量带下标的形式引用数组元素（如 p[5]），同样，可以用地址法（如*（p+5））引用数组元素 a[5]。

评：更是废话

而且前提条件“如果定义了字符指针变量 p，并使它指向数组 a 的首元素”是完全多余的

但是，如果指针变量没有指向数组，则无法用 p[5]或*(p+5)这样的形式引用数组中的元素。

评：简直让人喷饭

还有比这更费的话吗

不过也难说

恐怕没有最费

只有更费

若字符指针变量 p 指向字符串常量，就可以用指针变量带下标的形式引用所指的字符串中的字符。

评：误导

只要是指针就可以进行[]运算

根本不需要“若字符指针变量 p 指向字符串常量”这样的前提条件

(8)用指针变量指向一个格式字符串，可以用它代替 printf 函数中的格式字符串。

评：这个一点都不值得惊奇

实参与形参类型一致而已

P266

例如：

```
char *format;
format="a=%d,b=%f\n";
printf(format,a,b);
```

……这种 printf 函数称为可变格式输出函数。

评：少见多怪不要紧

但不要随意捏造什么“可变格式输出函数”

这种捏造对于学习者来说是故部迷阵，没有任何意义

printf()函数只有一种，根本不存在什么“可变格式输出函数”

也可以用字符数组实现。

评：离题万里的废话

8.5 指向函数的指针

P267

```
c=max(a,b);
printf("a=%d\nb=%d\nmax=%d\n",a,b,c);
```

评：不把值赋给变量就不放心
　　实际上 c 这个变量毫无必要

```
int max(int x,int y)
{int z;
   if(x>y) z=x;
   else z=y;
   return z;
}
```

评：不就是

```
int max(int x,int y)
{
   return (x>y)?x:y;
}
```

吗？

P268

和数组名代表数组首元素地址类似，函数名代表该函数的入口地址。
评：片面。
正如数组名并不总是代表指向起始元素的指针一样，函数名在有些情况并不代表函数的入口地址

定义指向函数的指针变量的一般形式为：
类型名 (*指针变量名)(函数参数列表);
评：那不叫“函数参数列表”
　　是参数类型列表

P269

用函数指针变量调用函数时，只须将(*p)代替函数名即可(p 为指针变量名)，在(*p)之后的括号中根据需要写上实参。例如：

c=(*p)(a,b);

评：这只是使用指向函数指针调用函数的方法之一
　　而且是一种比较笨拙的方式

int max(int,int);//函数声明
int min(int x,int y);//函数声明

评：风格不统一

P268~269

```
#include <stdio.h>
int main()
{int max(int,int);
int min(int x,int y);
int (*p)(int,int);
int a,b,c,n;
printf("please enter a and b:");
scanf("%d,%d",&a,&b);
printf("please choose 1 or 2:");
scanf("%d",&n);
if(n==1)p=max;
else if(n==2)p=min;
c=(*p)(a,b);
printf("a=%d,b=%d\n",a,b);
if(n==1)printf("max=%d\n",c);
else printf("min=%d\n",c);
return 0;
}

int max(int x,int y)
{int z;
if(x>y) z=x;
else       z=y;
return(z);
}

int min(int x,int y)
{int z;
```

```
    if(x<y) z=x;
    else    z=y;
    return(z);
}
```

评：首尾不一
假设用户一定会正确输入的话
那么 else if(n==2)p=min;中的 if(n==2) 就是多余的
假设用户输入可能会有错误
那么这个代码的安全性方面的问题就太严重了

P269

这个例子是比较简单的，只是示意性的，但它很有实用价值。
评：简单、示意性是对的
说有实用价值是不顾事实

在许多应用程序中常用菜单提示输入一个数字，然后根据输入的不同值调用不同的函数，实现不同的功能，就可以用此方法。
评：这是一相情愿的想当然
菜单调用的函数的类型多数不同
这种方法根本用不上

当然，也可以不用指针变量，而用 if 语句或 switch 语句进行判断，调用不同的函数。但是显然用指针变量更简洁和专业。
评：这就是在胡扯了
这个例题说是简单的示意性的代码还差强人意
但就这个题目而言
它显然不是很恰当地使用指向函数的指针
至于“更简洁和专业”更是无中生有的说法
这个代码一点也不简洁，一点也不专业
这个题目完全可以像下面这样写

```
#include <stdio.h>

int max(int,int);
int min(int,int);

int main( void )
{
int a,b,n;

printf("输入两个整数:");
scanf("%d%d",&a,&b);
```

```
printf("选择 1 或 2:");
scanf("%d",&n);

if(n==1)
    printf("max=%d\n",max(a,b));

if(n==2)
    printf("max=%d\n",max(a,b));

return 0;
}

int max(int x,int y)
{
return (x>y)?x:y;
}

int min(int x,int y)
{
return (x<y)?x:y;
}
```

相比之下，少了 4 个变量，代码更少。而且根本不存在谭代码中存在的严重的安全性隐患问题
这才叫简洁和专业

P270

指向函数的指针变量的一个重要用途是把函数的地址作为参数传递到其他函数。
评：隔靴搔痒式的议论
另外这里的“指向函数的指针变量”中的“变量”这个限定也是错误的

P271~272

```
#include <stdio.h>
int main()
{void fun(int x,int y,int (*p)(int,int));
int max(int ,int);
int min(int ,int);
int add(int ,int);
int a=34,b=-21,n;
printf("please choose 1,2 or 3:");
scanf("%d",&n);
if(n==1) fun(a,b,max);
```

```
else    if(n==2) fun(a,b,min);
else    if(n==3) fun(a,b,add);
return 0;
}

int fun(int x,int y,int (*p)(int,int))
{int result;
result=(*p)(x,y);
printf("%d\n",result);
}

int max(int x,int y)
{int z;
if(x>y) z=x;
else       z=y;
printf("max=");
return(z);
}

int min(int x,int y)
{int z;
if(x<y) z=x;
else       z=y;
printf("min=");
return(z);
}

int add(int x,int y)
{int z;
z=x+y;
printf("sum=");
return(z);
}
```

评：这个代码根本不可能通过编译

可蹊跷的是

在 272 页~273 页

老谭居然一本正经地给出了运行结果

```
please choose 1,2 or 3:1
max=34

please choose 1,2 or 3:2
min=-21

please choose 1,2 or 3:3
```

sum=13

这算不算伪造啊？
算不算欺骗读者啊

8.6 返回指针值的函数

P274

```
#include <stdio.h>
int main()
{float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
float *search(float (*pointer)[4],int n);
float *p;
int i,k;
printf("enter the number of student:");
scanf("%d",&k);
printf("The scores of No.%d are:\n",k);
p=search(score,k);
for(i=0;i<4;i++)
    printf("%5.2f\t",*(p+i));
printf("\n");
return 0;
}

float *search(float (*pointer)[4],int n)
{float *pt;
pt=*(pointer+n);
return(pt);
}
```

。

评：传说中的为赋新词强说愁
search()函数矫揉造作且毫无意义

下面的写法要好的多

```
#include <stdio.h>
int main()
{
float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
int i,k;
printf("enter the number of student:");
scanf("%d",&k);
```

```
    printf("The scores of No.%d are:\n",k);
    for(i=0;i<4;i++)
        printf("%5.2f\t",score[k][i]);
    putchar('\n');
    return 0;
    }
```

P275

对 pointer+1 加了"*"号后，指针从行控制转化为列控制了
评：“对 pointer+1 加了"*"号”，应该是对 pointer+1 进行了*运算

行控制，列控制：莫名其妙，不知所云

P276

```
#include <stdio.h>
int main()
{float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
float *search(float (*pointer)[4]);
float *p;
int i,j;
for(i=0;i<3;i++)
    {p=search(score+i);
     if(p==*(score+i))
        {printf("No.%d score:",i);
         for(j=0;j<4;j++)
            printf("%5.2f ",*(p+j));
         printf("\n");
        }
    }
return 0;
}

float *search(float (*pointer)[4])
{int i=0;
float *pt;
pt=NULL;
for(;i<4;i++)
    if(*(*pointer+i)<60)pt=*pointer;
return(pt);
}
```

评：这段代码毛病更多

```
float *search(float (*pointer)[4])
{int i=0;
float *pt;
pt=NULL;
for(;i<4;i++)
    if(*(*pointer+i)<60)pt=*pointer;
return(pt);
}
```

简直是傻到家了的写法
应该在循环语句内直接写 return 语句
而且 pt 是完全多余的
search()函数的返回值类型也设计得极不合理，根本没必要返回指针

main()中不应该写输出的代码，应该用函数完成
p=search(score+i); 很傻，search()这个函数应该回答的问题是 score+i 所指向的数据中是否有小于 60 的，根本没有必要返回指针，因为这个指针在 main()中是清清楚楚的，根本不用求

P277

什么情况下要用到指针数组呢？指针数组比较适合用来指向若干个字符串，使字符串处理更加方便灵活。
评：前面介绍指针数组是以 int *p[4]; 为例
这里又说“指针数组比较适合用来指向若干个字符串”
显然自相矛盾

指针数组并非“适合用来指向若干个字符串”（这里老谭显然混淆了 string 与 string literal）
况且指针数组指向若干字符串这话也根本说不通
数组不是指针，如果把数组作为指针的话它指向的只是其起始元素，指针不可能指向多个对象

可以分别定义一些字符串
评：变量和函数可以定义
字符串怎么定义

P278～279

```
#include <stdio.h>
#include <string.h>
int main()
{void sort(char *name[],int n);
```

```
void print(char *name[],int n);
char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer design"};

int n=5;
sort(name,n);
print(name,n);
return 0;
}

void sort(char *name[],int n)
{char *temp;
int i,j,k;
for(i=0;i<n-1;i++)
      {k=i;
       for(j=i+1;j<n;j++)
            if(strcmp(name[k],name[j])>0)k=j;
       if(k!=i)
           {temp=name[i];name[i]=name[k];name[k]=temp;}
      }
}
void print(char *name[],int n)
{int i;
  for(i=0;i<n;i++)
      printf("%s\n",name[i]);
}
```

评：公平地说

这是谭书中写得比较好的代码

但是很可惜

main()中的

int n=5;

sort(name,n);

print(name,n);

太荒唐了

倒不如直接写

sort(name,5);

print(name,5);

无论如何这个 5 不可能来自一个不相干的变量 n 的值

规矩的做法是根据 name 计算得到

此外 sort()函数中的交换应用一个函数实现

print()函数中的

printf("%s\n",name[i]);

不如
puts(name[i]);

程序改进：
print 函数也可以写为以下形式：

```
void print(char *name[],int n)
{int i=0;
char *p;
p=name[0];
while(i<n)
    {p=*(name+i++);
      printf("%s\n",p);
    }
}
```

评：不改倒还好
这一改就露马脚了
首先 p 多余
p=name[0];更是说明作者思路不清，因为这是一句废话
这不是“改进”是“改退”

P280

```
char **p;
……
printf("%d\n",*p);
```

评：用%d 输出指针是错误的

P 282

如果在一个指针变量中存放一个目标变量的地址，这就是“单级间址

指向指针的指针用的是“二级间址”方法。

评：怎么总发明这些没用的新概念
“间址”这个说法非常可笑

P 282

但实际上在程序中很少有超过二级间址的。级数愈多，愈难理解，容易产生混乱，出错机会也多。

评：出错不是因为级数多
而是还不懂

P 283

在某些情况下，main 函数可以有参数

评：这叫什么话
什么叫“在某些情况下”
在哪些情况下啊

P 283

int main(int argc,char *argv[])
……argv……,它是一个* char 指针数组，数组中的每一个元素(其值为指针)指向命令行中的一个字符串。

评：数组中有一个元素并不指向命令行中的字符串

P 283

命令行可以写成以下形式：
file1 China Beijing
file1 为可执行文件名，……实际上，文件名应包括盘符、路径，今为简化起见，用 file1 来代表。

评：那东西能简化吗？
和实际情况根本不符

P 284

在 Visual C++环境下对 程序编译和连接后，……

评：程序不可移植？

P 284

如果用 UNIX 系统的命令行输入：

```
$ ./echo Computer and C Language
```

评：前面用 VC++
这里居然冒出个 UNIX 系统

难得的是老谭居然知道了前几版的 echo 的荒谬

P 285

第 7 章介绍过全局变量和局部变量，全局变量是分配在内存中的静态存储区的，非静态的局部变量（包括形参）是分配在内存中的动态存储区的，这个存储区是一个称为栈（stack）的区域

评：C 语言并没有全局变量这个概念
只有外部变量
如果把外部变量叫全局变量
那么 static 的外部变量的“全局”在哪里体现？

此外静态存储区、动态存储区、栈之类也都是没有依据的说法

P 285

C 语言还允许建立内存动态分配区域

评：有时陈述越荒唐，越难以反驳
C 语言根本就没有“内存动态分配区域”这样的概念，更谈不上“允许建立”

P 285

这些数据是临时存放在一个特别的自由存储区，称为堆(heap)区

评：对错暂且不谈
这些与 C 没有任何关系”
http://bbs.chinaunix.net/thread-3591547-1-1.html
这个帖子里对前面几个相关问题有详细的讨论
这里不再赘述

P 285

```
void *malloc(unsigned int size);
```

……形参 size 的类型定为无符号整型（不容许为负数）

评：形参 size 的类型不一定是无符号整型
此外没有说明 size 为 0 的情况

P 285

第 7 章介绍过全局变量和局部变量，全局变量是分配在内存中的静态存储区的，非静态的局部变量（包括形参）是分配在内存中的动态存储区的，这个存储区是一个称为栈（stack）的区域
评：C 语言并没有全局变量这个概念
只有外部变量
如果把外部变量叫全局变量
那么 static 的外部变量的“全局”在哪里体现？

此外静态存储区、动态存储区、栈之类也都是没有依据的说法

P 285

C 语言还允许建立内存动态分配区域
评：有时陈述越荒唐，越难以反驳
C 语言根本就没有“内存动态分配区域”这样的概念，更谈不上“允许建立”

P 285

这些数据是临时存放在一个特别的自由存储区，称为堆(heap)区
评：对错暂且不谈
这些与 C 没有任何关系

P 285

void *malloc(unsigned int size);
……形参 size 的类型定为无符号整型（不容许为负数）
评：形参 size 的类型不一定是无符号整型
　　此外没有说明 size 为 0 的情况

P 286

void *calloc(unsigned n,unsigned　　size);

评：这种函数原型是几十年前的
早就过时了
老谭居然用这种过时的东西来冒充 C99
此外书中对该函数的功能描述也极不完整

P 286

void free(void *p);
……。p 应是最近一次调用 calloc 或 malloc 函数时得到的函数返回值。

评：简单的一句话
竟然有两个错误
而且都是原则性的错误

P 286

void *realloc(void *p, unsigned int size);
用 realloc 函数将 p 所指向的动态空间的大小改变为 size。p 的值不变。如果重分配不成功，返回 NULL。
如
realloc(p,50);//将 p 所指向的已分配的动态空间改为 50 字节

评：能够接二连三地在两页之内连续不断地信口开河
令人叹为观止

P 286

说明：以前的 C 版本提供的 malloc 和 calloc 函数得到的是指向字符型数据的指针，其原型为

```
char *malloc(unsigned int size);
```

评：必须补充说明的是
老谭的书提供的函数原型也是旧版本，最多是一种伪装成新版本的显得有点不伦不类的“半新半旧”版本

P 286

说明：以前的 C 版本提供的 malloc 和 calloc 函数得到的是指向字符型数据的指针，其原型为

```
char *malloc(unsigned int size);
```

评：必须补充说明的是
老谭的书提供的函数原型也是旧版本，最多是一种伪装成新版本的显得有点不伦不类的“半新半旧”版本

P 286

void *calloc(unsigned n,unsigned size);
评：这种函数原型是几十年前的
　　早就过时了
　　老谭居然用这种过时的东西来冒充 C99
　　此外书中对该函数的功能描述也极不完整

P 286

void free(void *p);
……。p 应是最近一次调用 calloc 或 malloc 函数时得到的函数返回值。
评：简单的一句话
　　竟然有两个错误
　　而且都是原则性的错误

P 286

void *realloc(void *p, unsigned int size);
用 realloc 函数将 p 所指向的动态空间的大小改变为 size。p 的值不变。如果重分配不成功，返回 NULL。
如
realloc(p,50);//将 p 所指向的已分配的动态空间改为 50 字节
评：能够接二连三地在两页之内连续不断地信口开河
令人叹为观止

P 286

说明：以前的 C 版本提供的 malloc 和 calloc 函数得到的是指向字符型数据的指针，其原型为
char *malloc(unsigned int size);
评：必须补充说明的是
老谭的书提供的函数原型也是旧版本，最多是一种伪装成新版本的显得有点不伦不类的“半新半旧”版本

P 287

```
pt=(int *)malloc(100);
```
……
要说明的是类型转换只是产生了一个临时的中间值赋给了 pt，但没有改变 malloc 函数本身的类型。

评：废话
malloc 函数返回值的类型怎么可能改变呢？
“一个临时的中间值”属于捏造
产生于根本不理解类型转换运算

P 287

C99 标准把以上 malloc,calloc,realloc 函数的基类型定为 void 类型，这种指针称为无类型指针(typeless pointer)，即不指向哪一种具体的类型数据，只表示用来指向一个抽象的类型的数据，即提供一个纯地址，而不能指向任何具体的对象。

评：

C99 标准把以上 malloc,calloc,realloc 函数的基类型定为 void 类型

这几个函数的类型和 C99 没有关系，它们并非 C99 的新内容
这是在为简介与封底所吹嘘的“按 C99 标准介绍”提供谎言背书
事实上这本书的很多内容异常陈旧，连 C89 都没达到

这种指针称为无类型指针(typeless pointer)

这是公然的捏造，欺骗对 C 一无所知的初学者
C 语言中任何表达式都有类型
根本就不存在无类型（typeless）这种概念

只表示用来指向一个抽象的类型的数据

这是用捏造的，子虚乌有的“抽象的类型”这个概念来胡扯

整个 8.8.1　　8.8.2 两小节，几乎处处是错，而且绝大多数都是硬伤

P 287

C99 允许使用基类型为 void 的指针类型

评：在 C89 中就已经是这样了
C99 在这一点上没有改变
老谭非要在此强调 C99 这三个字
究竟是何居心

P 287

例 8.30 建立动态数组，输入 5 个学生的成绩，另外用一个函数检查其中有无低于 60 分的，输出不合格的成绩。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{void check(int *);
int *pl,i;
```

```
pl=(int *)malloc(5*sizeof(int));
for(i=0;i<5;i++)
     scanf("%d",p1+i);
check(pl);
return 0;
}
void check(int *p)
{int i;
printf("They are fail:");
for(i=0;i<5;i++)
    if(p[ i]<60)printf("%d",p[ i]);
printf("\n");
}
```

评：嗯，初学者使用 malloc()函数最常见的两个毛病都占全了

P 287

```
pt=(int *)malloc(100);
```

……

要说明的是类型转换只是产生了一个临时的中间值赋给了 pt，但没有改变 malloc 函数本身的类型。

评：废话

malloc 函数返回值的类型怎么可能改变呢？

“一个临时的中间值”属于捏造

产生于根本不理解类型转换运算

P 287

C99 标准把以上 malloc,calloc,realloc 函数的基类型定为 void 类型,这种指针称为无类型指针(typeless pointer),即不指向哪一种具体的类型数据，只表示用来指向一个抽象的类型的数据，即提供一个纯地址，而不能指向任何具体的对象。

评：这几个函数的类型和 C99 没有关系，它们并非 C99 的新内容

这是在为简介与封底所吹嘘的“按 C99 标准介绍”提供谎言背书

事实上这本书的很多内容异常陈旧，连 C89 都没达到

这种指针称为无类型指针(typeless pointer)

这是公然的捏造，欺骗对 C 一无所知的初学者
C 语言中任何表达式都有类型
根本就不存在无类型（typeless）这种概念

只表示用来指向一个抽象的类型的数据

这是用捏造的，子虚乌有的“抽象的类型”这个概念来胡扯

整个 8.8.1　8.8.2 两小节，几乎处处是错，而且绝大多数都是硬伤

P 287

C99 允许使用基类型为 void 的指针类型
评：在 C89 中就已经是这样了
C99 在这一点上没有改变
老谭非要在此强调 C99 这三个字
究竟是何居心

P 287~288

例 8.30 建立动态数组，输入 5 个学生的成绩，另外用一个函数检查其中有无低于 60 分的，输出不合格的成绩。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{void check(int *);
int *pl,i;
pl=(int *)malloc(5*sizeof(int));
for(i=0;i<5;i++)
   scanf("%d",pl+i);
check(pl);
return 0;
}
void check(int *p)
{int i;
printf("They are fail:");
for(i=0;i<5;i++)
  if(p[ i]<60)printf("%d",p[ i]);
printf("\n");
}
```

评：嗯，初学者使用 malloc()函数最常见的两个毛病都占全了

P 289

指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替，例如，变量的指针就是变量的地址，指针变量就是地址变量。

评：大错特错
根据我的观察，绝大多数学不懂指针的人恰恰就是因为把指针当成了地址
另一些把指针当成地址的人，则往往是用一种扭曲的逻辑来理解指针

P 289

指针就是地址本身

评：不是去辨析两个概念之间的区别
而是武断地混淆它们
首先，这两个概念属于不同的范畴
指针是 C 语言层面的概念
而地址则是 C 语言之外的概念
地址用于描述内存（可能是实的也可能是虚的）
C 语言用指针的值来表示地址
但不能说指针就是地址
就像代码中可能用 1 表示问题中的一只猴子
不能说 C 代码中的 1 就是 1 只猴子一样
其次，指针表达式的值表示地址
但指针的含义通常不仅仅限于值这一方面
不能因为人有两只手
就说两只手就是人
第三
指针是 C 语言的一种数据类型
但 C 语言并没有地址这种数据类型

P 289

例如 2008 是某一变量的地址，2008 就是变量的指针。

评：2008 在 C 语言中只是一个 int 类型的字面量
绝对不可能是指针

P 289

有人认为指针是类型名，指针的值是地址

评：不是什么“有人认为”
而是 C 标准就是如此规定
老谭既然号称这本书是“按照 C99”写的
难道连 C99 都压根没有看过吗？

至于“指针的值是地址”，其含义是“指针类型表达式的值表示地址”
这有什么不对吗

P 289

类型是没有值的，只有变量才有值

评：“类型是没有值的”，是的，但所有表达式都有类型，也都有值
至于
“只有变量才有值”
这话说的太过分了吧
和“根据常识，偶数不是素数”有得一拼
希望谭先生自己改正

P 289

正确的说法是指针变量的值是一个地址

评：“这句话确实不能算错，但是和指针的值是地址的含义没有本质区别
对比起来
“指针变量的值是一个地址”是一种相当片面的说法
因为严格地说法应该是，指针类型表达式的值是一个地址，或曰指针的值是地址

P 289

不要杜撰出“地址的值”这样莫须有的名称。

评：老谭这是针对谁啊？
没看见有人杜撰“地址的值”这样的词啊
所以老谭的这个指责本身倒是显得非常“莫须有”
而且
谭先生自己恰恰一向是以杜撰莫须有名词为能事的
试举几例：“基类型”“中间变量”“行控制”“列控制”“列元素”“动态自由分配区”“动态空间”“无类型指针”……
而这些，都是子虚乌有的概念

P 289

什么叫“指向”，地址就意味着指向，因为通过地址能找到具有该地址的对象。

评：谭先生又错了
通过地址并不一定能找到具有该地址的对象
int i=6;
void *p=&i;
按照你所谓“指针变量的值是一个地址”的说法
你不否认 p 的值是一个地址吧
然而仅仅根据这个地址能找到 i 这个对象吗？
所以“地址就意味着指向”显然是错误的

P 289

对于指针变量来说，把谁的地址存放在指针变量中，就说指针变量指向谁

评：懒得多废话了
参见 LS
难道能说 p 指向 i 吗？
显然不能
p 谁也不指向

P 289

```
int a,*p;
float b;
p=&a;     //a 是 int 型，合法
p=&b;     //b 是 float 型，类型不匹配
```

评：居然用了“int 型”而不是“整型”，有进步，表扬
但最后一句也并非不合法

P 289

既然许多数据对象（如变量、数组、字符串和函数等）都在内存中被分配存储空间，就有了地址，也就有了指针。

评：这段话表明谭先生对数据对象这个词的确切含义不甚了了
其次“也就有了指针”的说法是错误的

P 289

可以定义一些指针变量，存放这些数据对象的地址，即指向这些对象

评：同上，谭先生还不懂得“对象”或“数据对象”这个词在 C 语言中的基本含义

P 289

指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替，例如，变量的指针就是变量的地址，指针变量就是地址变量。

评：大错特错
根据我的观察，绝大多数学不懂指针的人恰恰就是因为把指针当成了地址
另一些把指针当成地址的人，则往往是用一种扭曲的逻辑来理解指针

P 289

指针就是地址本身
评：不是去辨析两个概念之间的区别
而是武断地混淆它们

首先，这两个概念属于不同的范畴
指针是 C 语言层面的概念
而地址则是 C 语言之外的概念
地址用于描述内存（可能是实的也可能是虚的）
C 语言用指针的值来表示地址
但不能说指针就是地址
就像代码中可能用 1 表示问题中的一只猴子
不能说 C 代码中的 1 就是 1 只猴子一样
其次，指针表达式的值表示地址
但指针的含义通常不仅仅限于值这一方面
不能因为人有两只手
就说两只手就是人
第三
指针是 C 语言的一种数据类型
但 C 语言并没有地址这种数据类型
(暂时想到这么多，后面想到我再补充)

P 289

例如 2008 是某一变量的地址，2008 就是变量的指针。
评：2008 在 C 语言中只是一个 int 类型的字面量
绝对不可能是指针

P 289

有人认为指针是类型名，指针的值是地址
评：不是什么“有人认为”
而是 C 标准就是如此规定
老谭既然号称这本书是“按照 C99”写的
难道连 C99 都压根没有看过吗？

至于“指针的值是地址”，其含义是“指针类型表达式的值表示地址”
这有什么不对吗

P 289

类型是没有值的，只有变量才有值
评：“类型是没有值的”，是的，但所有表达式都有类型，也都有值
至于
“只有变量才有值”
这话说的太过分了吧
和“根据常识，偶数不是素数”有得一拼
希望谭先生自己改正

P 289

正确的说法是指针变量的值是一个地址
评：这句话确实不能算错，但是和指针的值是地址的含义没有本质区别
对比起来
“指针变量的值是一个地址”是一种相当片面的说法
因为严格地说法应该是，指针类型表达式的值是一个地址，或曰指针的值是地址

P 289

不要杜撰出“地址的值”这样莫须有的名称。
评：老谭这是针对谁啊？
没看见有人杜撰“地址的值”这样的词啊
所以老谭的这个指责本身倒是显得非常“莫须有”
而且
谭先生自己恰恰一向是以杜撰莫须有名词为能事的
试举几例：“基类型”“中间变量”“行控制”“列控制”“列元素”“动态自由分配区”“动态空间”“无类型指针”……
而这些，都是子虚乌有的概念

P 289

什么叫“指向”，地址就意味着指向，因为通过地址能找到具有该地址的对象。
评：谭先生又错了
通过地址并不一定能找到具有该地址的对象
int i=6;
void *p=&i;
按照你所谓“指针变量的值是一个地址”的说法
你不否认 p 的值是一个地址吧

然而仅仅根据这个地址能找到 i 这个对象吗？
所以“地址就意味着指向”显然是错误的

P 289

对于指针变量来说，把谁的地址存放在指针变量中，就说指针变量指向谁
评：懒得多废话了
参见 LS
难道能说 p 指向 i 吗？
显然不能
p 谁也不指向

P 289

只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。
评：写的书如果如同筛子一般处处都是漏洞的话
我觉得还是少使用“只有”这类只有懂得 C 语言的人才有资格使用的词汇
谭先生如果把自己书上的“只有”二字全部删除的话
错误至少会减少三分之一

根据我的经验
谭先生一说“只有”
十有八九是错的

P 289

```
int a,*p;
float b;
p=&a;     //a 是 int 型，合法
p=&b;     //b 是 float 型，类型不匹配
```

评：居然用了“int 型”而不是“整型”，有进步，表扬
但最后一句也并非不合法

P 289

既然许多数据对象（如变量、数组、字符串和函数等）都在内存中被分配存储空间，就有了地址，也就有了指针。
评：这段话表明谭先生对数据对象这个词的确切含义不甚了了

其次“也就有了指针”的说法是错误的

P 289

可以定义一些指针变量，存放这些数据对象的地址，即指向这些对象
评：同上，谭先生还不懂得“对象”或“数据对象”这个词在 C 语言中的基本含义

P 289

void *是一种特殊的指针，不指向任何类型的数据。
评：很对
但请谭先生注意在 287 页有一个自相矛盾的陈述
“只表示用来指向一个抽象的类型的数据”

所谓“抽象的类型的数据”是子虚乌有的说法

P 289

如果需要用此地址指向某类型的数据，应对地址进行类型转换
评：“此地址指向某类型的数据”是绝对不可能的
是进行类型转换后得到的指针指向数据

P 289

表 8.4 指针变量的类型及含义
int i; 定义整型变量 i
评：“整型”是国内 C 语言书最混乱不堪的概念
就谭这本书而言
有时它表示一个集合概念
在这里又表示 int 类型
国内对这种混乱熟视无睹
表明我们的教育在总体上来说缺乏起码的逻辑常识

P 289

int (*p)[4];p 为指向包含 4 个元素的一维数组的指针变量
评：“包含 4 个元素”应为“包含 4 个 int 类型元素”

P 289

int f();f 为返回整型函数值的函数
评：这个……
是否应该算是一种不完全类型？
请教 幻の上帝 或 OwnWaterloo

P 290

指针变量加(减)一个整数。
例如：p++, p--, p+i, p-i, p+=i, p-=i 等均是指针变量加(减)一个整数。
将该指针变量的原值（是一个地址）和它指向的变量所占用的存储单元的字节数相加（减）。
评：这些运算是有前提条件的
离开前提谈不上运算

P 290

指针变量赋值。
将一个变量地址赋给一个指针变量。例如
p=&a; （将变量 a 的地址赋给 p）
评：没人知道这里的 a 和 p 是什么
所以这里写的东西毫无意义

P 290

p=array; （将数组 array 的首地址赋给 p）
评：同上

P 290

p=&array[i]; （将数组 array 第 i 个元素的地址赋给 p）
评：这种写法很笨

p=max;
p1=p2;

脱离声明写这些毫无意义

P 290

两个指针变量可以相减。
如果两个指针变量都指向同一数组元素，则两个指针变量值之差是两个指针之间的元素个数
评：这个说法偷工减料，不完整
　　此外指针相减不限于指针变量

P 290

（5）指针运算
评：内容实际上是“指针变量”运算，而且严重不全。叙述不完整，不严密

P 290

指针变量可以有空值，即该指针变量不指向任何变量
评：指针变量不一定指向变量

P 290

在 stdio.h 头文件中对 NULL 进行了定义：
#define NULL 0
评：实际上正规一点的说法应该是在 stddef.h 中
　　当然 stdio.h 中也有 NULL 的定义

　　但说 NULL 被定义为 0
　　不知道老谭用的到底是什么编译器
　　该书的编译器是 VC++6.0
　　在 VC++6.0 的 stdio.h 中
　　NULL 并非是被定义成了 0
　　而是
　　#define NULL　　((void *)0)

P 291

任何指针变量或地址都可以与 NULL 作相等或不相等的比较
评：既然前面说“指针就是地址”
　　那么“指针变量或地址”是什么意思？
　　小学语文问题

P 291

使用指针的优点：①提高程序效率
评：未必

P 291

②在调用函数时当指针指向的变量值改变时，这些值能够为主调函数使用，即可以从函数调用得到多个改变的值；
评：这是话吗？

P 291

同时应该看到，指针使用实在太灵活
评：没看到
　　这是在用故弄玄虚的方法来吓唬读者

P 291

使用指针的优点：①提高程序效率
评：未必

第 9 章 用户自己建立数据类型

P 293

第 9 章 用户自己建立数据类型
评：这个标题不恰当
　　这一章讲的是结构体
　　但其实数组、指针也是用户自己建立的数据类型

P 293

```
struct Student
   {int num;
     char name[20];
     char s ex;
```

```
    int age;
    float score;
    char addr[30];
};
```

在定义了结构体变量后，系统会为之分配内存单元。根据结构体类型中包含的成员情况，在 Visual C++中占 63 个字节（4+20+1+4+4+30=63）

评：露怯了

这个表明谭先生对结构体类型所知极其有限

简单地把结构体各成员的长度加起来并不是结构体的尺寸

其实回避掉这个尺寸问题倒不失为一良策

既然无知就要懂得藏拙

P 294

只不过 int 等类型是系统已声明的

评：系统什么时候，在哪声明了？

P 295

计算机对内存的管理是以“字”为单位的（许多计算机系统以 4 个字节为一个“字”）。如果在一个“字”中只存放了一个字符，虽然只占一个字节，但该“字”中的其他 3 个字节不会接着存放下一个数据，而会从下一个“字”开始存放其他数据。因此在用 sizeof 运算符测量 student1 的长度时，得到的不是理论值 63，而是 64，必然是 4 的倍数。

评：“计算机对内存的管理是以“字”为单位的（许多计算机系统以 4 个字节为一个“字”）”

掩盖无知的秘诀之一的大而化之

笼统地说“计算机对内存的管理”

但是谁都知道计算机包括硬件软件

硬件包括 CPU MU IO 设备等

软件包括 OS 编译器

老谭所谓的计算机是指的那个呢

对不起，我就是不说

求甚解的话让你自己去瞎猜

不求甚解的话正好蒙过去了

“许多计算机系统以 4 个字节为一个“字”” 同样是企图蒙混过关

既然“许多”，那么想必还有不是以 4 个字节为一个字的

试问这种情况又怎么说呢？

至于“得到的不是理论值 63，而是 64”

这东西本来就没有理论值

而且我特意用 VC++编译器试了一下

得到的却是 68

这个 68 足以说明前面老谭所说的都是在胡扯

P 296

只能对变量赋值，存取或运算，而不能对一个类型赋值、存取或运算。

评：1.

赋值本身就是运算

2.

谁说对类型不能运算

sizeof(int) 难道不是运算？

P 298

在程序中可以对变量的成员赋值，例如：

student.num=10010;

“.”是成员运算符，它在所有的运算符中优先级最高，因此可以把 student.num 作为一个整体来看待，相当于一个变量。

评：1.如果变量的成员是用 const 限定的，就不可以对它赋值

2.确实“可以把 student.num 作为一个整体来看待，相当于一个变量”，但这和“它在所有的运算符中优先级最高”没有因果关系，因此“因此”二字是牵强附会。

P 298

```
struct Date
   {int month;
    int day;
    int year;
   };
struct Student
   {int num;
    char name[20];
    char se x;
    int age;
    struct Date birthday;
    char addr[30];
   };
struct Student student1;
```

如果成员本身又属一个结构体类型，则要用若干个成员运算符，一级一级地找到最低级的一级成员。只能对最低级的成员进行赋值或存取以及运算。……

不能用 student1.birthday 来访问 student1 变量中的成员 birthday，因为 birthday 本身是一个结构体

成员。
评：这简直是瞪着眼睛胡说八道

P 298

student1.age++;
由于“.”运算符的优先级最高，因此 student1.age++是对(student1.age)进行自加运算，而不是先对 age 进行自加运算。
评：错。“.”和“++”优先级一样高

P 299

说明：结构体变量的地址主要用作函数参数，传递结构体变量的地址。
评：这个绝对是无稽之谈
如果说结构体变量简化了函数间的参数可能还有点贴边
但是说指向结构体的指针的主要作用是函数参数
基本上就是胡扯了

P 299

例 9.2 输入两个学生的学号、姓名和成绩，输出成绩较高的学生的学号、姓名和成绩。

```
#include <stdio.h>
int main()
{struct Student
    { int num;
       char name[20];
       float score;
    }student1,student2;
scanf("%d%s%f",&student1.num,student1.name,&student1.score);
scanf("%d%s%f",&student2.num,student2.name,&student2.score);
printf("The higher score is:\n");
if(student1.score>student2.score)
    printf("%d  %s  %6.2f\n",student1.num,student1.name,student1.score);
else if(student1.score<student2.score)
    printf("%d  %s  %6.2f\n",student2.num,student2.name,student2.score);
else
   {printf("%d  %s  %6.2f\n",student1.num,student1.name,student1.score);
     printf("%d  %s  %6.2f\n",student2.num,student2.name,student2.score);
   }
return 0;
}
```

评：从没见过这么傻的代码
居然如此笨拙地使用结构体
跟买辆汽车用驴拉没什么两样

P 300

输出 student1 的全部信息是轻而易举的……如果用普通变量则难以方便地实现这一目的
评：恰恰相反
这种不恰当地使用结构体的方式效果反而不如用普通变量

P 300

例 9.3 有 3 个候选人，每个选民只能投票选一人，要求编一个统计选票的程序，先后输入被选人的名字，最后输出各人得票结果。

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

struct Person {char name[20];
               int count;
              }leader[3]={"Li",0,"Zhang",0,"Sun",0};
int main()
{int i,j;
char leader_name[20];
for(i=1;i<=10;i++)
   {scanf("%s",leader_name);
     for(j=0;j<3;j++)
       if(strcmp(leader_name,leader[j].name)==0)leader[j].count++;
   }
printf("\nResult:\n");
for(i=0;i<3;i++)
   printf("%5s:%d\n",leader[i].name,leader[i].count);
return 0;
}
```

评：把类型声明写在了前面，很好。但同时又定义了一个外部变量，不伦不类，毫无意义且肆无忌惮地破坏代码结构
为只有一个 main()的代码定义外部变量无论怎么说都属于头脑不清醒。严重的误导

代码中出现了一个莫名其妙的 10，非但是一个 Magic Number,而且毫无依据

就这个写法（只有一个 main()）而言，不使用结构体，代码更简洁

还有一点，这个代码的命名很荒唐。
我发现有个奇怪的现象，主张用英文单词做标识符的人从来不指责这种荒唐的英文

P 301

```
Li
Li
Sun
Zhang
Zhabg
Sun
Li
Sun
Zhang
Li

Result:
   Li:4
Zhang:2
  Sun:3
```

评：结果很荒唐
4+2+3 共 9 票
但书中对此没有做任何交代和说明

P 301~302

例 9.4 有 n 个学生的信息（包括学号、姓名、成绩），要求按照成绩的高低顺序输出各学生的信息

```
#include <stdio.h>
struct Student {  int num;
                  char name[20];
                  float score;
               };
int main()
{struct Student stu[5]={{10101,"Zhang",78},{10103,"Wang",98.5},{10106,"Li",86},
         {10108,"Ling",73.5},{10110,"Sun",100}};
struct Student temp;
const int n=5;
int i,j,k;
printf("The order is:\n");
for(i=0;i<n-1;i++)
   {k=i;
    for(j=i+1;j<n;j++)
      if(stu[j].score>stu[k].score)
```

```
            k=j;
     temp=stu[k];stu[k]=stu[i];stu[i]=temp;
    }
for(i=0;i<n;i++)
    printf("%6d%8s%6.2f\n",stu[i].num,stu[i].name,stu[i].score);
printf("\n");
return 0;
}
```

评：这段代码最可笑的就

const int n=5;

企图追认数组尺寸

其次

printf("The order is:\n");

这句话的位置也不正确

其他的毛病这里就不说了

P 302

程序分析：

（1）程序中第 11 行定义了常变量 n，在程序运行期间它的值不能改变。如果学生数改为 30，只须把第 11 行改为下行即可，其余各行不必修改。

```
const int n=30;
```

也可以不用常变量，而用符号常量，可以取消第 11 行，同时在第二行前加一行：

```
#define N 5
```

读者可比较这两种方法，觉得哪一种方法方便好用？

评：其实两种方法都行不通

都是一厢情愿

常变量这个翻译非常蹩脚

而且 n 的值也并非“在程序运行期间它的值不能改变”

P 303

一个结构体变量的起始地址就是这个结构体变量的指针。如果把一个结构体变量的起始地址存放在一个指针变量中，那么，这个指针变量就是指向该结构体变量

评：“一个结构体变量的起始地址就是这个结构体变量的指针”

struct x{ char c ;} s;

&s.c 和 & s 的值相同，但前者并不是这个结构体变量的指针

“如果把一个结构体变量的起始地址存放在一个指针变量中，那么，这个指针变量就是指向该结构体变量”

void *q = &s;

q 并不指向结构体变量

P 303

指向结构体对象的指针既可以指向结构体变量，也可以指向结构体数组中的元素。

评：这是毫无意义的废话

P 305

```
#include <stdio.h>
struct Student
  {int num;
   char name[20];
   char six;
   int age;
};
struct Student stu[3]={{10101,"Li Lin",'M',18},{10102,"Zhang Fang",'M',19},
                       {10104,"Wang Min",'F',20}};
int main()
{struct Student *p;
printf("No.  Name      six age\n");
for(p=stu;p<stu+3;p++)
printf("%5d %-20s%2c%4d\n",p->num,p->name,p->six,p->age);

return 0;
}
```

评：只有一个 main()函数，但却使用了外部变量，无厘头

只有古代的 C 语言才必须这样

P 305

本例中一个元素所占的字节数理论上为 4+20+1+4=29 字节

评：什么“理论”

根本就不存在这种理论，哪里谈得上“理论上”？

P 306

不要认为反正 p 是存放地址的，可以将任何地址赋给它。

评：造成这种错误认识的根源恰恰是老谭自己一向把地址与指针混为一谈
在 303 页，老谭是这样写的

一个结构体变量的起始地址就是这个结构体变量的指针。如果把一个结构体变量的起始地址存放在一个指针变量中，那么，这个指针变量就是指向该结构体变量

P 306

如果要将某一成员的地址赋给 p，可以用强制类型转换，先将成员的地址转换成 p 的类型。例如：
p=(struct Student *)stu[0].name;
此时，p 的值是 stu[0]元素的 name 成员的起始地址。可以用“printf("%s",p);”输出 stu[0]中成员 name 的值。但是，p 仍保持原来的类型。如果执行“printf("%s",p+1);”，则会输出 stu[1]中 name 的值。执行 p++时，p 的值增加了结构体 struct Student 的长度。
评：这是 胡闹+教唆
一相情愿的臆测

P 306

在调用 inpu 函数时，将主函数中的 stu 数组的首元素的地址传给形参数组 stu，使形参数组 stu 与主函数中的 stu 数组具有相同的地址
评：概念错误
形参 stu 的地址是&stu ， 它不可能与 main()中的 stu 或&stu 相同

P 309

实参是指针变量 p，形参是结构体数组
评：形参不可能是数组，哪怕形式上是用[]声明的
实参也不是指针变量 p，严格的说法是 p 的值

P 309

用数组存放数据时，必须事先定义固定的数组长度（即元素个数）
评：从上下文来看，这本号称“按照 C99 标准进行介绍”的书的作者明显忘记了 C99 还有 VLA

P 310

例 9.8 建立一个如图 9.9 所示的简单链表，它由三个学生数据的结点组成，要求输出各结点中的数据。
评：很难说图 9.9 所示的是否是链表,因为没头没尾
而且和后面给出的代码不一致

P 310~311

```
#include <stdio.h>
struct Student
{
  int num;
  float score;
  struct Student *next;
};
int main()
{struct Student a,b,c,*head,*p;
a.num = 10101;a.score = 89.5;
b.num = 10103;b.score = 90;
c.num = 10107;c.score = 85;
head = &a;
a.next = &b;
b.next = &c;
c.next = NULL;
p = head;
do
   {printf("%ld%5.1f\n",p->num,p->score);
    p=p->next;
   }while (p != NULL );
return 0;
}
```

评：不应该使用 do-while 语句，而应该使用 while 语句

P 311

请读者分析：……②没有头指针 head 行不行。③p 起什么作用，没有它行不行？
评：行！至少可以删去一个，如果用函数实现输出，两个都可以去掉
　　所以定义这两个变量多此一举

P 311

本例是比较简单的，所有结点都是在程序中定义的，不是临时开辟的，也不能用完后释放，这种链表称为“静态链表”
评：所谓“静态链表”是一种捏造，说是“伪链表”倒是比较符合实际
实际上定义 a,b,c 还不如定义一个数组

P 311

例 9.9 写一函数建立一个有 3 名学生数据的单向动态链表。

评：这个题目的要求很滑稽

已经确定是 3 名学生了

建立链表是无聊之极的行为

这和链表的本质是相矛盾的

P 311

解题思路：……定义 3 个指针变量：head,p1 和 p2,它们都是用来指向 struct Student 类型数据的。先用 malloc 函数开辟第一个结点，并使 p1 和 p2 指向它。然后从键盘读入一个学生的数据给 p1 所指的第 1 个结点。在此约定学号不会为零，如果输入的学号为 0，则表示建立链表的过程完成，该结点不应连接到链表中。先使 head 的值为 NULL（即等于 0），这是链表为“空”时的情况（即 head 不指向任何结点，即链表中无结点），当建立第 1 个结点就使 head 指向该结点。

评：总的印象是，这段“思路”颠三倒四，混乱不堪

首先，“定义 3 个指针变量”，没头没脑。其实也没有必要

“先用 malloc 函数开辟第一个结点，并使 p1 和 p2 指向它”，还没输入数据就开辟结点，无事生非

“然后从键盘读入一个学生的数据给 p1 所指的第 1 个结点”，结点是否开辟出来他是不管的

“在此约定学号不会为零，如果输入的学号为 0”，小学语文不过关

“则表示建立链表的过程完成，该结点不应连接到链表中”，当然不应该，应该去造成内存泄露

“先使 head 的值为 NULL（即等于 0),”，又是没头没脑

“当建立第 1 个结点就使 head 指向该结点”，这话应该在前面说吧

P 312

由于 p1->num 的值为 0，不再执行循环，此新结点不应被连接到链表中。此时将 NULL 赋给 p2->next,见图 9.14(b)。建立链表过程至此结束，p1 最后所指的结点未链入链表中

评：嗯。成功地造成了内存泄露

老谭在这里演示了他制造内存泄露的独家秘笈

学这本书的人有望成为破坏之神或者是混乱之神

P 313

图 9.14(b)

评：如果有人不清楚令程序员心惊胆战谈虎色变在写代码时如履薄冰竭力避免的内存泄露是什么

看一下这个就清楚了

P 313~314

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct Student)
struct Student
   {long num;
    float score;
    struct Student *next;
   };
int n;
struct Student *creat(void)
{struct Student *head;
struct Student *p1,*p2;
n=0;
p1=p2=(struct Student *)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
head=NULL;
while(p1->num!=0)
   {n=n+1;
    if(n==1)head=p1;
    else p2->next=p1;
    p2=p1;
    p1=(struct Student*)malloc(LEN);
    scanf("%ld,%f",&p1->num,&p1->score);
   }
  p2->next=NULL;
  return(head);
}

int main()
{ struct Student *pt;
  pt=creat();
  printf("\nnum:%ld\nscore:%5.1f\n",pt->num,pt->score);//输出第 1 个结点的成员值
  return 0;
}
```

评：这段代码的错误和毛病太多了
数不过来
但最严重的一个恐怕是产生了内存泄露(memory leak)

另外，如果一开始就输入 0 值，程序会发生可怕的崩溃
这是因为根本没有考虑
printf("\nnum:%ld\nscore:%5.1f\n",pt->num,pt->score);

这条语句的前提条件是 pt 不为 0
建立链表，居然要借助一个外部变量的做法前无古人
这样对外部变量有依赖性的链表基本上是没有意义的
更为怪异的是
int n;
在定义时不赋初值
却在函数调用内部赋值为 0

P 315

例 9.10 编写一个输出链表的函数 print。

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct Student)
struct Student
   {long num;
    float score;
    struct Student *next;
   };
int n;
void print(struct Student *head)
{struct Student *p;
printf("\nNow,These %d records are:\n",n);
p=head;
if(head!=NULL)
   do
    {printf("%ld %5.1f\n",p->num,p->score);
     p=p->next;
    }while(p!=NULL); //当 p 不是空地址
}
```

评：这段代码很傻
首先
搞不清楚为什么#include <stdlib.h>
其次，那个 n 充分暴露了代码的弱点
第三，此 head 非彼 head，定义 p 并把 head 的值赋给 p，多此一举
第四，

```
if(head!=NULL)
    do
     {printf("%ld %5.1f\n",p->num,p->score);
      p=p->next;
     }while(p!=NULL);
```

这明明就是一个 while 语句，能把一条简单的 while 语句写得如此复杂且别扭，却也难得
第五，即使写 if(head!=NULL)它也应该在

printf("\nNow,These %d records are:\n");
p=head;
之前，而不是之后
第六，“空地址”是老谭发明的新概念，C 语言里没有这东东

P 316

把例 9.7 和 9.8 合起来加上一个主函数，组成一个程序，即：

```
#include <stdio.h>
#include < malloc.h >
#define LEN sizeof(struct Student)
struct Student
   {long num;
    float score;
    struct Student *next;
   };
int n;
struct Student *creat()
{struct Student *head;
struct Student *p1,*p2;
n=0;
p1=p2=(struct Student *)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
head=NULL;
while(p1->num!=0)
   {n=n+1;
    if(n==1)head=p1;
    else p2->next=p1;
    p2=p1;
    p1=(struct Student*)malloc(LEN);
    scanf("%ld,%f",&p1->num,&p1->score);
   }
  p2->next=NULL;
  return(head);
}

void print(struct Student head)
{struct Student *p;
printf("\nNow,These %d records are:\n",n);
p=head;
if(head!=NULL)
   do
    {printf("%ld %5.1f\n",p->num,p->score);
     p=p->next;
```

```
    }while(p!=NULL);
}

void main()
{ struct Student *head;
  head=creat();
  print(head);
}
```

评：

居然弄出个 void main()

老谭不是说 C99 不带这样的吗

此外

```
void print(struct Student head)
{struct Student *p;
printf("\nNow,These %d records are:\n",n);
p=head;
```

是一个显而易见的低级错误

#include <malloc.h>

是一种古老得已经腐朽的写法

P 317

实型变量

评：把 float 称为实型既没有任何依据也没有任何意义

是错误的

P 317

也就是使用覆盖技术，后一个数据覆盖了前面的数据

评：这是故弄玄虚

向内存或寄存器写入后以前的数据就消失，这是存储器的基本性质

根本不是什么“覆盖技术”

P 317

图 9.17　　1000 地址

评：不知所云

P 318

```
union Data
  { int I;
     char ch;
     float f;
   }
```

评：根据上下文
应为 int i;

P 318

结构体变量所占内存长度是各成员占的内存长度之和。每个成员分别占有其自己的内存单元。而共用体变量所占内存长度等于最长成员的长度。
评：大错特错

P 318

只有先定义了共用体变量才能引用它，。
评：如果不算废话的话只能算是错话

P 318

但应注意，不能引用共用体变量，而只能引用共用体中的成员 nion Data
评：胡扯。完全是无稽之谈

P 318

例如，前面定义了 a,b,c 为共用体变量，下面的引用方式是正确的
a.i
a.ch
a.f
不能只引用共用体变量，例如下面的引用是错误的：
printf("%d",a);
因为 a 的存储区可以按不同的类型存放数据，有不同的长度，仅用共用体变量名 a，系统无法知道究竟应该输出哪一个成员的值。
评：printf("%d",a); 确实是错误的
但后面的解释是错误的
printf("%d",a);错误的原因是对于共用体没有相应的转换格式

P 319

```
union Data
  {
     int i;
     char ch;
     float f;
  }a;
a.i=97;
```

……可以用以下的输出语句：

```
printf("%d",a.i);
printf("%c",a.ch);
printf("%f",a.f);
```

评：误导

后两句都是有问题的

无法确定输出什么样的结果

P 319

```
a.ch='a';
a.f=1.5;
a.i=40;
```

……用“printf("%c",a.ch);”，输出的不是字符'a'，而是字符'('。因为……

评：这个问题涉及到计算机字符编码和大小端等问题

是不确定的

P 320

不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。

评：错

很明显，老谭根本不懂得在 C 语言中“值”(value)这个术语的基本含义

P 320

C99 允许同类型的共用体变量互相赋值

评：在谭的这本书中，一再出现把 C99 之前就已经存在的规定硬说成是 C99 的规定

竭力把这本书伪装成 C99（事实上该书使用 VC++这种编译器注定不可能用来学习 C99）

我个人认为这不是学识问题

P 320

C99 允许共用体变量作为函数参数
评：再次用 C89 冒充 C99

P 320

例 9.11 ……要求用同一表格来处理
评：对不起，C 语言里没有“表格”这种东东

P 320

```
#include <stdio.h>
struct
{int num;
char name[10];
char six;
char job;
union
  {int clas;
   char position[10];
  }category;
}person[2];

int main()
{
int i;
for(i=0;i<2;i++)
{printf("please enter the data of person:\n");
  scanf("%d %s %c %c",&person[i].num,&person[i].name,
           &person[i].six,&person[i].job);
  if(person[i].job=='s')
      scanf("%d",&person[i].category.clas);
  else if(person[i].job=='t')
      scanf("%s",&person[i].category.position);
  else
      printf("Input error");
}
printf("\n");
printf("No. name    six job class/position\n");
for(i=0;i<2;i++)
{
```

```
  if(person[i].job=='s')
      printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,
           person[i].six,person[i].job,person[i].category.clas);
  else
      printf("%-6d%-10s%-4c%-4c%-10s\n",person[i].num,person[i].name,
           person[i].six,person[i].job,person[i].category.position);
}
  return 0;
}
```

评：毛病很多

首先，只有一个 main()却随手定义了一个外部数组，无厘头行为

```
  scanf("%d %s %c %c",&person[i].num,&person[i].name,
              &person[i].six,&person[i].job);
```

这句是错误的

```
else
      printf("Input error");
```

没有意义，在输入错误时依然会导致后面错误的输出

```
  if(person[i].job=='s')
      printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,
           person[i].six,person[i].job,person[i].category.clas);
  else
      printf("%-6d%-10s%-4c%-4c%-10s\n",person[i].num,person[i].name,
           person[i].six,person[i].job,person[i].category.position);
```

过于啰嗦

P 322

由于 class 是 C++的关键字，用 Visual C++时不应该用 class 作成员名，故 clas 代表

评：不要侮辱 Visual C++

是老谭自己掰不清 C 和 C++

Visual C++还是能分得清 C 和 C++的

P 322

```
union Categ
  { int banji;
    char position[10];
  }
```

评：int banji;很有趣，与 char position[10]; 堪称中西合璧的绝配

9.6 使用枚举类型

P 323

如果一个变量只有几种可能的值，则可以定义为枚举(enumeration)类型，所谓“枚举”就是指把可能的值一一列举出来，变量的值只限于列举出来的值的范围内。
评：两个错误：
1.不是所有的东西都可以枚举，必须是整数
2.枚举变量的值并非限于列举出的值的范围

P 323

枚举变量 workday 和 weekend 的值只能是 sun 到 sat 之一
评：不对

P 323

……枚举常量。不要因为它们是标识符（有名字）而把它们看做变量，
评：在同书 42 页却是这样说的
常量是没有名字的不变量

P 324

由于枚举变量的值是整数，因此 C99 把枚举类型也作为整型数据中的一种，即用户自行定义的整数类型
评：这个“由于”、“因此”是在瞎扯
根本不存在这样的因果关系
而且这和 C99 没什么关系，早在 C89 中枚举就是整数类型之一
这里“整型数据”是个很烂的概念
因为在书中“整型”不知道有多少种含义
再则，这里的说法与前面 43 页图 3.4 对数据的分类是根本互相矛盾的

P 324~326

例 9.12 口袋中有红、黄、蓝、白、黑 5 种颜色的球若干个。每次从口袋中先后取出 3 个球，问得到 3 种不同颜色的球的可能取法，输出每种排列的情况。

```
#include <stdio.h>
int main()
{enum Color{red,yellow,blue,white,black};
```

```
enum Color i,j,k,pri;
int n,loop;
n=0;
for(i=red;i<=black;i++)
    for(j=red;j<=black;j++)
      if(i!=j)
      { for(k=red;k<=black;k++)
        if((k!=i)&&(k!=j))
          {n=n+1;
            printf("%-4d",n);
            for(loop=1;loop<=3;loop++)
              {switch(loop)
                {case 1:pri=i;break;
                 case 2:pri=j;break;
                 case 3:pri=k;break;
                 default :break;
                }
               switch(pri)
                {case red:printf("%-10s","red");break;
                 case yellow:printf("%-10s","yellow");break;
                 case blue:printf("%-10s","blue");break;
                 case white:printf("%-10s","white");break;
                 case black:printf("%-10s","black");break;
                 default :break;
                }
              }
            printf("\n");
          }
      }
  printf("\ntotal:%5d\n",n);
  return 0;
}
```

评：枚举是个好东西

可惜被 i、j、k 以及那两个丑陋不堪的 switch 给糟蹋了，代码味道很恶

繁琐不堪，丑陋不堪，根本没有体现出半点 enum 的从容和优美

一本书都快结束了，居然还是一个 main()写到底，老谭究竟会不会写代码啊

P 326

显然用枚举变量（red、yellow 等）更直观

评：那 i,j,k 又怎么说？

P 326

此外，枚举变量的值限定在定义时规定的几个枚举元素范围内，如果赋予它其他值，就会出现出错信息，便于检查
评：胡扯

P 327

```
typedef int Integer;
typedef float Real;
……
```
这样可以使熟悉 FORTRAN 的人能用 Integer 和 Real 定义变量，以适应他们的习惯
评：异想天开
　　这样的人根本不可能写 C 代码
　　他们应该适应 C 而不是应该让 C 去适应他们

P 327

```
float *[]     (指针数组)
```
评：错

P 327

```
float (*)[5]     (指向 10 个元素的一维数组的指针)
```
评：两处错误

P 327

```
typedef struct
( int month;
  int day;
  int year;
}Date;
```
评：居然能犯这种低级错误

P 328

按定义变量的方式，把变量名换上新类型名，并且在最前面加“typedef”，就声明了新类型名代表原来的类型。
评：C 语言并没有要求一定要"在最前面"

P 329

同样可以定义字符串类型
评：C 语言根本就没有“字符串类型”

P 329

#define 是在预编译时处理的，它只能作简单的字符串替换。
评：错

P 329

ypedef 是在编译阶段处理的。实际上它并不是作简单的字符串替换，例如：
typedef int Num[10];
Num a;
并不是用"Num[10]"去替代“int”,而是采用如同定义变量的方法先生成一个类型名（就是前面介绍过的讲原来的变量名换成类型名），然后用它去定义变量。
评：这是一种对 typedef 实现方式的虚妄、无知且无聊的揣测
对编程没有任何用处

P 329

数值范围为-32768~32767……数值范围为±21 亿
评：语文问题

P 330

在移植时只须改动 typedef 定义体即可：
typedef long Integer;
评：移植没那么简单
只改 typedef 是远远不够的

第 10 章 对文件的输入输出

P 331

第 10 章 对文件的输入输出
评：这话通吗？

P 331

10.1 C 文件的有关基本知识
评：“C 文件”？
到底想说什么
没这种东西

P 331

操作系统把各种设备都统一作为文件来处理
评：不知道 OS 把 CPU 作为了什么文件？

P 331

从操作系统的角度看，每一个与主机相连的输入输出设备都看做一个文件
评：看来磁盘驱动器也是文件喽

P 331

所谓“文件”一般指在外部介质上数据的集合
评：嗯。纸张也算“外部介质”吧？

P 331

操作系统是以文件为单位对数据进行管理的
评：哦，那就是说一张格式化而没有文件的磁盘上没有数据了

P 331

常将输入输出形象地称为流（stream），即数据流
评：是谭大爷把它形象地理解为流吧
stream 其实恰恰是一个抽象的概念

P 332

数据可以运行环境流入程序中，或从程序流至运行环境
评：对运行环境（execution environment）的误解

P 332

C 的数据文件由一连串得字符（或字节）组成，而不考虑行的界限，两行数据间不会自动加分隔符，对文件的存取是以字符（或字节）为单位的。
评：错。文本流是由行分隔的字符序列

P 332

文件标识包括 3 部分：（1）文件路基；（2）文件名主干；（3）文件后缀
评：文件名主干，又开始发明新概念了
通常的说法是主文件名

P 332

文件名主干命名规则遵循标识符的命名规则
评：常识性错误

P 332

数据文件可分为 ASCII 文件和二进制文件
评：ASCII 文件：这恐怕又是老谭臆造的新概念，没有这种东西
应该是文本文件和二进制文件

P 332

二进制文件……也称之为映像文件（imagine file）
评：前所未闻
至少 C 标准从来没有这种说法

P 332

字符一律以 ASCII 形式存储
评：错
C 语言与 ASCII 码没有必然的关系

P 333

ANSI C 标准采用“缓冲文件系统”处理数据文件
评：没有这回事

再说不是号称“按照 C99 标准”
怎么露怯了
扯什么 ANSI C 标准去了

P 333

缓冲文件系统中，关键的概念是“文件类型指针”，简称“文件指针”
评：驴唇不对马嘴
跟缓冲与否无关

P 334

在程序中可以直接用 FILE 类型名定义变量。每一个 FILE 类型变量对应一个文件的信息区，在其中存放该文件的有关信息。例如，可以定义以下 FILE 类型的变量：
FILE f1;
定义了一个结构体变量 f1，用它来存放一个文件的有关信息。这些信息是在打开文件时由系统根据文件的情况自动放入的，用户不必过问
评：胡扯

P 334

一般不对 FILE 类型变量命名，也就是不通过变量的名字来引用这些变量，而是设置一个指向 FILE 类型变量的指针变量，然后通过它来引用这些 FILE 变量。这样使用起来方便。
评：这与方便无关

P 334

通过文件指针变量能够找到与它相关的文件
评：错
这个指针操作的对象是流

P 335

ANSI C 规定了用标准输入输出函数 fopen 来实现打开文件
评：友情提醒，这里又忘记把 ANSI C 改成 C99 了
实际上还有一个 freopen()函数

P 335

fopen(文件名，使用文件方式)；

例如
fopen("a1","r");
评：不是“文件标识”吗？（332 页）
怎么又改成“文件名”了
“例如”中怎么直接使用了“文件名主干”

P 335

表 10.1 使用文件方式

	如果文件不存在
“a”（追加）	出错
“ab”（追加）	出错

评：这也能弄错

P 336

表 10.1 使用文件方式

	如果文件不存在
“a+”（追加）	出错
“ab+”（追加）	出错

评：错

P 336

用“r”方式打开的文件只能用于向计算机输入……
评：看来在老谭的概念里，文件不属于计算机

P 336

用“w”方式打开的文件只能用于向该文件写数据(即输出文件)，而不能用来向计算机输入。
评：看来在老谭的概念里，文件不属于计算

P 336

如果原来已存在一个以该文件名命名的文件，则再打开文件前先将该文件删去，然后重新建立一个文件。
评：毫无根据的臆测

P 336

如果希望向文件末尾添加新的数据（不希望删除原有的数据），则应该用“a”方式打开。但此时应保证

该文件已存在；否则将得到出错信息。
评：胡扯

P 336

打开文件时，文件读写位置标记移到文件末尾
评：错
这是由实现定义的

P 336

在每个数据文件中自动设置了一个隐式的“文件读写位置标记”，它指向的位置就是当前进行读写的位置
评：看到这个还以为是科幻小说呢

P 337

但目前使用的有些 C 编译系统可能不完全提供所有这些功能（例如，有的只能用"r"、"w"、"a"方式），有的 C 版本不用"r+","w+","a+"，而用"rw","wr","ar"等，请读者注意所用系统的规定。
评：这是几十年前的陈词滥调
几十年后还这样写就太可笑了

P 337

计算机输入从 ASCII 文件读入字符时，遇到回车换行符，系统把它转换为一个换行符，在输出时把换行符转换成为回车和换行两个字符。
评：这只是 DOS 或 Windows 下的情况而已

此外“ASCII 文件”也是胡编乱造的不存在的概念
C90 已经支持 wchar,难道这也是 ASCII？

P 337

可以通过这 3 个指针变量以上 3 种流进行操作
评：这是什么话呀

P 337

如果程序中指定要从 stdin 所指的文件输入数据，就是指从终端键盘输入数据
评：stdin 表示标准输入流，未必就是键盘，尽管很多的时候是

P 338

对文件读写数据的顺序和数据在文件中的物理顺序是一致的。
评：有没有搞错啊，什么叫文件的“物理顺序”
请谭先生先了解一下再说好吗

P 338

表 10.2
fgetc(fp)　　读成功，带回所读的字符，失败则返回文件结束标志 EOF(即-1)
评：“带回所读的字符”这个说法含糊不清
文件结束标志和 EOF 不是一回事

EOF 也未必就是-1

P 338~339

例 10.1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{FILE *fp;
char ch,filename[10];
printf("请输入所用的文件名：");
scanf("%s",filename);
if((fp=fopen(filename,"w"))==NULL)
   {
   printf("无法打开此文件\n");
   exit(0);
   }
ch=getchar();
printf("请输入一个准备存储到磁盘的字符串(以#结束)：");
ch=getchar();
while(ch!='#')
  {
   fputc(ch,fp);
   putchar(ch);
   ch=getchar();
  }
fclose(fp);
putchar(10);
```

```
return 0;
}
```

评：ilename[10];
这个 10 即使是在 DOS 年代也是一个不合格选择

scanf("%s",filename);
ch=getchar();

这个组合极其笨拙，不如用 gets()干净利索
而且存在着越界的危险

exit(0);
很可笑，在 main()中它和 return 0 没什么区别

```
ch=getchar();
while(ch!='#')
  {
   fputc(ch,fp);
   putchar(ch);
   ch=getchar();
  }
```

这里出现了两句 ch=getchar(); ，完全没必要，丑陋

putchar(10);
令人作呕的写法
降低可移植性和可读性

P 340

例 10.2

```
#include <stdio.h>
#include <stdlib.h>
int main()
{FILE *in,*out;
char ch,infile[10],outfile[10];
printf("输入读入文件的名字：");
scanf("%s",infile);
printf("输入输出文件的名字：");
scanf("%s",outfile);
if((in=fopen(infile,"r"))==NULL)
   {printf("无法打开此文件\n");
   exit(0);
   }
```

```
if((out=fopen(outfile,"w"))==NULL)
   {printf("无法打开此文件\n");
   exit(0);
   }
while(!feof(in))
   {ch=fgetc(in);
    fputc(ch,out);
    putchar(ch);
   }
putchar(10);
fclose(in);
fclose(out);
return 0;
}
```

评：这段代码除了有例 10.1 的所有毛病
还有个非常隐蔽的 BUG
这个 BUG 是由于对 feof()函数的误解造成的
二十多年来这个 BUG 在谭书中一直存在

P 340

在访问磁盘文件时，是逐个字符（字节）进行的
评：晕死
这种外行话都说得出来

P 341

feof(in)是检查 in 所指向的文件是否结束。如果是，则函数值为 1（真），否则为 0（假）
评：“函数值为 1”，错

P 341

运行结果是将 file1.dat 文件中的内容复制到 file2.dat 中。
评：不只如此
还多复制了一个无意义的字符

P 341

C 系统已把 fputc 和 fgetc 函数定义为宏名 putc 和 getc：

```
#define putc(ch,fp)   fputc(ch,fp)
#define getc(fp)   fgetc(fp)
```

评：有的如此
有的不是

P 341

在程序中用 putc 和 fputc 作用是一样的，用 getc 和 fgetc 作用是一样的。在使用形式上，可以把它们当作相同的函数对待
评：恰恰是把前者定义为后者的宏的时候它们不一样，不能当作相同的函数

P 341

用 feof 函数可以检查到文件读写位置标记是否移到文件的末尾
评：feof()函数根本不检查文件读写位置

P 341

fgets(str,n,fp);
作用是从 fp 所指向的文件中读入一个长度为 n-1 的字符串，并在最后加一个’\0’字符，然后把这 n 个字符存放到字符数组 str 中
评：错
字符串是包含'\0'字符的
最多能读 n-1 个字符

P 341

表 10.3 读写一个字符串的函数
从 fp 所指向的文件中读入一个长度为(n-1)的字符串，存放到字符数组 str 中
评：错
最多读 n-1 个字符，且添加 null character

P 341

表 10.3 读写一个字符串的函数
把 str 所指向的字符串写到文件指针变量 fp 所指向的文件中
评：错
结尾的'\0'并不写入

P 341

表 10.3 读写一个字符串的函数

输出成功，返回 0；否则返回非负值
评：胡扯

P 342

fputs 函数……若输出成功，函数值为 0，失败时函数值为 EOF
评：这和表 10.3 说的根本就是驴唇不对马嘴
　　而且都是错的

P 342

fgets 和 fgets 这两个函数……
评：不好意思
　　那是一个函数

P 342

……

```
char str[3][10],temp[10];
int i,j,k,n=3;
……
for(i=0;i<n;i++)
   gets(str[ i]);
```

评：str[3][10],小家子气，极易出错
　　n=3;　再次追认数组尺寸，前无古人的写法

P 343

为运行简单起见，本例只输入 3 个字符串，如果有 10 个字符，只须把第 7 行的 n=3 改为 n=10 即可。
评：很奇怪
　　如此明显的胡说八道竟能三番五次地出现在教科书里

P 343

见程序中第 27 行中的 fputs("\n",fp);
评：其实不如写 fputc('\n',fp);

P 344

fprintf 和 fscanf 函数的读写对象不是终端而是文件

评：fprintf 和 fscanf 函数可以面向标准输入输出设备
printf 和 scanf 函数也可以面向文件

P 345~346

```
struct Student_type
       { char name[10];
          int num;
          int age;
          char addr[30];
       }stud[40];
for(i=0;i<40;i++)
    fread(&stud[ i],sizeof(struct Student_type),1,fp);
```

评：实在是有点辜负 fread()设计者的苦心了

P 346

```
for(i=0;i<40;i++)
    fwrite(&stud[ i],sizeof(struct Student_—type),1,fp);
```

fread 或 fwrite 函数的类型为 int 型
评：没给说成是"整型"是一个进步
但这个说法还是错误的

P 346~347

```
#include <stdio.h>
#define SIZE 10
struct Student_type
   {char name[10];
    int num;
    int age;
    char addr[15];
   }stud[SIZE];

void save()
  {FILE *fp;
   int i;
   if((fp=fopen("stu.dat","wb"))==NULL)
      {printf("cannot open file\n");
       return ;
      }
   for(i=0;i<SIZE;i++)
```

```
        if(fwrite(&stud[i],sizeof(struct Student_type),1,fp)!=1)
          printf("file write error\n");
    fclose(fp);
  }
int main()
{int i;
printf("Please enter data of students:\n");
for(i=0;i<SIZE;i++)
    scanf("%s%d%d%s",stud[i].name,&stud[i].num,&stud[i].age,stud[i].addr);
save();
return 0;
}
```

评：1.随手定义外部变量
2.void save()函数定义不标准
3.void save()中的 return ;是错误的写法
4.fwrite 用得极其笨拙，且出错时没有恰当的处理
5.把 main 的定义写在后面，头重脚轻

P 347

一个 struct Student_type 类型结构体变量的长度为它的成员之和，即 10+4+4+15=33，实际上占 36 字节，是 4 的倍数

评：“一个 struct Student_type 类型结构体变量的长度为它的成员之和”：这是胡扯
“实际上占 36 字节”：自相矛盾
“是 4 的倍数”：没有根据

P 347

用 fopen 函数打开文件时没有指定路径，只写了文件名 stu.dat，系统默认其路径为当前用户所使用的子目录（即源文件所在路径），在此目录下建立一个新文件 stu.dat，输出数据存放在此文件中

评：不指定路径本身就是一种恶习
“源文件所在路径”是胡扯，源文件能运行吗

P 347~348

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
struct Student_type
   {char name[10];
    int num;
```

```
    int age;
    char addr[15];
   }stud[SIZE];

int main()
{int i;
FILE *fp;
if((fp=fopen("stu.dat","rb"))==NULL)    //打开输入文件 stu.dat
   {printf("cannot open file\n");
    exit(0) ;
   }
for(i=0;i<SIZE;i++)
    {fread(&stud[i],sizeof(struct Student_type),1,fp);
     printf("%-10s%4d%4d%-15s\n",stud[i].name,stud[i].num,stud[i].age,stud[i].addr);
    }
fclose(fp);          //关闭文件"stu_list"
return 0;
}
```

评：1.随手定义外部变量

2.exit(0)笨拙且丧失其应有的作用

3.fread(&stud[i],sizeof(struct Student_type),1,fp); 笨拙且没有进行检查

4."打开输入文件 stu.dat" 最后 “关闭文件"stu_list"” 喝多了吧

P 348

在前一个程序中，从键盘输入 10 个学生的数据是 ASCII 码，也就是文本文件。

评：发散性思维！

居然能发散到“文本文件”上去

P 348

在送到计算机内存时，回车和换行符转换成一个换行符，再从内存以“wb”方式（二进制写方式）输出到“stu.dat”文件，此时不发生字符转换，按内存中存储形式原样输出到磁盘文件上。

评：没睡醒吧？

谭大爷！

那个代码根本就没有涉及到这个转换啊

P 348

最后在验证程序中，用 printf 函数输出到屏幕，printf 是格式输出字符，输出 ASCII 码，在屏幕上显示字符。换行符又转换为回车加换行符

评：恭喜谭大爷
穿越成功！
能够对例题中根本不存在的现象发出如此一大篇无中生有的议论
令人佩服
（窃窃地小声问一下，秘诀是不是复制粘贴贴错地方？）

P 348

fread 和 fwrite 函数一般用于二进制文件的输入输出。因为它们是按数据块的长度来处理输入输出的，
评：这个“因为”很可笑
因为根本不存在这个因果关系

P 348

stdin 是指向标准输入流的指针变量
评：它是指针变量吗？
即使在 VC++中它也不是变量啊
谭大爷这结论是哪来的

P 348

若写出
```
fread(&stud[ i],sizeof(struct Student_type),1,fp);
……
```
如用以下形式输入数据：
```
Zhang 1001 19 room_101
……
```
由于 fread 函数要求一次输入 36 个字节（而不问这些字节的内容），因此输入数据中的空格也作为输入数据而不作为数据间的分隔符了。连空格也存储到 stu[i]中了，显然是不对的。
评：告诉读者不可以 fread(&stud[i],sizeof(struct Student_type),1,fp);是对的
但后面的解释就太离谱了
谭大爷总能凭着无中生有的想象力为未定义的行为解释出根本不存在的因为所以
我得说
酱紫“显然是不对的”

P 349

这个题目……（至 10.3 结束）
评：完全错乱
驴唇不对马嘴
复制粘贴错了都不至于如此
有理由怀疑嗑药了

而且编辑也嗑药了
这种出版奇迹只有天朝才能创造得出来

P 349

随机访问不是按数据在文件中的物理位置次序进行读写。
评：无语了

P 350

一般情况下，在对字符文件进行顺序读写时，文件位置标记指向文件开头
评：“字符文件”以及“指向文件开头”都是无中生有

P 350

在下一次执行写操作时把数据写入指针所指的位置
评：什么指针？哪个指针？
上下文中根本就没有这个所谓的“指针”

P 350

对文件读写的顺序和数据在文件中的物理顺序一般是不一致的
评：有没有搞清什么叫“物理顺序”啊

P 350

可以在任何位置写入数据，在任何位置读取数据
评：太夸张了吧
任何计算机中都不存在任何这两个字

P 350~351

例 10.5 有一个磁盘文件，内有一些信息。要求第 1 次将它的内容显示在屏幕上，第 2 次把它复制到另一文件上。

```
#include <stdio.h>
int main()
{FILE *fp1,*fp2;
fp1=fopen("file1.dat","r");
fp2=fopen("file2.dat","w");
while(!feof(fp1))putchar(getc(fp1));
```

```
putchar(10);
rewind(fp1);
while(!feof(fp1))putc(getc(fp1),fp2);
fclose(fp1); fclose(fp2);
return 0 ;
}
```

评：1.feof()错用
2.标准要求 putc()等的实参不能有副效应，函数调用显然不应该作为实参
3.没对文件没打开进行处理

P 351

用函数 feof 可以测出文件位置标记是否已指到文件末尾
评：错

P 351

fseek(文件类型指针，位移量，起始点)
“起始点”用 0、1 或 2 代替，0 代表“文件开始位置”，1 为“当前位置”，2 为“文件末尾位置”
评：这是教人学坏
这个参数应该用 SEEK_SET,SEEK_CUR 或 SEEK_END

此外 fseek()清除 eof 标志的功能没有提及

P 351

fseek 函数一般用于二进制文件
评：文本文件也可以用

P 352

如调用函数出错，ftell 返回值为-1L
评：不仅如此
还会设置 error 的值

P 353

每次位置指针的移动量是结构体变长度的两倍
评：不是说“指示文件读写位置的不宜称为'指针'”（350 页脚注）吗？
这个“位置指针”是怎么回事？

“结构体变长度” ？？？

P 354

应当在调用函数时一个输入输出函数后立即检查 ferror 函数的值。否则信息会丢失
评：不切实际

P 354

在执行 fopen 函数时，ferror 函数的初始值自动置 0
评：要是 fopen 失败呢？
函数哪来的初始值？

P 354

clearerr 的作用是使文件错误标志和文件结束标志置为 0
评：“置为 0”这个说法没有依据

P 354

假设在调用一个输入输出函数时出现错误，ferror 函数值为一个非零值。应该立即调用 cleaerr(fp)，使 ferror(fp)的值变为 0，以便再进行下一次的检测。
评：这是毫无开发经验的人对开发所做的一种幻想

P 354

C 要求对程序中用到的每一个变量都必须定义其类型
评：“定义其类型”，这叫什么话
应该是必须声明

第 11 章常见错误分析

P 355

应在函数体的开头加
int x,y;
评：实际上要在前面对 x,y 进行声明
在函数体的开头加 int x,y; 只是其中的一种声明形式而已

P 355

输入输出的数据的类型与用户指定的输入输出格式声明不一致
评：什么叫“用户指定的输入输出格式声明”？
　　谁是“用户”？

P 355

若 a 已定义为整型，b 已定义为实型
评：“整型”，“实型” 都是错误的概念

P 356

```
nt num;
m=89101;
printf("%d",num);
```

如果用 Turbo C 编译系统，得到的却是 23565，原因是……
评：不要污蔑 Turbo C 好不好
　　这段代码任何编译器都不可能得到 23565

P 356

对于超过整数范围的数，要用 long 型
评：还真不清楚什么样的数会超过“整数范围”
　　分数？小数？复数？
　　long 类型难道不是一种整数类型吗？

P 356

例如：

```
num=198607;
```

输出得-1.因为……
评：对未定义行为解释的头头是道

P 357

在数组名前面多加了&。例如

```
char a[20];
scanf("%s",&a);
```
……数组名 a 本身就是地址，再加&就是画蛇添足了。只须直接写数组名即可：
```
scanf("%s",a);
```
评：这里确实应该直接写数组名
但是“数组名 a 本身就是地址，再加&就是画蛇添足了”这个理由是错误的、荒谬的
因为它无法回答，&a 也是地址，为什么 a 这个地址可以而&a 这个地址不可以这样的问题

P 357

```
scanf("%s",a);
```
这样，输入的字符串就会送到数组名 c 所代表的地址去
评：不会吧。前面写的是 a，怎么会输入到 c 中呢

P 357

```
int a[20];
scanf("%d",a);
```
这是错误的。
评：这不是错误的
其含义是输入 a[0]的值

P 357~358

……多个数据，必须分别通过指定数组元素输入。即
```
int a[20];
int i;
for(i=0;i<20;i++)
   scanf("%d",a[ i]);
```
评：O!MGD!
这个才是错误的
这一章是“常见错误分析”
谭大爷居然现身说法
用更大的错误去纠正错误

P 358

C 语言规定语句末尾必须有分号。分号是 C 语句不可缺少的一部分。
评：胡扯
C 语言根本就没有这个规定
语句可以没有分号

P 358

在 C 语言中，没有分号的就不是语句。
评：考考老谭：
题目：试写出一没有分号的 C 语句。

P 360

if(score=100)n++;
……if 语句检查 score 是否为零。若为非零，则作为“真”；若为零作为“假”
评：错

P 361

在 C 语言中，数组名 a 代表数组 a 的首元素地址……
评：常见误解

P 363

if(score=100)n++;
……if 语句检查 score 是否为零。若为非零，则作为“真”；若为零作为“假”
评：在附录 D 中根本没有写++的优先级高于*，而是把它们当作优先级相同
此外也没什么“先执行”

P 363

```
int main()
  {int *p,a[5]={1,3,5,7,9};
    p=a;
    printf("%d",*p++);
  }
```

……结论是先输出 a[0]的值，然后再使 p 加 1。
评：这个结论是错误的
此外示意代码本身有不符合规范的毛病

P 363

(24)忘记对所调用的函数进行函数原型声明。
……
在编译时有出错信息

评：应该是警告信息

P 363~364

(24)忘记对所调用的函数进行函数原型声明。
①……
②……
评：这部分讲的是“忘记对所调用的函数进行函数原型声明”这种错误
但老谭给出的① ② 两个写法本身也有“忘记对所调用的函数进行函数原型声明”这种错误
叫人情何以堪啊？

P 365

```
int main()
{int a,b,*p1,*p2;
  a=3;b=4;
  p1=&a;p2=&b;
swap(p1,p2);
printf("%d,%d",a,b);
}

void swap(int *pt1,int *pt2)
{ int temp;
   temp=*pt1;*pt1=*pt2;*pt2=temp;
}
```

评：这是谭针对错误代码给出的修改后的代码
可是它本身就是错误连篇
不仅有他刚刚批评过的“忘记对所调用的函数进行函数原型声明”的错误
而且代码不规范
没有写他告诉读者应该写的 return 0;
此外 p1,p2 两个变量多余
风格方面
a=3;b=4;
  p1=&a;p2=&b;
temp=*pt1;*pt1=*pt2;*pt2=temp;
如同挤在一起的几条蛆

P 366

```
int *p1;
float *p2;
```

指向不同类型的指针间的赋值必须进行类型转换。例如：
p2=(float *)p1;
评：吃饱了撑的
这是想告诉读者什么啊

P 366

可以进行如下的类型转换：
```
struct studend
{ int num;
  char name[20];
  float score;
}
struct studend studend1,*p;
p=(struct studend)malloc(LEN);
```
评：LEN 是什么？
studend 是什么意思还真不清楚？古英语？

P 367

```
int i=3;
printf("%d,%d,%d\n",i,++i,++i);
```
……在 Turbo C 和 Visual C++6.0 系统中输出是
5，5，4
因为这些系统的处理方法是：按自左至右的顺序求函数参数的值。
评：这个写法本身就是错的
老谭居然能讲出“因为”

P 367

```
printf("%d,%d,%d\n",i,i++,i++);
```
……在 Turbo C 和 Visual C++6.0 系统中输出是
3，3，3
求值的顺序仍然是自由向左，但是需要注意的是：……由于 i++是“后自加”，是在执行完 printf 语句后再使 i 加 1
评：什么叫不懂装懂
这就是

P 368

```
nt a[5],*p;
p=a;
```

```
for(p=a;p<a+5;p++)
……
```

评：连续赋值两次可以更保险？

P 368

应改为

```
struct worker
  {int num;
    char name[10];
    char six;
    int age;
  };
struct worker  worker1;
……
strcpy(worker1.name ,"Zhang Fang");。
```

评：哦卖糕的

应该再加一段

“应进一步改为”

P 369

（34）在打开文件时，指定的文件名找不到。

如：

```
if((fp=fopen("test","r" ) )==NULL)
{printf("cannot open this file\n" );
   exit(0);
}
```

……应当在指定文件名时指出文件路径。如：

```
if((fp=fopen("D:\temp\test","r" ) )==NULL)
{printf("cannot open this file\n" );
   exit(0);
}
```

评：居然能把正确的改成错误的

太有喜感了

本章谭浩强一共罗列了 35 种错误

但自己犯的错误绝对不少于 35 个

应该把这章的名字《常见错误分析》改为《常见错误展示》

P 370

Visual C++6.0 有英文版和中文版
评：有中文版吗？
从没听说过

P 372

输入例 1.1 程序（见图 A.4）
评：居然是
void main()

P 373

找到已有的 C 程序名……双击此文件名，则进入了 Visual C++集成环境
评：未必

P 377

ASCII 代码
评：听说过 C 代码
ASCII 代码是什么东东？

P 377

128~255 是 IBM-PC 上专用的
评：令人有“今夕是何年”之慨
大概可以用来进行 IT 考古

P 378

附录 C C 语言中的关键字
_bool
评：竟然能把关键字写错
而且是在附录中

P 378~379

附录 D 运算符和结合性

评：首先，标题就不正确
其次，这是一张过时的表格，应该是上个世纪 80 年代的
C90 的一些运算阙如，如 一元 + 运算
C99 的一些新运算就更没有
把两种++、--视为一种运算也是错误的
第 3，优先级错误
根据 C 标准，这个表的 cast 运算优先级不正确
第 4，逗号运算是几目运算搞不清楚
第 5，乱起名称，运算符名称不当
如把 () 叫做“圆括号”
把->叫“指向结构体成员运算符”
把* 叫“指针运算符”

P 379

同一优先级的运算符，运算次序由结合方向决定
评：优先级和结合性，与运算次序没有关系

P 379

初等运算符 () [] -> .
评：概念不清，错乱

P 380

为了容易记忆，使用位运算时可加圆括号
评：看不懂。“加圆括号”怎么是“为了容易记忆”

P 380

附录 E C 语言常用语法提要
评：常识性和概念性错误很多
例如：

P 380

标识符可由字母、数字和下划线组成。
评：按照 C99
这其实是早已过时的说法

P 380

不同的系统对标识符的字符数有不同的规定，一般允许 7 个字符
评：实在弄不清这个 7 有什么依据
但有一点可以肯定
这种说法至少过时二十多年了

P 380

十六进制常数（以 0x 开头的数字序列）
评：常识性错误

P 380

长整型常数（在数字后加字符 L 或 l）
评：什么叫“数字”？
概念性错误

P 380

(2) 字符常量
用单撇号括起来的一个字符，可以使用转义字符
评：“一个字符”，错

P 380

(3) 实型常量（浮点型常量）
评：实数类型和浮点类型是两个完全不同的概念

P 381

算术表达式
评：与其说是自创的概念倒不如说是捏造的概念
C 语言没有这个概念也不需要这个概念

P 380

用单撇号括起来的一个字符，可以使用转义字符
评：“一个字符”，错

P 380

用单撇号括起来的一个字符，可以使用转义字符
评：“一个字符”，错

P 381

实型表达式：参见运算的运算量是实型量，运算过程中先转换成 double 型，结果为 double 型
评：这是在胡扯

P 381

3. 表达式
(1) 算术表达式
(2) 逻辑表达式
(3) 字位表达式
(4) 强制类型转换表达式
(5) 逗号表达式
(6) 赋值表达式
(7) 条件表达式
(8) 指针表达式
评：这个分类很荒唐，除了带来混乱没有任何意义
其中对逻辑运算，逗号运算的描述是片面的
对条件表达式加了不必要的苛刻的约束

P 381

也可以是不包含任何运算的初等量
评：把 Primary expression 说成是 初等量
概念错误
Primary expression 也并非不包含任何运算

P 381

数据定义
评：乱点鸳鸯
根本就没有“数据定义”这回事

P 381

对数据要定义其数据类型，需要时要指定其存储类别
评：常量也是数据
怎么定义类型？
怎么指定存储类别？

P 381

(1)类型标识符可用
评：类型标识符 这个说法不妥
此外列表中缺少若干 type-specifier

P 382

结构体与共用体的定义形式为
评：这里说的应该是类型声明
而不是变量的定义

P 382

如不指定存储类别，作 auto 处理
评：明显忘了还有外部变量

P 383

7. 语句
（1）表达式语句；
（2）函数调用语句；
……
评：逻辑混乱
此外缺少一种语句 labeled-statement

P 383

(5)switch 语句……
评：描述错误

P 384

8. 预处理命令
评：不全

P 384

```
#include <math.h>
int abs(int x);
```
评：abs()不是在 math.h 中声明的

P 384

本书列出 ANSI C 标准建议提供的、常用的库函数
评：记得本书是号称“按照 C99 标准”
C99 标准和 ANSI C 标准完全是两回事

而且所列也并不符合 ANSI C 标准

P 384

使用数学函数时，应该在该源文件中使用以下命令行：
评：在源文件中使用“命令行”

P 385

```
int rand(void)
```
产生—90 到 32767 间的随机整数
评：这个函数不是数学函数
-90 是错误的
32767 也有问题

P 386

```
int isalnum (int ch) ;
```
是字母或数字返回 1，否则返回 0
评：返回 1 的说法不对
后面其余在 ctype.h 中声明的 10 个函数的返回值都不正确

P 386

int iscntrl(int ch);　　检查 ch 是否控制字符（其 ASCII 码在 0 和 0x1F 之间）
评：第 1，执行环境不一定使用 ASCII 码
　　第 2，即使执行环境使用 ASCII 码，控制字符也不仅限于“在 0 和 0x1F 之间”那些

P 386

在 ox21 到 ox7E 之间),不包括空格
评：把 0 写成了 o 是初学者很常见的错误

P 386

ox20 到 ox7E
评：o 0 不分

P 386

int tolower(int ch);　　将 ch 转换为小写字母
评：ch 本身不可能被转换
　　就是转换了也没用
　　这是函数调用的常识

P 386

int toupper(int ch);　　将 ch 转换为大写字母
评：返回和转换不是一回事

P 387

3. 输入输出函数
评：表格中的内容非但不是 C99 的
　　而且连 ANSI C 的都算不上
　　陈旧不堪，错漏百出

P 387

int fclose(FILE *fp);　　有错返回非 0，否则返回 0
评：实际上应该是
　　The fclose function returns zero if the stream was successfully closed, or EOF if any errors were detected.

P 387

int fputc(char ch, FILE *fp);　　成功，则返回该字符；否则返回非 0

评：建议谭先生再次印刷时学学“金瓶梅”
把这些自己弄不清的地方都改为
“XXXXXX（此处删去 78 字）”
这样不但可以增加一些神秘感
书也会干净许多

P 387

fputs

评：很奇怪，这种东西居然能抄错

P 387

fread ……

fscanf……

评：错

P 388

fseek ……

……

评：388 页一共罗列了 14 个函数的原型、功能、返回值
其中有 4 个非标准函数
除去这 4 个非标准函数没有考察
考察了其余 10 个标准函数
其中对 futc()的介绍基本正确（但不够全面）
对其余 9 个的介绍全部有错
这 9 个函数为：
fseek
ftell
fwrite
getc
getchar
printf
putchar
puts
rename

P 389

rewind　将 fp 指示的文件中的位置指针置于文件开头位置，并清除文件结束标志和错误标志
评：位置指针，这个说法不当
　　清除错误标志，无根据。况且 rewind 调用本身就可能产生错误，从逻辑上讲，它怎么也不可能清除错误标志

P 389

int scanf(char * format,args,…);
返回值：读入并赋值给 args 的数据个数，遇文件结束返回 EOF，出错返回 0
评：args,… ：…　在 C 语言中有特定的含义，所以这样的描述很不规范

　　遇文件结束返回 EOF，出错返回 0 ：这是错的

P 389

write　非 ANSI 标准函数
评：对其正确性不予评价
　　但是一本自称“按照 C99 标准”的书把非标准的东西塞进来充数
　　无疑是货不对板以次充好的假冒伪劣
　　是对读者的欺骗

P 389

4.动态存储分配函数
目前有的 C 编译所提供的这类函数返回 char 指针
评：今夕是何年？

P 389

void *calloc(unsigned n, unsign size);
功能：分配 n 个数据项的内存连续空间，每个数据项的大小为 size
返回值：……如不成功，返回 0
评：unsign　：
　　功能：calloc()的一个重要功能是初始化为 0，压根没提
　　返回 0：应为返回 NULL。

　　此外，这是一个过时的原型
　　n 和 size 的类型应该是 size_t ,而 size_t 未必是 unsigned 类型

P 389

free
评：if the argument does not match a pointer earlier returned by the calloc, malloc, or realloc function, or if the space has been deallocated by a call to free or realloc, the behavior is undefined

压根没提
不过也难怪
老谭根本不懂什么是 undefined behavior
undefined behavior 他可以解释得津津有味

P 389

malloc
评：返回 0：应为返回 NULL。
unsigned 应为 size_t

P 389

realloc
评：没有提对原来存储区数据的处理情况
这使得 realloc()看上去和 malooc()毫无区别而失去意义

P 389

4. 动态存储分配函数
评：“内存连续空间”、“内存单元”、“内存区”、“储存区”，其实指的都是一回事，希望能够统一术语。
这个看法是下面网页中的网友贡献的
http://zh.thqerrata.wikia.com/in ... 9&variant=zh-cn

参考文献

P 389

[8]H M. Peitel,
评：应该是 H M. Deitel

《C 程序设计》（第四版）习题辅导错误汇集

《C 程序设计（第四版）学习辅导》部分

P14

假如我国国民生产总值的年增长率为 10%，计算 10 年后我国国民经济生产总值与现在相比增长多少百分比。计算公式为：

p=(1+r)^n

r 为年增长率，n 为年数，p 为与现在相比的倍数。

解：从附录 D（库函数）可以查到：可以用 pow 函数求 x^y 的值，调用 pow 函数的具体形式是 pow(x,y)。在使用 pow 函数时需要在程序的开头用#include 指令将<math.h>头文件包含到本程序模块中。可以用下面的程序求出 10 年后国民生产总值是现在的多少倍。

```
#include <stdio.h>
#include <math.h>
int main()
{float p,r,n;
r=0.1;
n=10;
p=pow(1+r,n);
printf("p=%f\n",p);
return 0;
}
```

运行结果：

```
p=2.593742
```

即 10 年后国民生产总值是现在的 2.593742 倍。

评：这个解答至少存在如下几方面的问题：

1.驴唇不对马嘴

从小学开始，老师就会教导我们不要答非所问。这个程序犯的第一个错误就是答非所问。

题目中要求“计算 10 年后我国国民经济生产总值与现在相比增长多少百分比”，然而程序给出的结果却是“10 年后国民生产总值是现在的 2.593742 倍”。程序错没错，小学生都知道。

写程序最基本的一个常识是，要满足功能要求。不能装疯卖傻，指东打西。

把功能要求撇在一边，自说自话、自娱自乐且像盲人骑瞎马一样地写代码，是头脑不清的表现。这样的代码把小学老师多年的辛勤教诲毁之于一旦，是教小朋友学坏。

2.古怪的常数

10%、10 这两个数是问题给出的条件，这种数据通常应该写成符号常量的方式：

```
#define GROWTH_RATE ((double)10/(double)100)
#define YEARS 10
```

这才是良好的编程习惯。这样的好处至少有以下三点：

1.在某种程度上实现了数据与代码的分离，“把上帝的还给上帝，把魔鬼的交给魔鬼”。这是现代程序

设计的一个基本思想。那种把数据和代码不分青红皂白地搅和在一起“乱炖”的写法，是缺乏基本编程素养的表现。

2.代码更具有可读性。显然“r=GROWTH_RATE;”的写法要比“r=0.1;”要好的多。

3.便于测试。只要将

#define YEARS 10

稍做修改就可以测试其他年后（比如 1 年后）的情况。注意，这个修改是在预处理命令部分进行的，因此对 main()中的代码没有任何影响。这从另一个角度表明了把数据与代码分离开的优越性。

3.滥用变量

很容易看出，代码中的那个变量 p 是压根不必要的。这个 p 变量唯一起到的作用是记录“pow(1+r,n)”的值完成输出，然而“pow(1+r,n)” 本身既然有值，为什么不直接输出呢？printf("p=%f\n",pow(1+r,n));不是很漂亮的一条语句吗？何必画蛇添足地定义一个 p 变量呢？

同理，r 与 n 这两个变量也是毫无必要的，因为代码根本就不需要这两个值改变，pow(1+0.1,（double）10)的值和 pow(1+r,n)完全一样。所以定义 r 与 n 这两个变量纯属于脱裤子放屁，多此一举，是糊里糊涂、莫名其妙的写法。

或问，pow(1+r,n)写起来不是比 pow(1 + 0.1 ,（double）10)更简洁么？不然。常量就是常量，变量就是变量，把常量的值赋值给变量，利用变量的值进行运算事实上增加了代码出错的风险，因为变量的值可能无意中被错误地被改变，并且毫无补偿地浪费内存资源。用增加错误风险并浪费内存资源的代价来换取代码的简洁不是正路子。

4.数据类型的问题

代码中的 p、n、r 被定义成了 float 类型，然而 0.1、pow(1+r,n)都是 double 类型，在

r=0.1

p=pow(1+r,n)

这两个赋值表达式中基本上必然会产生精度损失。这种精度牺牲非常无谓，没有任何回报。

5.文件名的问题

题解中说“将<math.h>头文件包含到本程序模块中”，抱歉，谭大爷，“<”和“>”怎么竟然成了文件名的一部分？（这两个符号可以出现在文件名中吗？——此句有误，删除）那是预处理命令的成分，根本不是文件名的组成成分。

6.不规范

C 标准中提到 main()的写法有两种

main(void){/**/}

和

main(int argc,char *agrv[]){/**/}

把“()”内的形参省略，不符合规范。

最后，给出一个参考代码

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

#define GROWTH_RATE    ( (double)10 / (double)100 )
#define YEARS                        10
```

```
int main( void )
{

    printf("%d 年后我国国民经济生产总值比现在增长  %f%%\n" ,
                YEARS ,
                ( pow ( 1. + GROWTH_RATE , ( double ) YEARS ) - 1. ) / 1. * 100.
            ) ;

    return 0;
}
```

P14~16

2.存款利息的计算。有 1000 元，想存 5 年，……

（1）一次存 5 年期。

……

5 年期定期存款利息为 5.85%；

……

运行结果：

p1=5292.500000

……

评：想钱想疯了是怎么的

很想问一下，谭大爷是在哪家银行存钱的?

许霆可是给人家判了个无期的，您这个要是真敢去取估计判您个恶意取款是一点问题都没有的

P17

……

float d=300000,

……

评：这是很糟糕的写法

在一定条件下会发生错误

P17

只要知道：用 char 类型变量时，给它赋的值应在 0~127 范围内

评：作茧自缚

C 语言根本没有这个限制

P18

int c1,c2;
c1=197;
c2=198;
printf("c1=%c,c2=%c",c1,c2);
评：谭认为
printf("c1=%c,c2=%c",c1,c2);的输出结果“不可预料”
这是错误的

P18

scanf("a=%db=%d",&a,&b);
评：变态和误导

P18

只要知道：用 char 类型变量时，给它赋的值应在 0~127 范围内
评：作茧自缚
C 语言根本没有这个限制

P19

关于习题解答（习题 6）
评：滥用变量

P19

7.设圆半径 r=1.5,……求……圆球表面积，圆球体积……

P20~23

关于习题解答（习题 8）
评：把溢出行为当成正常行为

P25~26

从键盘输入一个小于 1000 的正数，要求输出它的平方根（如平方根不是整数，则输出其整数部分）。要求在输入数据后先对其进行检查是否为小于 1000 的正数。若不是，则要求重新输入
……将程序改为多次检查，直到正确输入为止。程序如下：

```
#include <stdio.h>
#include <math.h>
#define M 1000
int main()
{
   int i,k;
   printf("请输入一个小于%d 的整数 i:",M);
   scanf("%d",&i);
   while(i>M)
   {printf("输入的数据不符合要求，请重新输入一个小于%d 的整数 i:",M);
      scanf("%d",&i);
      k=sqrt(i);
   }
   printf("%d 的平方根的整数部分是%d\n",i,k);
   return 0;
}
```

评：错误百出

首先，输入 -1 ，竟然能输出结果

其次，输入 100，输出一个莫名其妙的结果

第三，题目：“输入一个小于 1000 的正数”，代码：“i>M”，

第四，scanf()写了两次，啰嗦

第五，sqrt()并不能保证结果>=平方根

第六，double sqrt(double)， sqrt(i)的写法不严谨

这种“学习辅导”让学习者情何以堪？

评：首先说说题目。编程的题目一般有两种形式：要么描述程序的行为，仅仅要求编程者设计代码；要么纯粹地只描述问题，程序的行为及代码都由编程者设计。前者相当于给出了软件规格说明（Software Specification）。后者则相当于提出了一个软件需求。这个题目无疑属于前一种情况。

软件规格说明应当对软件应满足的要求，以可验证的方式作出完全、精确的描述。

但是题目中的“从键盘输入一个小于 1000 的正数”却并不是一个完全、精确的描述。程序员在看到这个要求之后不可能知道这个数据究竟应该具有什么样的性质，这个“正数”究竟是整数还是小数？如果不清楚这个，在代码中就无法在确定这个数据的类型。当然也不可能完成程序。

那么，自作聪明地假设一个怎么样？对不起，这是一种职业恶习，完全背离程序员的职业的基本准则。如果学习编程的结果是养成了一种职业恶习，显然与学习编程的初衷南辕北辙。因此这样的题目简直就是打着红旗反红旗。

描述程序的行为的题目中可能涉及到输入。输入是由程序用户完成的，用户的输入可能正确也可能有错误。

如果考虑到程序的强健性，题目通常要求程序考虑用户输入有错误的情况。如果不考虑程序的强健性，那么代码可以只考虑用户输入没有错误的情况。通常，题目要么要求编程者考虑强健性，要么不考虑强健性。

从题目中的“检查是否为小于 1000 的正数”来看，题目显然是要求程序具有一定的强健性。然而题目却没有说明在输入不满足的情况下程序应有的行为——即第二次输入不小于 1000 的正数时程序的行为，所以这个题目本身就是不完整的。

既然要求考虑健壮性，就应该把这个意图贯彻始终。不能虎头蛇尾，前后自相矛盾。因而针对用户输入不满足的“正数”的情况下程序的行为，题目也应该给出相应的说明。然而题目中对此却只字未提，毫无疑问，这个题目本身就是一个不合格的题目。

求解烂题危害是很大，因为烂题本身就是违背程序员根本职业要求——周密、严谨。解这样的题目，一无所获不说，反而有伤自身的素质，而得到的代码也必然似是而非，经不起推敲。

测试一下前面引文中的代码就不难发现，当输入“-1”时，程序立刻崩溃——这个程序无比脆弱。然而自相矛盾的是题目却暗示需要考虑程序的健壮性。

此外当第一次输入大于等于 1000 的数，且第二次依然输入大于等于 1000 的数的情况下，程序依然能给出结果。代码中的那句 if 语句在这种情况下竟然毫无意义。这样的代码毫无价值。

结论就：路边的野花不要采，书上的滥题不要做。初学者，乱做习题你伤不起啊!!!!!!!

此外需要指出的是，代码中的

“k=sqrt(i);”

也是一种武断的错误写法，它是建立在 sqrt(i)得到的值一定大于或等于 i 的平方根这个假设之上的，然而这个假设没有任何依据。

“要求输出它的平方根（如平方根不是整数，则输出其整数部分）”，这句话也说的缺乏素质，至少不够简洁。无非就是输出它平方根的整数部分么，如此简单的意思怎么会说的那么复杂且啰嗦不清呢

评：这题目设计的漏洞百出，莫名其妙

而且在刚刚学完选择结构的条件下根本没办法完成

“从键盘上输入一个正数”是一个不明确的要求，输入整数还是实数？

“要求输出它的平方根（如平方根不是整数，则输出其整数部分）”更是啰嗦的不成话，无非是输出其平方根的整数部分而已

对于不可能求解的题目，老谭给出的解答

```
#include <stdio.h>
#include <math.h>
#define M 1000
int main()
{
int i,k;
printf("请输入一个小于%d 的整数 i:",M);
scanf("%d",&i);
if(i>M)
{printf("输入的数据不符合要求，请重新输入一个小于%d 的整数 i:",M);
   scanf("%d",&i);
}
k=sqrt(i);
printf("%d 的平方根的整数部分是%d\n",i,k);
```

return 0;
}
评：当输入一个负值的时候，立刻崩溃
　　而且如果两次输入都是大于 1000 的情况下，依然能输出结果

P29

70~70 分为'C'

P30~31

9.给一个不多于 5 位的正整数，要求：
1）求出它是几位数；
2）分别输出每一位数字；
3）按逆序输出各位数字，例如原数为 321，应输出 123.
解：程序如下：

```
#include <stdio.h>
#include <math.h>
int main()
{
   int num,indiv,ten,hundred,thousand,ten_thousand,place;
                                                    //分别代表个位，十位，百位，千位，万位和位数
   printf("请输入一个整数(0-99999):");
   scanf("%d",&num);
   if(num>9999)
         place=5;
   else if(num>999)
         place=4;
   else if(num>99)
         place=3;
   else if(num>9)
         place=2;
   else place=1;
   printf("位数：%d\n",place);
   printf("每位数字为：");
   ten_thousand=num/10000;
   thousand=(int)(num-ten_thousand*10000)/1000;
   hundred=(int)(num-ten_thousand*10000-thousand*1000)/100;
   ten=(int)(num-ten_thousand*10000-thousand*1000-hundred*100)/10;
   indiv=(int)(num-ten_thousand*10000-thousand*1000-hundred*100-ten*10);
   switch(place)
      {case 5:printf("%d,%d,%d,%d,%d",ten_thousand,thousand,hundred,ten,indiv);
```

```
            printf("\n 反序数字为：");
            printf("%d%d%d%d%d",indiv,ten,hundred,thousand,ten_thousand);
            break;
    case 4:printf("%d,%d,%d,%d",thousand,hundred,ten,indiv);
            printf("\n 反序数字为：");
            printf("%d%d%d%d",indiv,ten,hundred,thousand);
            break;
    case 3:printf("%d,%d,%d",hundred,ten,indiv);
            printf("\n 反序数字为：");
            printf("%d%d%d",indiv,ten,hundred);
            break;
    case 2:printf("%d,%d",ten,indiv);
            printf("\n 反序数字为：");
            printf("%d%d",indiv,ten);
            break;
    case 1:printf("%d",indiv);
            printf("\n 反序数字为：");
            printf("%d",indiv);
            break;
  }
 return 0;
}
```

评：1.似是而非

例如：

thousand=(int)(num-ten_thousand*10000)/1000;

#include <math.h>

2.拖泥带水

例如那个 switch 语句

3.标识符有点怪异。

英语不好，不敢妄加评论，呵呵

但发现只要写的像英文单词，总是受到格外的宽容。

P35

有 4 个圆塔，圆心分别为(2,2),(-2,2),(-2,-2),(2,-2),圆半径为 1，见图 4.5。这 4 个塔的高度为 10m，塔以外无建筑物。今输入任一点的坐标，求该点的建筑物高度（塔外的高度为零）。

```
#include <stdio.h>
int main( )
{
  int h=10;
  float x1=2,y1=2,x2=-2,y2=2,x3=-2,y3=-2,x4=2,y4=-2,x,y,d1,d2,d3,d4;
```

```
    printf("请输入一个点(x,y):");
    scanf("%f,%f",&x,&y);
    d1=(x-x4)*(x-x4)+(y-y4)*(y-y4);//求该点到各中心点距离      d2=(x-x1)*(x-x1)+(y-y1)*(y-y1);
    d3=(x-x2)*(x-x2)+(y-y2)*(y-y2);
    d4=(x-x3)*(x-x3)+(y-y3)*(y-y3);
    if(d1>1&&d2>1&&d3>1&&d4>1) h=0 ;    //判断该点是否在塔外
    printf("该点高度为%d\n",h);
    return 0;
}
```

评：int h=10;

float x1=2,y1=2,x2=-2,y2=2,x3=-2,y3=-2,x4=2,y4=-2

这些都是常量，没有理由设置变量

d1=(x-x4)*(x-x4)+(y-y4)*(y-y4); //求该点到各中心点距离

量纲不对，这不是距离

d1>1&&d2>1&&d3>1&&d4>1

概念错误。仿佛用面积与长度相比较

P37~38

1.请画出例 5.6 中给出的 3 个程序段的流程图。

……

图 5.1

……

图 5.2

……

图 5.3

评：终于有幸见到传说中史前的“耗子窝”和“烂面条”了

比见到恐龙复活还惊奇

P40

输入两个正整数，求其最大公约数和最小公倍数。

```
#include <stdio.h>
int main( )
{
    int p , r , n , m , temp ;
    printf("请输入两个正整数 n,m:");
    scanf("%d,%d,",&n,&m);
    if(n<m)
      {
```

```
        temp=n;
        n=m;
        m=temp;
      }
    p=n*m;
    while(m!=0)
      {
        r=n%m;
        n=m;
        m=r;
      }
    printf("它们的最大公约数为:%d\n",n);
    printf("它们的最小公倍数为:%d\n",p/n );
    return 0;
}
```

评：经典的算法，竟然给糟蹋成这样

那个比较交换完全没必要

后面的 while 语句也写的拖泥带水

p=n*m;也不够好，使代码的适用范围降低

下面的写法是常识性的写法

```
while((r=m%n)!=0)
    {
      m=n;
      n=r;
    }
```

P42~43

$$7.\text{求}\ \sum_{k=1}^{100} k + \sum_{k=1}^{50} k^2 + \sum_{k=1}^{10} (1/k)$$

```
#include <stdio.h>
int main()
{
    int n1=100,n2=50,n3=10;
    double k,s1=0,s2=0,s3=0;
    for(k=1;k<=n1;k++)
        {s1=s1+k;}
    for(k=1;k<=n2;k++)
        {s2=s2+k*k;}
    for(k=1;k<=n3;k++)
        {s3=s3+1/k;}
    printf("sum=%15.6f\n",s1+s2+s3);
    return 0;
```

}

评：最大的毛病是用 double 类型做循环的计数器变量

记数应该用整数类型

这是一个编程常识

n1,n2,n3 明显是画蛇添足

连变量常量都没弄清楚

s1,s2,s3 也没必要那么多

{sl=sl+k;}

这种怪异的风格不伦不类

搞不清写{}的意义何在

此外

s1=s1+k

应该写成

s1+=k;

p43~44

9.一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子为 1,2,3，而 6=1+2+3,因此 6 是“完数”。编程找出 1000 之内所有完数，并按下面格式输出其因子：

6 ,Its factors are 1 2 3

解：方法一。

程序如下：

```
#define M 1000                              //定义寻找范围
#include <stdio.h>
int main()
{
   int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;
   int i,a,n,s;
   for(a=2;a<=M;a++)                        //a 是 2~1000 之间的整数，检查它是否完数
    {n=0;                                   //n 用来累计 a 的因子的个数
     s=a;                                   //s 用来存放尚未求出的因子之和，开始时等于 a
      for(i=1;i<a;i++)                     //检查 i 是否 a 的因子
         if(a%i==0)                         //如果 i 是 a 的因子
     {n++;                                  //n 加 1，表示新找到一个因子
      s=s-i;                                //s 减去已找到的因子，s 的新值是尚未求出的因子之和
      switch(n)                             //将找到的因子赋给 k1~k9，或 k10
        {case 1:
             k1=i;break;                    //找到的第 1 个因子赋给 k1
         case 2:
             k2=i;break;                    //找到的第 2 个因子赋给 k2
         case 3:
             k3=i;break;                    //找到的第 3 个因子赋给 k3
         case 4:
             k4=i;break;                    //找到的第 4 个因子赋给 k4
         case 5:
             k5=i;break;                    //找到的第 5 个因子赋给 k5
         case 6:
             k6=i;break;                    //找到的第 6 个因子赋给 k6
         case 7:
             k7=i;break;                    //找到的第 7 个因子赋给 k7
         case 8:
             k8=i;break;                    //找到的第 8 个因子赋给 k8
         case 9:
             k9=i;break;                    //找到的第 9 个因子赋给 k9
         case 10:
             k10=i;break;                   //找到的第 10 个因子赋给 k10
        }
     }
```

```
    if(s==0)
    {
      printf("%d,Its factors are ",a);
      if(n>1)printf("%d,%d",k1,k2);   //n>1 表示 a 至少有 2 个因子
      if(n>2)printf(",%d",k3);        //n>2 表示 a 至少有 3 个因子
      if(n>3)printf(",%d",k4);        //n>3 表示 a 至少有 4 个因子
      if(n>4)printf(",%d",k5);        //以下类似
      if(n>5)printf(",%d",k6);
      if(n>6)printf(",%d",k7);
      if(n>7)printf(",%d",k8);
      if(n>8)printf(",%d",k9);
      if(n>9)printf(",%d",k10);
      printf("\n");
    }
  }
  return 0;
}
```

评：这代码！丑的都能惊动 D 中秋了

1.题目本身是错的：“一个数如果恰好等于它的因子之和”

6 的因子为 1,2,3 6 的因子还有 6

2.int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;

这个巨生猛，一口气生了 10 个，颇有愚公移山的气概，吃奶的力气都使出来了
为什么偏偏定义 10 个呢？莫名其妙

3.s=a;　　//s 用来存放尚未求出的因子之和，开始时等于 a
实际上 s 不是注释中所说的含义
赋值为 a 的做法也非常笨拙

4.

```
        if(a%i==0)          //如果 i 是 a 的因子
    {n++;
```

雷人的风格

5.6

```
      switch(n)             //将找到的因子赋给 k1~k9，或 k10
        {case 1:
```

“将找到的因子赋给 k1~k9，或 k10 ”，看起来不像中国话
case 1:的位置也很扎眼

7.
if(n>1)printf("%d,%d",k1,k2); //n>1 表示 a 至少有 2 个因子
居然能一口气连写 9 句
作者耐力很好

8.
if(n>3)printf(",%d",k4); //n>3 表示 a 至少有 4 个因子
if(n>4)printf(",%d",k5); //以下类似
原来注释还能这么写

9.
更滑稽的是运行结果
6,Its factors are 1,2,3
28,Its factors are 1,2,4,7,14
496,Its factors are 1,2,4,8,16,31,62,124,248
根本不符合题目的要求
按下面格式输出其因子：
6 ,Its factors are 1 2 3

P47~48

用迭代法求 x= √a。求平方根的迭代公式为
xn+1=1/2(xn+a/xn)
要求前后两次求出的 x 的差的绝对值小于 10^-5。

```
#include <stdio.h>
#include <math.h>
int main(    )
{
   float a,x0,x1;
   printf("enter a positive number:");
   scanf("%f",&a);
   x0=a/2;
   x1=(x0+a/x0)/2;
   do
   {x0=x1;
    x1=(x0+a/x0)/2;
   }while(fabs(x0-x1)>=1e-5);
    printf("The square root of %f is %f\n",a,x1);
    return 0;
}
```

评：1.重复笨拙

```
x1=(x0+a/x0)/2;
do
{x0=x1;
```

x1=(x0+a/x0)/2;

2.fabs(x0-x1)>=1e-5 基本没有意义，因为 float 类型的精度非常有限

P51~52

16.输出以下图案：

```
    *
   ***
  *******
 *********
  *******
   ***
    *
```

```
#include <stdio.h>
int main()
{int i,j,k;
for(i=0;i<=3;i++)
    {for(j=0;j<=2-i;j++)
        printf(" ");
     for(k=0;k<=2*i;k++)
        printf("*");
     printf("\n");
    }
for(i=0;i<=2;i++)
    {for(j=0;j<=i;j++)
        printf(" ");
     for(k=0;k<=4-2*i;k++)
        printf("*");
     printf("\n");
    }
  return 0;
}
```

评：生动地介绍了如何把一个极其简单的问题弄得无比复杂的方法

实际上这个题目只要

```
#include <stdio.h>
int main()
{
   printf("       *       \n");
   printf("      ***      \n");
   printf(" ******* \n");
```

```
        printf("**********\n");
        printf(" ******* \n");
        printf("     ***     \n");
        printf("      *      \n");
        return 0;
    }
```

就可以了

回复 1453# pmerofc

这个貌似把后面的几个 printf 都去掉更好看一些

是的。谢谢

下面写法更好

```
#include <stdio.h>
int main()
{
   printf( "      *      \n"
           "     ***     \n"
           " ******* \n"
           "**********\n"
           " ******* \n"
           "     ***     \n"
           "      *      \n"   );

   return 0;
}
```

P52~53

17.两个乒乓球队进行比赛，各出 3 人。甲队为 A、B、C 3 人，乙队为 X、Y、Z 3 人。已抽签决定比赛名单。有人向队员打听比赛的名单。A 说他不和 X 比，C 说他不和 X、Y 比。请编程找出 3 对赛手的名单。

```
#include <stdio.h>
int main()
{
   char i,j,k;
   for(i='x';i<='z';i++)
      for(j='x';j<='z';j++)
         if(i!=j)
            for(k='x';k<='z';k++)
               if(i!=k&&j!=k)
                  if(i!='x'&&k!='x'&&k!='z')
                     printf("A--%c\nB--%c\nC--%c\n",i,j,k);
   return 0;
}
```

评：又是用最复杂最笨拙的方法解决最简单的问题
这道题小学生心算都能答出来
描述计算过程并也不复杂
根本没必要搞成 6 层循环与条件语句的嵌套
这个代码的风格也极烂

第 6 章

P54

1.用筛选法求 100 之内的素数。

评：自打中国人民知道陈景润起就知道了数学里有个筛法
筛法这个名字精简且信息充分
并且早已为大家所接受
老谭非要整出个“筛选法”
好像不这样就显不出“大师”具有独创性似的

解：所谓“筛选法”指的是“埃拉托色尼(Eratoshenes)筛法”。埃拉托色尼是古希腊的著名数学家。他采用的方法是，在一张纸上写上 1~1000 的全部整数，然后逐个判断它们是否素数，找出一个非素数，就把它挖掉，最后剩下的就是素数，见图 6.1。

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50……

图 6.1

评：“求 100 之内的素数”，却要“在一张纸上写上 1~1000 的全部整数”
而且要“逐个判断”（那还是筛法吗？）
著名数学家的智商居然比小学生还要低
这是一个奇迹
不管你们信不信，反正我信

（2）用 2 除它后面的各个数，把能被 2 整除的数挖掉，即把 2 的倍数挖掉。
（3）用 3 除它后面各数，把 3 的倍数挖掉。

评：“把能被 2 整除的数挖掉，即把 2 的倍数挖掉”没有问题
但“用 2 除它后面的各个数”就太 2 了，
Eratosthenes 还不至于如此愚蠢
“用 3 除它后面各数”同样愚蠢

上面的算法可表示为：

(1)挖去 1；

评：这没什么问题，尽管也要看怎么挖

(2)用下一个未被挖去的数 p 除 p 后面的各数，把 p 的倍数挖掉；

评：“把 p 的倍数挖掉”是对的
但使用的手段——“除 p 后面的各数”则是极其愚蠢的。这种做法根本就能不称其为筛法

(3)检查 p 是否小于 √n 的整数部分(如果 n=1000,则检查 p<31 是否成立)，如果是，则返回(2)继续执行，否则就结束。

(4)……

评：这里的主要错误是次序，这个步骤应该在(2)的前面而不是在它后面

这次高铁追尾就有这么个特点，发车的次序反了

（4）分别用 4,5……各数作为除数除这些数以后的各数。这个过程一直进行到在除数后面的数全部被挖掉为止。例如在图 6.1 中找 1~50 之间的素数，要一直进行到除数为 47 为止。事实上，可以简化，如果需要找 1~n 范围内的素数，只须进行到除数为 √n(取其整数)即可，例如对 1~50，只须进行到将 √7 作为除数即可。请读者思考为什么？

评：4 已经被你“挖掉”了，怎么做除数？

至于“为什么”“对 1~50，只须进行到将 √7 作为除数即可”

我思考的结果是：

没有最蠢，只有更蠢

1.用筛选法求 100 之内的素数。

```
#include <stdio.h>
#include <math.h>                    //程序中用到求平方根函数 sqrt
int main( )
{int i,j,n,a[101];                   //定义 a 数组包含 101 个元素
   for(i=1;i<=100;i++)               //a[0]不用，只用 a[1]~a[100]
        a[i]=i;                      //使 a[1]~a[100]的值为 1~100
   a[1]=0;                           //先“挖掉”a[1]
   for(i=2;i<sqrt(100);i++)
       for(j=i+1;j<=100;j++)
           {if(a[i]!=0&&a[j]!=0)
                if(a[j]%a[i]==0)
                  a[j]=0;            //把非素数“挖掉”
           }
   printf("\n");
   for(i=2,n=0;i<=100;i++)
     { if(a[i]!=0)                   //选出值不为 0 的数组元素，即素数
           {printf("%5d",a[i]);      //输出素数，宽度为 5 列
            n++;                     //累计本行已输出的数据个数
           }
        if(n==10)
           {printf("\n");
            n=0;
           }
     }
   printf("\n");
   return 0;
}
```

评：第 4 行：

int i,j,n,a[101];　　//定义 a 数组包含 101 个元素

这个在合格的程序员看来绝对是自甘堕落的写法
但初学者对这种写法通常并不能引起什么反感
正规的写法应该是

```
int i,j,n,a[100];

第 5~6 行：
   for(i=1;i<=100;i++)                    //a[0]不用，只用 a[1]~a[100]
         a[ i ]=i;                                                  //使 a[1]~a[100]的值为 1~100
```

这是继续堕落
“a[0]不用”，“不用”你定义它作甚？典型的“豆浆要两碗，喝一碗，倒一碗”
应该

```
   for(i=0;i<100;i++)
         a[ i ]=i+1;
```

第 7 行：

```
a[1]=0;
```

错倒没有，写得 too naive

第 8 行：

```
for(i=2;i<sqrt(100);i++)
```

这行错的非常离谱
首先，表达式 i<sqrt(100)中的<其实应该是<=,(把题目中的 100 换成 121 很容易发现这个马脚)
其次它的语意是每次循环都毫无意义地调用一下 sqrt()，效率底下
（当然有的编译器可能会对此进行优化，但这不表明代码不垃圾）
最后没人保证 i<sqrt(100)等价于 i<10
第 9~13 行：

```
for(j=i+1;j<=100;j++)
         {if(a[i]!=0&&a[j]!=0)
                  if(a[j]%a[i]==0)
                     a[j]=0;                    //把非素数“挖掉”
            }
```

这里的 j++和 a[j]%a[i]极其笨拙
完全不是筛法
筛法并不需要逐个检验
也根本不需要做费时的求余运算

即使按照原来不合理的数据结构

第 8~13 行也应该写成

```
for(i=2 ; i*i<=100;i++)
         if(a[i]!=0)
             for(j=i+a[i];j<=100;j+=a[i])
                    a[j]=0;
```

才称得上是筛法
可以看到这里 j 并不是每次加 1
而且不需要费时的%运算
效率方面天壤之别

第 15~24 行：
这几行没有什么错误
只是写的很傻
那个很次要的

```
if(n==10)
   {printf("\n");
    n=0;
   }
```

很笨拙也很扎眼

P55~56

2.用选择法对 10 个整数排序。

```
#include <stdio.h>
int main( void )
{int i,j,min,temp,a[11];
   printf("enter data:\n");
   for(i=1;i<=10;i++)
      { printf("a[%d]=",i);
        scanf("%d",&a[i]);
      }
   printf("\n");
   printf("The orginal numbers:\n");
   for(i=1;i<=10;i++)
      printf("%5d",a[i]);
   printf("\n");
   for(i=1;i<=9;i++)
      { min=i;
        for(j=i+1;j<=10;j++)
           if(a[min]>a[j])min=j;
        temp=a[i];                          //以下 3 行将 a[i+1]~a[10]中最小值与 a[i]对换
        a[i]=a[min];
        a[min]=temp;
      }
   printf("\nThe sorted numbers:\n");
   for(i=1;i<=10;i++)
      printf("%5d",a[i]);
   printf("\n");
   return 0;
```

}

评：“以下 3 行将 a[i+1]~a[10]中最小值与 a[i]对换”
是错误的
实际上是将 a[i]~a[10]中最小值与 a[i]对换

需要一个 10 个元素的数组，却定义了 int a[11]，明显属于不上路子
for(i=1;i<=10;i++) 是半吊子写法

P56

3.求一个 3×3 的整型矩阵对角线元素之和。

评：这个题目本身就很成问题

因为一共有两条对角线

究竟是求那条对角线上的元素之和

还是求所有处于对角线上的元素之和是不明确的

```
int a[3][3],sum=0;
    ……
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
          scanf("%3d",&a[ i ][j]);
    ……
```

评：那个 3 明显是愚蠢的作茧自缚

P57

如果将程序中的第 7~9 行改为

```
for(j=0;j<3;j++)
    scanf("%d%d%d",&a[0][j],&a[1][j],&a[2][j]);
```

应如何输入？……

答案是可以按此方式输入，也可以不按此方式输入，而采用前面介绍的方式输入……

评：误导。这行代码和所替代的 1601 楼的代码功能完全不同

而且谭在这里的讨论毫无意义，因为这属于 scanf()的用法，和数组没有关系

p57~58

4.有一个已排好序的数组，要求输入一个数后，按原来排序的规律将它插入数组中。

```
#include <stdio.h>
int main( )
{ int a[11]={1,4,6,9,13,15,19,18,40,100};
   int temp1,temp2,number,end,i,j;
   printf("array a:\n");
   for(i=0;i<10;i++)
      printf("%5d",a[ i ]);
   printf("\n");
   printf("insert data:");
   scanf("%d",&number);
   end=a[9];
   if(number>end)
      a[10]=number;
   else
        {for(i=0;i<10;i++)
           {if(a[ i ]>number)
               {temp1=a[ i ];
                a[ i ]=number;
                for(j=i+1;j<11;j++)
                     {temp2=a[j];
                       a[j]=temp1;
                       temp1=temp2;
                     }
               break;
               }
           }
        }
   printf("Now array a:\n");
   for(i=0;i<11;i++)
      printf("%5d",a[ i ]);
   printf("\n");
   return 0;
}
```

评：很简单的题目

如此笨拙的写法倒是第一次看到

最可笑的就是

```
end=a[9];
if(number>end)
```

为什么不直接用 number 与 a[9]比较呢

定义 end 这个变量并对其赋值极其无聊

```
{for(i=0;i<10;i++)
        {if(a[ i ]>number)
            {temp1=a[ i ];
             a[ i ]=number;
             for(j=i+1;j<11;j++)
                 {temp2=a[j];
                  a[j]=temp1;
                  temp1=temp2;
                 }
            break;
            }
        }
     }
```

评：笨拙且复杂的令人拍案惊奇

是我所见到过的最复杂的思路不清

p61

7.输出“魔方阵”。所谓魔方阵是指这样的方阵，它的每一行、每一列和对角线之和均相等。例如，三阶魔方阵为

8 1 6
3 5 7
4 9 2

要求输出 1~n*n 的自然数构成的魔方阵。

评：倒是很有兴趣见识一下老谭如何输出 2 阶魔方阵

p61

解：魔方阵中各数的排列规律如下：

(1)将 1 放在第 1 行的中间一列。

【因惨不忍睹，下面删去 XXXX 个字】

评：2 阶魔方阵的中间一列是哪列？

p61~62

```
#include <stdio.h>
int main()
{ int a[15][15],i,j,k,p,n;
   p=1;
   while(p==1)
      {printf("Enter n(1--15):");//要求阶数为 1~15 之间的奇数
       scanf("%d",&n);
       if((n!=0)&&(n<=15)&&(n%2!=0))
          p=0;
      }
//初始化
   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       a[i][j]=0;+++
//建立魔方阵
   j=n/2+1;
   a[1][j]=1;
   for(k=2;k<=n*n;k++)
      {i=i-1;
       j=j+1;
```

```
        if((i<1)&&(j>n))
           {i=i+2;
            j=j-1;
           }
        else
           {if(i<1) i=n;
            if(j>n) j=1;
           }
        if(a[i][j]==0)
           a[i][j]=k;
        else
           {i=i+2;
            j=j-1;
            a[i][j]=k;
           }
      }
//输出魔方阵
   for(i=1;i<=n;i++)
      {for(j=1;j<=n;j++)
            printf("%4d",a[i][j]);
       printf("\n");
      }
   return 0;
}
```

其中一段：

```
p=1;
   while(p==1)
      {printf("Enter n(1--15):");//要求阶数为 1~15 之间的奇数
       scanf("%d",&n);
       if((n!=0)&&(n<=15)&&(n%2!=0))
          p=0;
      }
```

评：这段的功能貌似是输入一个 1~15 之间的奇数给 n（但题目中根本就没提到 n 应该是奇数）

写得啰嗦笨拙不说

压根就给写错了

这里就不详细解释错在哪里了

看不懂错在哪里的童鞋请举手

```
//初始化
   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       a[i][j]=0;+++
```

评：这段错的更是无法无天
i=1,j=1
i<=n , j<=n 越界
最后居然还有个 +++这连编译都通过不了
可笑的是老谭居然给出了运行结果
老谭给出的结果只可能是伪造的

```
//建立魔方阵
   j=n/2+1;
   a[1][j]=1;
   for(k=2;k<=n*n;k++)
      {i=i-1;
       j=j+1;
       if((i<1)&&(j>n))
          {i=i+2;
           j=j-1;
          }
       else
          {if(i<1) i=n;
           if(j>n) j=1;
          }
       if(a[i][j]==0)
          a[i][j]=k;
       else
          {i=i+2;
           j=j-1;
           a[i][j]=k;
          }
      }
```

评：如果看完这段代码你竟然没呕吐
那我算你狠！

```
//输出魔方阵
   for(i=1;i<=n;i++)
      {for(j=1;j<=n;j++)
            printf("%4d",a[i][j]);
       printf("\n");
      }
```

评：如果你天真地在前面输入 15
这段没让你死机就算你走运

代码后居然有运行结果（咋弄出来的呢）
接着下面一行小字
说明：魔方阵的阶数应为奇数。

啥叫坑爹啊
这就是

P63

8 找出一个二维数组中的鞍点，即该位置上的元素在该行上最大，在该列上最小。也可能没有鞍点。

评：这个题目本身就有毛病

自然语言中的“最大”、“最小”是一个含糊的说法

题目并没有明确这两个词的真正含义

解：一个二维数组最多只有一个鞍点，也可能没有。解题思路是：先找出一行中值最大的元素，然后检查它是否为该列的最小值，如果是，则是鞍点（不需要再找别的鞍点了），输出该鞍点；如果不是，则再找下一行的最大数……如果每一行的最大数都不是鞍点，则此数组无鞍点。

评：这个解题思路有着十分明显的逻辑漏洞

造成这个漏洞的原因就是它假定每行都存在最大值

```
#include <stdio.h>
#define N 4
#define M 5
int main()
{
   int i,j,k,a[N][M],max,maxj,flag;
   printf("please input matrix:\n");
   for(i=0;i<N;i++)
      for(j=0;j<M;j++)
         scanf("%d",&a[ i][j]);
   for(i=0;i<N;i++)
      {max=a[ i][0];
       maxj=0;
       for(j=0;j<M;j++)
          if(a[ i][j]>max)
             {max=a[ i][j];
              maxj=j;
             }
       flag=1;
       for(k=0;k<N;k++)
          if(max>a[k][maxj])
              {flag=0;
               continue;}
       if(flag)
         {printf("a[%d][%d]=%d\n",i,maxj,max);
          break;
         }
      }
   if(!flag)
       printf("It is not exist\n");
```

```
    return 0;
}
```

评：当输入

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

时

程序输出为：

a[0][0]=1

就是说

要么这个数组存在 20 个鞍点（注：这和“一个二维数组最多只有一个鞍点”相矛盾）它只能求出一个

要么这个数组不存在鞍点但它居然硬给求出来了一个

无论怎样都是错的

记得老谭的这本书最早是 91 年出的

20 年来居然连个鞍点都没找对

这真的是一个奇迹！ 

不管你信不信

反正我是信了

P65~66

```
#include <stdio.h>
#define N 15
int main()
{ int i,number,top,bott,mid,loca,a[N],flag,sign;
   char c;
   printf("enter data:\n");
   scanf("%d",&a[0]);
   i=1;
   while(i<N)
     {scanf("%d",&a[ i]);
      if(a[ i]>=a[i-1])
        i++;
      else
        printf("enter this data again:\n");
     }
   printf("\n");
   for(i=0;i<N;i++)
```

```
        printf("%5d",a[ i]);
    printf("\n");
    while(flag)
        {printf("input number to look for:");
         scanf("%d",&number);
         sign=0;
         top=0;
         bott=N-1;
         if((number<a[0])||(number>a[N-1]))
           loca=-1;
         while((!sign)&&(top<=bott))
           {mid=(bott+top)/2;
            if(number==a[mid])
               {loca=mid;
                printf("Has found%d,its position id %d:\n",number,loca+1);
                sign=1;
               }
            else if(number<a[mid])
               bott=mid-1;
            else
               top=mid+1;
           }
           if(!sign||loca==-1)
              printf("cannot find %d.\n",number);
           printf("continue or not(Y/N)?");
           scanf(" %c",&c);
           if(c=='N'||c=='n')
              flag=0;
        }
    return 0;
}
```

有 15 个数按由小到大顺序存放在一个数组中

评：表明对于程序来说这 15 个数构成的有序数组是一个已知的前提条件

但代码却擅自将其改为由用户输入

```
printf("enter data:\n");
    scanf("%d",&a[0]);
    i=1;
    while(i<N)
      {scanf("%d",&a[ i]);
       if(a[ i]>=a[i-1])
         i++;
       else
```

```
        printf("enter this data again:\n");
    }
  printf("\n");
```

评：表明代码作者要么根本就无视需求，要么根本就没读懂题目

这种情况

在小学生身上叫答非所问

在中学生身上叫不解题意

在软件工程叫曲解需求

在软件测试叫多做之过

```
while(flag)
    {……
       printf("continue or not(Y/N)?");
       scanf(" %c",&c);
       if(c=='N'||c=='n')
          flag=0;
    }
```

评：是另一处多做之过

因为题目根本就没有要求

题目的要求是“输入一个数”

谭书最大的危害不是把 C 讲错了

而是破坏摧毁了小朋友本来就单薄脆弱的逻辑思维能力

脑残是怎样炼成的？

就是这样炼成的

P67

10.有一篇文章，共有 3 行文字，每行有 80 个字符。要求分别统计出其中英文大写字母、小写字母、数字、空格以及其他字符的个数。

评：这叫什么烂题啊

根本就没有说清楚程序的输入

P68

```
#include <stdio.h>
int main()
{int i,j,upp,low,dig,spa,oth;
char text[3][80];
upp=low=dig=spa=oth=0;
for(i=0;i<3;i++)
    {printf("please input line %d:\n",i+1);
     gets(text[ i]);
```

```
        for(j=0;j<80&&text[ i][j]!='\0';j++)
          {if(text[ i][j]>='A'&&text[ i][j]<='Z')
              upp++;
           else if(text[ i][j]>='a'&&text[ i][j]<='z')
              low++;
           else if(text[ i][j]>='0'&&text[ i][j]<='9')
              dig++;
           else if(text[ i][j]==' ')
              spa++;
           else
              oth++;
          }
      }
         printf("\nupper case:%d:\n",upp);
         printf("lower case:%d:\n",low);
         printf("digit          :%d:\n",dig);
         printf("space          :%d:\n",spa);
         printf("other          :%d:\n",oth);
    return 0;
}
```

评：并非是“一篇文章”，也没有“3 行文字”，更谈不上“每行有 80 个字符”

for(j=0;j<80&&text[i][j]!='\0';j++)中的
j<80&&text[i][j]!='\0'　是滑稽可笑的

运行结果：

```
please input line 1:
I am a student.
please input line 2:
123456
please input line 3:
ASDFG

upper case:6:
lower case:10:
digit          :6:
space          :3:
other          :1:
```

评：不是有 3 行文字吗？
那 other 至少也应该是 2 啊(因为至少要有 2 个换行符)
怎么可能是 1 呢

此外在同一个 for 语句中完成输入和统计
char text[3][80];

就成了脱裤子放屁
因为只要
char text[80];
就完全可以了

P69

11.输出以下图案：

```
     *****
        *****
           *****
              *****
                 *****
```

解：程序如下：

```
#include <stdio.h>
int main()
{ char a[5]={'*','*','*','*','*'};
   int i,j,k;
   char space=' ';
   for(i=0;i<5;i++)
     {printf("\n");
      printf("      ");
      for(j=1;j<=i;j++)
         printf("%c",space);
      for(k=0;k<=i;k++)
         printf("%c",a[k]);
     }
   printf("\n");
   return 0;
}
```

评：没把人给累死也把人给笨死了

12.有一行电文，已按下面规律译成密码：

```
   A-Z a-z
    B-Y b-y
    C-X c-x
      ……
```

即第 1 个字母变成第 26 个字母，第 i 个字母变成第(26-i+1)个字母,非字母字符不变。要求编程序将密码译回原文，并输出密码和原文。

评：比较恶心的地方有：
两种几乎一模一样的代码（区别仅仅在于第一种多定义了一个本可以不定义的数组）
tran[j]=155-ch[j];
tran[j]=219-ch[j];
ch[j]=155-ch[j];

ch[j]=219-ch[j];
最恶心的一句是
ch[j]=ch[j];

p71

13.编一程序，将两个字符串连接起来，不要用 strcat 函数。

```
#include <stdio.h>
int main()
{ char s1[80],s2[40];
   int i=0,j=0;
   printf("input string1:");
   scanf("%s",s1);
   printf("input string2:");
   scanf("%s",s2);
   while(s1[ i]!='\0')
        i++;
   while(s2[j]!='\0')
        s1[ i++]=s2[j++];
   s1[ i]='\0';
   printf("\nThe new string is:%s\n",s1);
   return 0;
}
```

评：基本写对了
难得

```
   while(s2[j]!='\0')
        s1[ i++]=s2[j++];
   s1[ i]='\0';
```

应为

```
while( ( s1[i++]=s2[j++]) !='\0')
          ;
```

P71~72

14.编一个程序，将两个字符串 s1 和 s2 比较，若 s1>s2,输出一个正数；若 s1=s2，输出 0；若 s1<s2，输出一个负数。不要用 strcpy 函数。……

```
#include <stdio.h>
int main()
{ int i,resu;
   char s1[100],s2[100];
   printf("input string1:");
   gets(s1);
   printf("\ninput string2:");
   gets(s2);
   i=0;
   while((s1[ i]==s2[ i])&&(s1[ i]!='\0'))i++;
   if(s1[ i]=='\0'&&s2[ i]=='\0')
        resu=0;
   else
        resu=s1[ i]-s2[ i];
   printf("\nserult:%d.\n",resu);
   return 0;
}
```

评：if(s1[i]=='\0'&&s2[i]=='\0')
resu=0;
else
resu=s1[i]-s2[i];
非常滑稽

“不要用 strcpy 函数”
也很滑稽
比较字符串和 strcpy 有什么关系

p72

15.编写一个程序，将字符数组 s2 中的全部字符复制到字符数组 s1 中，不用 strcpy 函数。复制时，'\0'也要复制过去。'\0'后面的字符不复制。

评：话都说不明白

颠三倒四

开始说“全部字符”

后来又“'\0'后面的字符不复制”

自相矛盾

不就是复制字符串吗

p72~73

```
#include <stdio.h>
#include <string.h>
int main()
{ char s1[80],s2[80];
   int i;
   printf("input s2:");
   scanf("%s",s2);
   for(i=0;i<=strlen(s2);i++)
      s1[ i]=s2[ i];
   printf("s1:%s\n",s1);
   return 0;
}
```

评：还不如调用 strcpy 函数

P74

写两个函数，分别求两个整数的最大公约数和最小公倍数，用主函数调用这两个函数，并输出结果。两个整数由键盘输入。

评：题目本身就有问题

求两个整数的最大公约数和最小公倍数

本身若不是一个错误的问题就是一个问题的错误提法

P74~75

方法一：

```
#include <stdio.h>
int main()
{int hcf(int,int);
int lcd(int,int,int);
int u,v,h,l;
scanf("%d,%d",&u,&v);
h=hcf(u,v);
printf("H.C.F=%d\n",h);
l=lcd(u,v,h);
printf("L.C.D=%d\n",l);
return 0;
}

int hcf(int u,int v)
{int t,r;
if(v>u)
    {t=u;u=v;v=t;}
while((r=u%v)!=0)
    {u=v;
     v=r;}
return(v);
}

int lcd(int u,int v,int h)
{
    return(u*v/h);
}
```

评：首先把函数类型声明挤在 main()中是一种拙劣的做法

其次，用 l 作为变量名违反了一般性的编程忌讳，况且 l 这个变量本来就是多余的

```
if(v>u)
{t=u;u=v;v=t;}
```

是废话，根本不需要

```
return(u*v/h);
```

是蠢话，不经意间缩小了函数的适用范围

恶劣的风格：

```
{int hcf(int,int);
        {int t,r;
        {t=u;u=v;v=t;}
        {u=v;
        v=r;}
```

赤裸的 scanf("%d,%d",&u,&v);及“,”

最荒唐的是

```
int lcd(int,int,int);
```

问题只提供两个数据

可这个函数竟然有三个参数

这样的函数毫无意义，是废品

```
#include <stdio.h>

unsigned gcd(unsigned,unsigned);
unsigned lcm(unsigned,unsigned);

int main( void )
{
int u,v;

printf("输入两个正整数:");
scanf("%d%d",&u,&v);

if ( u <= 0 || v <= 0 ){
            printf("无法计算\n");
            return 1;
        }

printf("最大公约数为=%u\n" , gcd(u,v) );
printf("最小公倍数为=%u\n" , lcm(u,v) );

return 0;
}

unsigned gcd(unsigned u,unsigned v)
{
```

```
    unsigned r;
    while((r=u%v)!=0){
            u = v ;
            v = r ;
        }
    return v;
    }

    unsigned lcm(unsigned u,unsigned v)
    {
        return u / gcd(u,v) * v ;
    }
```

P75~76

方法二：

```
#include <stdio.h>
int Hcf,Lcd;
int main()
{void hcf(int,int);
void lcd(int,int);
int u,v;
scanf("%d,%d",&u,&v);
hcf(u,v);
printf("H.C.F=%d\n",Hcf);
lcd(u,v);
printf("L.C.D=%d\n",Lcd);
return 0;
}

void hcf(int u,int v)
{int t,r;
if(v>u)
    {t=u;u=v;v=t;}
while((r=u%v)!=0)
    {u=v;
     v=r;
    }
Hcf=v;
}

void lcd(int u,int v)
```

```
{
    Lcd=u*v/Hcf ;
}
```

评：这个写法比 2130 楼的“方法一”更蠢
外部变量用得非常恶劣，有百害而无一利
甚至，它根本不满足题目要求
这是在教唆初学者学坏

P76

求方程 ax^2+bx+c=0 的根，用 3 个函数分别求当：b^2-4ac 大于 0、等于 0 和小于 0 时的根并输出结果。从主函数输入 a,b,c 的值。
评：题目要求有点怪异

P76~77

```
#include <stdio.h>
#include <math.h>
float x1,x2,disc,p,q;
int main()
{void greater_than_zero(float,float);
   void equal_to_zero(float,float);
   void smaller_than_zero(float,float);
   float a,b,c;
   printf("input a,b,c:");
   scanf("%f,%f,%f",&a,&b,&c);
   printf("equation:%5.2f*x*x+%5.2f*x+%5.2f=0\n",a,b,c);
   disc=b*b-4*a*c;
   printf("root:\n");
   if(disc>0)
     {
        greater_than_zero(a,b);
        printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else if(disc==0)
     {equal_to_zero(a,b);
      printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else
     {smaller_than_zero(a,b);
        printf("x1=%f+%fi\tx2=%f-%fi\n",p,q,p,q);
```

```
    }
  return 0;
}

void greater_than_zero(float a,float b)
{x1=(-b+sqrt(disc))/(2*a);
  x2=(-b-sqrt(disc))/(2*a);
}

void equal_to_zero(float a,float b)
{
  x1=x2=(-b)/(2*a);
}

void smaller_than_zero(float a,float b)
{
  p=-b/(2*a);
  q=sqrt(-disc)/(2*a);
}
```

评：垃圾得简直难以评说

有句名言，再好的程序设计语言也挡不住有人写出垃圾代码。

2146 楼的代码完美地诠释了这一点

本来，函数是实施结构化程序设计的利器

但 2146 楼的代码却使得函数成了破坏或违反结构化程序设计的利器

而帮凶就是不伦不类的外部变量

评：垃圾得简直难以评说

首先

void greater_than_zero(float a,float b)

这个函数有两个参数迹近胡扯

因为解这个方程只两个参数根本就不充分

其次

看不出把 float x1,x2,disc,p,q;作为外部变量

而把 float a,b,c;作为局部变量的任何理由

如果暂时不考虑外部变量的危害

倒不如把 a,b,c 同样作为外部变量

下面的代码比原来要强很多

说明了

原来把 float x1,x2,disc,p,q;作为外部变量，而把 float a,b,c;作为局部变量，完全是一种稀里糊涂的无厘头之举

是一种没有经过大脑的行为

因而也没有任何有益的效果

```
#include <stdio.h>
#include <math.h>
void greater_than_zero(void);
void equal_to_zero(void);
void smaller_than_zero(void);

float x1,x2,disc,p,q;
float a,b,c;

int main(void)
{
   printf("input a,b,c:");
   scanf("%f,%f,%f",&a,&b,&c);

   printf("equation:%5.2f*x*x+%5.2f*x+%5.2f=0\n",a,b,c);
   disc=b*b-4*a*c;

   printf("root:\n");
   if(disc>0)
     {
        greater_than_zero();
        printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else if(disc==0)
     {equal_to_zero();
      printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else
     {smaller_than_zero();
        printf("x1=%f+%fi\tx2=%f-%fi\n",p,q,p,q);
     }
   return 0;
}

void greater_than_zero(void)
{x1=(-b+sqrt(disc))/(2*a);
x2=(-b-sqrt(disc))/(2*a);
}

void equal_to_zero(void)
{
   x1=x2=(-b)/(2*a);
}
```

```
void smaller_than_zero(void)
{
   p=-b/(2*a);
   q=sqrt(-disc)/(2*a);
}
```

是的。math.h 提供的只是函数原型。C 语言确实把函数原型、宏、类型分别写在不同的.h 中，但这不意味着要求编译器分别提供若干个库

我的印象，谭书没有正式介绍过库的概念，在这方面，老谭一向是喜欢用空洞的概念搪塞的

而且，即使在库中，也谈不上“定义”，因为库是已经编译过了的东西，而“定义”则是代码层面的概念

从此不难看出那几个函数非常垃圾
同时外部变量也没有任何意义
不使用这两者
代码反而更好——至少更简洁

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

int main(void)
{
   float x1,x2,disc,p,q;
   float a,b,c;

   printf("input a,b,c:");
   scanf("%f,%f,%f",&a,&b,&c);

   printf("equation:%5.2f*x*x+%5.2f*x+%5.2f=0\n",a,b,c);
   disc=b*b-4*a*c;

   printf("root:\n");

   if(disc>0)
     {
        x1=(-b+sqrt(disc))/(2*a);
        x2=(-b-sqrt(disc))/(2*a);
        printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else if(disc==0)
     {
        x1=x2=(-b)/(2*a);
        printf("x1=%f\t\tx2=%f\n",x1,x2);
```

```
    }
  else
    {
      p=-b/(2*a);
      q=sqrt(-disc)/(2*a);
      printf("x1=%f+%fi\tx2=%f-%fi\n",p,q,p,q);
    }
  return 0;
}
```

同时展示一下自己的垃圾代码供初学者临摹？

这个题目的正确提法应该是用函数求解方程并输出

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

void solve(double,double,double);

int main(void)
{
      double x1, x2, disc, p, q;
      double a, b, c;

      printf("input a,b,c:");
      scanf("%lf%lf%lf", &a, &b, &c);

      printf("equation:%5.2f*x*x+%5.2f*x+%5.2f=0\n", a, b, c);
      printf("root:\n");
      solve(a,b,c);

      return 0;
}

void solve( double a , double b , double c )
{
    double disc , p , q ;

    disc = b * b - 4. * a * c ;
    p = -b / (2. * a) ;

    if ( disc >= 0.0 ){
        q = sqrt( disc )    / ( 2. * a );
        printf("x1=%f\tx2=%f\n", p + q , p - q );
    }
```

```
        else {
            q = sqrt( -disc ) / (2 * a);
            printf("x1=%f+%fi\tx2=%f-%fi\n", p, q, p, q);
        }
    }
```

P77

写一个判素数的函数，在主函数输入一个整数，输出是否为素数的信息。
评：有个初学者前几天发现这个题目的解答有错误

P77~78

```
#include <stdio.h>
int main ()
{int prime (int);
   int n;
   printf ("input an integer:");
   scanf ("%d",&n);
   if (prime(n))
      printf ("%d is a prime. \n",n);
   else
      printf ("%d is not a prime. \n",n);
   return 0;
}
int prime (int n )
{int flag=1,i;
   for (i=2;i<n/2&&flag==1;i++)
      if (n%i==0)
         flag=0;
   return (flag);
}
```

评：正如 huangzhenfan 这名初学者所指出的那样
按照这个程序
4 也是素数

非但这样
其实按照这个程序

1 也是素数

代码中的 flag 用得非常愚蠢
完全没有必要
而且既然题目中所说的是“整数”
应该考虑排除负整数的情况
这有两种写法
一种是由调用者排除
另一种是由函数判断
下面代码是后一种写法
但在工程实践中前一种写法可能更为常见

```
#include <stdio.h>

#define TRUE    1
#define FALSE 0

int prime (int);

int main ( void )
{
   int n;

   printf ("输入一个整数:");
   scanf ("%d",&n);

   printf ("%d%s 是素数。\n" , n , prime (n)?"":"不" ) ;
   return 0;
}

int prime ( int n )
{
   int i;

   if ( n <= 1 )
         return FALSE;

   for    ( i = 2 ; i <= n / 2 ; i++ )
            if    (    n % i == 0 )
                  return FALSE;

   return    TRUE ;
}
```

P78~79

4.写一个函数，使给定的一个 3*3 的二维整型数组转置，即行列互换。

```
#include <stdio.h>
#define N 3
int array[N][N];
int main()
{ void convert(int array[][3]);
int i,j;
   printf("input array:\n");
   for(i=0;i<N;i++)
      for(j=0;j<N;j++)
           scanf("%d",&array[i][j]);
   printf("\noriginal array:\n");
   for(i=0;i<N;i++)
      {for(j=0;j<N;j++)
        printf("%5d",array[i][j]);
        printf("\n");
      }
   convert(array);
   printf("\nconvert array:\n");
   for(i=0;i<N;i++)
      {for(j=0;j<N;j++)
        printf("%5d",array[i][j]);
        printf("\n");
      }
   return 0;
}
void convert(int array[][3])
{int i,j,t;
   for(i=0;i<N;i++)
      for(j=i+1;j<N;j++)
      {t=array[i][j];
        array[i][j]=array[j][i];
        array[j][i]=t;
      }
}
```

评：这段代码的愚蠢体现在

定义了一个外部数组

而 void convert(int array[][3])居然同时使用了参数

#define N 3

```
……
void convert(int array[][3])
{int i,j,t;
   for(i=0;i<N;i++)
      for(j=i+1;j<N;j++)
      {t=array[i][j];
       array[i][j]=array[j][i];
       array[j][i]=t;
      }
}
```

函数里的两个 N 非常滑稽
同时使得形参中的那个 3 显得很荒唐
此外这个函数的参数不完整

更荒唐的是
2167 楼的代码中居然出现了两段完全一模一样的代码（输出数组）
这种代码出现在函数这章
简直是一种讽刺

P79

写一个函数，使输入的一个字符串按反序存放，在主函数中输入和输出字符串。

```
#include <stdio.h>
#include <string.h>
int main()
   { void inverse(char str[]);
      char str[100];
      printf("input string:");
      scanf("%s",str);
      inverse(str);
      printf("inverse string:%s\n",str);
      return 0;
   }
void inverse(char str[])
{char t;
   int i,j;
   for(i=0,j=strlen(str);i<(strlen(str)/2);i++,j--)
   {t=str[ i];
      str[ i]=str[j-1];
      str[j-1]=t;
   }
}
```

评：1.求字符串长度是件很简单的事情，j=strlen(str)完全没有必要，作为练习就更不应该
2.i<(strlen(str)/2)很愚蠢，非但不必而且费时
3.str[j-1]也很蹩脚，之所以如此蹩脚是因为一开始 j 的初始值就是错误的

这个题目可以简单地写为

```
#include <stdio.h>

void inverse(char str[]);

int main( void )
{
   char str[100];

   puts("input string:");
   gets(str);

   inverse(str);

   puts("inversed string:");
   puts(str);

   return 0;
}

void inverse(char str[])
{
   int i = 0 ,j = 0;

   while(str[j]!='\0')
        j ++ ;

   j -- ;                        //'\0'之前的位置

   while ( i < j )
      {
         char t;
         t = str[ i];
         str[i++] = str[j];
         str[j--] = t;
      }
}
```

P80

6.写一个函数，将两个字符串连接。

```
#include <stdio.h>
int main( void )
{ void concatenate(char string1[],char string2[],char string[]);
   char s1[100],s2[100],s[100];
   printf("input string1:");
   scanf("%s",s1);
   printf("input string2:");
   scanf("%s",s2);
   concatenate(s1,s2,s);
   printf("\nThe new string is %s\n",s);
   return 0;
}

void concatenate(char string1[],char string2[],char string[])
{ int i , j ;
   for(i=0;string1[i]!='\0';i++)
         string[i]=string1[i];
   for(j=0;string2[j]!='\0';j++)
         string[i+j]=string2[j];
   string[i+j]='\0';
}
```

评：1.在 C 语言中 concatenate 通常指把一个字符串接到另一个字符串后面，这个代码对连接的含义有误导之嫌，s 这个字符数组没有必要

2.把 string 作为最后一个参数也不符合常规

3.

```
for(j=0;string2[j]!='\0';j++)
      string[i+j]=string2[j];
```

很蹩脚，不如

```
for(j=0;string2[j]!='\0';j++,i++)
      string[ i]=string2[j];
```

这个函数应该这样写

```
#include <stdio.h>

void concatenate(char [] , const char []);

int main( void )
{
   char s1[100],s2[100];
```

```
    puts("input string1:");
    gets(s1);
    puts("input string2:");
    gets(s2);

    concatenate(s1,s2);

    puts("\nThe new string is ");
    puts(s1);

    return 0;
}

void concatenate(char str1[] , const char str2[])
{
    int i = 0 , j = 0 ;

    while(str1[i]!='\0')
        i ++ ;

    while((str1[i++]=str2[j++])!='\0')
        ;
}
```

写一个函数，将一个字符串中的元音字母复制到另一字符串，然后输出。

……

```
if(s[ i]=='a'||s[ i]=='A'||s[ i]=='e'||s[ i]=='E'||s[ i]=='i'||s[ i]=='I'||s[ i]==
    'o'||s[ i]=='O'||s[ i]=='u'||s[ i]=='U')
```

……

评：这段代码最可圈可点的就是这个罕见的、巨长无比的表达式
要是复制的辅音字母恐怕就更壮观无比了
我认为代码长一点没什么，良好的结构才是最重要的

P81

8.写一个函数，输入一个 4 位数字，要求输出这 4 个数字字符，但每两个数字之间空一个空格。如输入 1990，应输出“1 9 9 0”

评：“一个 4 位数字”？天知道这是什么东西
要求也极其无聊

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

```
#include <string.h>
int main( void )
{void insert(char []);
char str[80];
printf("input four digit:");
scanf("%s",str);
insert(str);
return 0;
}
void insert(char str[])
{int i;
for(i=strlen(str);i>0;i--)
   {str[2*i]=str[i];
    str[2*i-1]=' ';
   }
printf("output:\n%s\n",str);
}
```

评：1.函数要求能输入，insert()并没有实现这个要求
2.这题跟数字不数字的毫无关系，题目要求本身画蛇添足
3.不但如此，代码完成的方式同样画蛇添足，与题目配成了绝代双足

实际上只要简单地

```
#include <stdio.h>

void whatever( void );

int main( void )
{
   whatever();
   return 0;
}

void whatever( void )
{
   int i ;
   for( i = 0 ; i < 4 ; i++ )
   {
    putchar(getchar());
    putchar(' ');
   }
}
```

9.编写一个函数，由实参传来一个字符串，统计此字符串中字母、数字、空格和其他字符的个数，在主函数中输入字符串以及输出上述的结果。

评：就谭书的结构安排和进度来说

在主函数中输出结果这个要求基本上是一个逼良为娼的要求

况且，即使从老谭的代码（2191 楼）中，也搞不清楚题目中提到的“输入一个 4 位数字，要求输出这 4 个数字字符，但每两个数字之间空一个空格”究竟有什么意义，完全是废话

P82

```
#include <stdio.h>
int letter,digit,space,others;
int main()
{void count(char []);
char text[80];
printf("input string:\n");
gets(text);
printf("string:");
puts(text);
letter=0;
digit=0;
space=0;
others=0;
count(text);
printf("\nletter:%d\ndigit:%d\nspace:%d\nothers:%d\n",letter,digit,space,others);
return 0;
}
void count(char str[])
{int i;
for(i=0;str[i]!='\0';i++)
if((str[i]>='a'&&str[i]<='z')||(str[i]>='A'&&str[i]<='Z'))
    letter++;
else if(str[i]>='0'&&str[i]<='9')
    digit++;
else if(str[i]== 32)
    space++;
else
    others++;
}
```

评：果然，使用了丑陋无比的外部变量。这就是逼良为娼几乎必然的后果

其他可吐槽的也很多

外部变量使得 count()函数的参数变得很傻
还不如把 char text[80];也作为外部变量
这样还免得传递参数

第 10~13 行的赋值更蠢
因为外部变量的初值本来就是 0
如果把这几个清零看成必要的初始化
它们仍然愚蠢
因为这几个清零根本就不该在 main()中进行
如果 main()还需要统计若干字符串的话
难道再写几次清零的语句？
这几个清零的语句正确的位置应该在 count()函数中

str[i]== 32，不具备可移植性，可读性也很差，应该 str[i]== ' '

count()函数中 for 语句与 if 语句齐头并进，风格极其垃圾

P82~84

10.写一个函数，输入一行字符，将此字符串中最长的单词输出。

```
#include <stdio.h>
#include <string.h>
int main()
{int alphabetic(char);
int longest(char []);
int i;
char line[100];
printf("input one line:\n");
gets(line);
printf("The longest word is:");
for(i=longest(line);alphabetic(line[i]);i++)
      printf("%c",line[i]);
printf("\n");
return 0;
}

int alphabetic(char c)
{
    if((c>='a'&&c<='z')||(c>='A'&&c<='z'))
       return(1);
    else
       return(0);
}
```

```
int longest(char string[])
{int len=0,i,length=0,flag=1,place=0,point;
for(i=0;i<=strlen(string);i++)
    if(alphabetic(string[i]))
     if(flag)
       {point=i;
        flag=0;
       }
      else
       len++;
    else
      {flag=1;
       if(len>=length)
          {length=len;
           place=point;
           len=0;
          }
      }
  return(place);
}
```

评：所谓"一个函数"不知道是指的哪个
alphabetic()和 longest()函数都没有实现题目要求

此外一个明显的 BUG 是
(c>='a'&&c<='z')||(c>='A'&&c<='z')

另外 alphabetic()写的也过于愚蠢,其实只要

```
Int alphabetic(char c)
{
      return          (c>='a'&&c<='z')
                  ||    (c>='A'&&c<='Z');
}
```

这个代码非但是逻辑混乱的，而且是错误的，甚至使用了垃圾值

P83

图 7.1
评：这个 NS 图是错误的
与 2244 楼的代码也根本对不上

P84

11.写一个函数，用“起泡法”对输入的 10 个字符按由小到大顺序排序。

评：如果一个函数只能对 10 个字符排序
这种函数几乎是没有任何意义的
按照这种愚昧的想法
对 2 个字符排序需要写一个函数
对 3 个字符排序需要另外再写一个函数
……
代码更荒谬

```
char str[N];
int main()
{
……
scanf("%s",&str);
……
}
```

P85

```
#include <stdio.h>
#include <string.h>
#define N 10
char str[N];
int main()
{void sort(char [ ]);
int i,flag;
for(flag=1;flag==1;)
   {scanf("%s",&str);
    if(strlen(str)>N)
      printf("string too long,input again!");
    else
      flag=0;
   }
sort(str);
printf("string sorted:\n");
for(i=0;i<N;i++)
    printf("%c",str[i]);
printf("\n");
return 0;
}
void sort(char str[ ])
{int i,j;
```

```
char t;
for(j=1;j<N;j++)
    for(i=0;(i<N-j)&&(str[i]!='\0');i++)
       if(str[i]>str[i+1])
          {t=str[i];
           str[i]=str[i+1];
           str[i+1]=t;
          }

}
```

评：有必要更正一下 2249 楼的说法
不是“代码更荒谬”
而是代码荒谬绝伦

```
for(flag=1;flag==1;)
   {scanf("%s",&str);
    if(strlen(str)>N)
      printf("string too long,input again!");
    else
      flag=0;
   }
```

flag，scanf("%s",&str);用得很蠢不说
费了这么大劲，代码还是错的

P87

输入 10 个学生 5 门课的成绩，分别用函数实现下列功能：
1.计算每个学生的平均分；
2.计算每门课的平均分；
3.找出所有 50 个分数中最高分数所对应的学生和课程；
4.计算平均分方差
评：“3.找出所有 50 个分数中最高分数所对应的学生和课程”

“所有”二字含义不清，属于题目本身的缺陷。题目没有说明存在并列最高分的情况应该如何处理

P88~90

```
#include <stdlib.h>
#define N 10
#define M 5
float score[N][M];
float a_stu[N],a_cour[M];
```

```
int r,c;
int main()
{int i,j;
float h;
float s_var(void);
float hightest( );
void input_stu(void);
void aver_stu(void);
void aver_cour(void);
input_stu();
aver_stu();
aver_cour();
printf("\n NO. cour1 cour2 cour3 cour4 cour5 aver\n");
for(i=0;i<N;i++)
    {printf("\n NO%2d",i+1);
     for(j=0;j<M;j++)
        printf("%8.2f",score[i][j]);
     printf("%8.2f\n",a_stu[i]);
    }
printf("\naverage:");
for(j=0;j<M;j++)
    printf("%8.2f",a_cour[j]);
printf("\n");
h=hightest( );
printf("hightest:%7.2f NO%2d court%2d\n",h,r,c);
printf("variance %8.2f\n",s_var());
return 0;
}

void input_stu(void)
{int i,j;
for(i=0;i<N;i++)
    {printf("\ninput score of student%2d:\n",i+1);
     for(j=0;j<M;j++)
        scanf("%f",&score[i][j]);
    }
}

void aver_stu(void)
{int i,j;
float s;
for(i=0;i<N;i++)
    {for(j=0,s=0;j<M;j++)
        s+=score[i][j];
```

```
        a_stu[i]=s/5.0;
    }
}

void aver_cour(void)
{int i,j;
float s;
for(j=0;j<M;j++)
    {s=0;
     for(i=0;i<N;i++)
        s+=score[i][j];
     a_cour[j]=s/(float)N;
    }
}

float hightest( )
{float high;
int i,j;
high=score[0][0];
for(i=0;i<N;i++)
    for(j=0;j<M;j++)
       if(score[i][j]>high)
          {high=score[i][j];
           r=i+1;
           c=j+1;
          }
return(high);
}

float s_var(void)
{int i;
float sumx,sumxn;
sumx=0.0;
sumxn=0.0;
for(i=0;i<N;i++)
    {sumx+=a_stu[i]*a_stu[i];
     sumxn+=a_stu[i];
    }
return(sumx/N-(sumxn/N)*(sumxn/N));
}
```

评：主要有这样一些毛病：

1.使用外部变量

2.float hightest( ); 过时且与其他函数类型声明风格不统一

3.main()中粗细不分，结构混乱。把 main()几乎弄成了一个专司输出的函数
4.几个函数没有参数，没有通用性，尤其是 s_var()，没有参数是无论怎样都说不过去的
5.aver_stu() 中的 5.0 来历不明
6.在 main()中声明函数对混乱起了令人无法轻视的作用
7.存在并列最高分时，只能输出一个

这种又臭又长的程序实在让人受不了
下不为例了

P91~92

15.写几个函数：
1.输入 10 个职工的姓名和职工号；
2.按职工号由小到大排序，姓名顺序也随之调整；
3 要求输入一个职工号，用折半法找出该职工的姓名，从主函数输入要查找的职工号，输出该职工姓名

```
#include <stdio.h>
#include <string.h>
#define N 10
int main( void )
{void input(int num[],char name[][8]);
void sort(int num[],char name[][8]);
void search(int n,int num[],char name[][8]);
int num[N],number,flag=1,c;
char name[N][8];
input(num,name);
sort(num,name);
while(flag==1)
    {printf("\ninput number to look for:");
     scanf("%d",&number);
     search(number,num,name);
     printf("continue ot not(Y/N)?");
     getchar();
     c=getchar();
     if(c=='N'||c=='n')
        flag=0;
    }
return 0;
}

void input(int num[],char name[N][8])
{int i;
for(i=0;i<N;i++)
   {printf("input NO.:");
    scanf("%d",&num[i]);
```

```
      printf("input name:");
      getchar( );
      gets(name[i]);
    }
}

void sort(int num[],char name[N][8])
{int i,j,min,temp1;
char temp2[8];
for(i=0;i<N-1;i++)
    {min=i;
      for(j=i;j<N;j++)
         if(num[min]>num[j])min=j;
      temp1=num[i];
      strcpy(temp2,name[i]);
      num[i]=num[min];
      strcpy(name[i],name[min]);
      num[min]=temp1;
      strcpy(name[min],temp2);
    }
printf("\n result:\n");
for(i=0;i<N;i++)
      printf("\n %5d%10s",num[i],name[i]);
}

void search(int n,int num[],char name[N][8])
{int top,bott,mid,loca,sign;
top=0;
bott=N-1;
loca=0;
sign=1;
if((n<num[0])||(n>num[N-1]))
      loca=-1;
while((sign==1)&&(top<=bott))
    {mid=(bott+top)/2;
      if(n==num[mid])
         {loca=mid;
           printf("No.%d,his name is %s.\n",n,name[loca]);
           sign=-1;
         }
      else if(n<num[mid])
           bott=mid-1;
      else
           top=mid+1;
```

```
    }
if(sign==1||loca==-1)
          printf("%d not been found.\n",n);
}

#include <stdio.h>
#define N 10
int main( void )
{
int num [N],number,flag=1,c;
char name[N][8];
/*……*/
while(flag==1)
       {printf("\ninput number to look for:");
        scanf("%d",&number);
        search(number,num,name);
        printf("continue ot not(Y/N)?");
        getchar();
        c=getchar();
        if(c=='N'||c=='n')
           flag=0;
       }
   /*……*/
   return 0;
}
```

评：又是 flag!

看到 flag 总能从代码中闻到一股馊味

很显然，不应该用 while 语句，而应该用 do-while 语句

```
do
{
    printf("\ninput number to look for:");
    scanf("%d",&number);
    search(number,num,name);
    printf("continue ot not(Y/N)?");
    getchar();
    c=getchar();
    if(c=='N'||c=='n')
          flag=0;
}
while(flag==1);
```

其次 flag 明显毫无必要

```
do
```

```
    {
        printf("\ninput number to look for:");
        scanf("%d",&number);
        search(number,num,name);
        printf("continue ot not(Y/N)?");
        getchar();
        c=getchar();
    }
    while(!(c=='N'||c=='n'));
```

getchar();用的牵强蹩脚，且容错性差

应该

```
    do{
        printf("\ninput number to look for:");
        scanf("%d",&number);
        search(number,num,name);
        printf("continue ot not(Y/N)?");
        scanf(" %c",&c );
    }
    while( c!='N' && c!='n' );
```

P92

```
void input(int num[],char name[N][8])
{int i;
for(i=0;i<N;i++)
   {printf("input NO.:");
    scanf("%d",&num[i]);
    printf("input name:");
    getchar( );
    gets(name[i]);
   }
}
```

评：首先，参数不全

```
    void input(int num[],char name[N][8])
```

这个 N 很垃圾，因为它毫无意义

```
    for(i=0;i<N;i++)
```

这个 N 莫名其妙，使得函数丧失通用性

没有通用性的函数是废品函数

```
        getchar( );
```

显得累赘多余，且效果可能并不干净
如果后面一定用 gets()
这里应该
whille(getchar()!='\n')
;

```
void sort(int num[],char name[N][8])
{int i,j,min,temp1;
char temp2[8];
for(i=0;i<N-1;i++)
   {min=i;
    for(j=i;j<N;j++)
       if(num[min]>num[j])min=j;
    temp1=num[i];
    strcpy(temp2,name[i]);
    num[i]=num[min];
    strcpy(name[i],name[min]);
    num[min]=temp1;
    strcpy(name[min],temp2);
   }
printf("\n result:\n");
for(i=0;i<N;i++)
    printf("\n %5d%10s",num[i],name[i]);
}
```

评：参数不全，以及在函数内使用无厘头的常数 N 的错误同 2276 楼

最严重的错误是
strcpy(name[i],name[min]);
是对 strcpy()函数的错用
是潜藏着的 BUG，可引起未定义行为
这里不可以用 strcpy()函数
只能用 memmove()

P93

```
void search(int n,int num[],char name[N][8])
{int top,bott,mid,loca,sign;
top=0;
bott=N-1;
loca=0;
sign=1;
if((n<num[0])||(n>num[N-1]))
```

```
    loca=-1;
while((sign==1)&&(top<=bott))
  {mid=(bott+top)/2;
   if(n==num[mid])
      {loca=mid;
       printf("No.%d,his name is %s.\n",n,name[loca]);
       sign=-1;
      }
   else if(n<num[mid])
       bott=mid-1;
   else
       top=mid+1;
  }
if(sign==1||loca==-1)
       printf("%d not been found.\n",n);
}
```

评：这部分主要问题是两个不必要的标志变量 loca,sign
首先
loca=0; 毫无道理，没有任何意义
倒不如
loca=-1;
这样，只有在找到的情况下这个值才会改变
后面的 if(sign==1||loca==-1)就可以简化为 if(loca==-1)

```
void search(int n,int num[],char name[N][8])
{int top,bott,mid,loca,sign;
top=0;
bott=N-1;
loca=-1;
sign=1;
if((n<num[0])||(n>num[N-1]))
    loca=-1;
while((sign==1)&&(top<=bott))
   {mid=(bott+top)/2;
    if(n==num[mid])
       {loca=mid;
        printf("No.%d,his name is %s.\n",n,name[loca]);
        sign=-1;
       }
    else if(n<num[mid])
        bott=mid-1;
    else
        top=mid+1;
```

```
    }
  if(loca==-1)
        printf("%d not been found.\n",n);
  }
```

P94

16.写一个函数，输入一个十六进制数，输出相应的十进制数。

评：题目可笑

下面写法应该满足要求吧

```
  void gun(void)
  {
      int i;
      printf("输入一个十六进制数:" ) ;
      scanf("%x",&i);
       printf("相应的十进制数为%d\n",i);
  }
```

P94~96

16.写一个函数，输入一个十六进制数，输出相应的十进制数。

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
int main()
{ int htoi(char s[]);
   int c,i,flag,flag1;
   char t[MAX];
   i=0;
   flag=0;
   flag1=1;
   printf("input a HEX number:");
   while((c=getchar())!='\0'&&i<MAX&&flag1)
    {if(c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F')
     {flag=1;
      t[i++]=c;
     }
     else if(flag)
     {t[i]='\0';
      printf("decimal number%d\n",htoi(t));
      printf("continue or not?");
      c=getchar();
```

```
        if(c=='N'||c=='n')
           flag1=0;
        else
           {flag=0;
            i=0;
            printf("\ninput a HEX number:");
           }
      }
    }
  return 0;
}
int htoi(char s[])
{ int i,n;
  n=0;
  for(i=0;s[i]!='\0';i++)
    {if(s[i]>='0'&&s[i]<='9')
       n=n*16+s[i]-'0';
     if(s[i]>='a'&&s[i]<='f')
       n=n*16+s[i]-'a'+10;
     if(s[i]>='A'&&s[i]<='F')
       n=n*16+s[i]-'A'+10;
    }
  return(n);
}
```

评：竟然能写出 (c=getchar())!='\0'

实在雷人

#define MAX 1000 很蠢

清楚地说明了什么叫心里没“数”

因为

char t[MAX];

htoi(t)

而 htoi()原型是

int htoi(char s[]);

flag1 在 main()中的主要作用有：

1.使代码变得混乱不堪

2.使代码显得重复啰嗦

3.表明作者虽然不会使用 break 语句也会写代码

因此删除 flag1

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
int htoi(char s[]);
```

```
int main()
{
   int c,i,flag;
   char t[MAX];
   i=0;
   flag=0;

   printf("input a HEX number:");
   while((c=getchar())!='\0'&&i<MAX)
   {
        if(c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F')
        {
            flag=1;
            t[i++]=c;
        }
        else
            if(flag)
            {
                t[i]='\0';
                printf("decimal number %d\n",htoi(t));
                printf("continue or not?");
                c=getchar();
                if(c=='N'||c=='n')
                    break;
                else
                {
                    flag=0;
                    i=0;
                    printf("\ninput a HEX number:");
                }
        }
   }
   return 0;
}
```

while((c=getchar())!='\0'&&i<MAX)

中，由于 i 的初值为 0，i 的变化只出现在 t[i++]=c; 之后，所以可以把这个条件移动到这条语句之后

(c=getchar())!='\0'是弱智才能想到写法，因为从键盘输入不可能使得 getchar()的值为 0,它起到的作用仅仅是 c=getchar()

虽然如此，还是暂时对此不予理睬而把它等价地移入循环体内

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
```

```
int htoi(char s[]);
int main()
{
    int c,i,flag;
    char t[MAX];
    i=0;
    flag=0;

    printf("input a HEX number:");
    while( 1 )
    {
        if(!((c=getchar())!='\0'))
        {
            break ;
        }
        if(c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F')
        {
            flag=1;
            t[i++]=c;
            if( !(i<MAX))
            {
                break ;
            }
        }
        else
            if(flag)
            {
                t[i]='\0';
                printf("decimal number%d\n",htoi(t));
                printf("continue or not?");
                c=getchar();
                if(c=='N'||c=='n')
                    break;
                else
                {
                    flag=0;
                    i=0;
                    printf("\ninput a HEX number:");
                }
        }
    }
    return 0;
}
```

现在总算可以看到这个晦涩不明的 flag 所起的作用了

1.如果一开始读入的不是十六进制数字字符，则产生一个死循环，直到输入十六进制数字字符

2.开始输入十六进制数字字符后，flag 的值为 1

3.在 2 之后，一旦输入非十六进制数字字符，则把这个字符改为'\0'写入字符数组，调用 htoi()。

4.之后要求用户输入 Y/N?如果是 N 则结束程序，否则重复 1

这些功能要求完全没必要使用晦涩不明的 flag

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
int htoi(char s[]);
int main()
{
    int c,i;
    char t[MAX];
    i=0;

    while( printf("input a HEX number:") )
    {
        do{                         //跳过非十六进制字符
            c=getchar();
            if(c=='\0')       //暂时保留原来代码中的愚蠢
                return 1;
        }
        while(!((c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F')));

        do{                         //读十六进制字符
            t[i++]=c;
            if( !(i<MAX))    //暂时保留原来代码中的愚蠢
                return 1; ;
            c=getchar();
        }
        while(c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F');

        t[i]='\0';
        printf("decimal number %d\n",htoi(t));    //输出转换结果

        printf("continue or not?");                    //询问是否继续
        c=getchar();
        if(c=='N'||c=='n')
            break;
        else
            i = 0;
```

```
        printf("\n");
    }

    return 0;
}
```

进一步简化

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
int htoi(char s[]);
int main()
{
    int c,i=0;
    char t[MAX];

    do
    {
        printf("input a HEX number:") ;
        do{                         //跳过非十六进制字符
            c=getchar();
            if(c=='\0')       //暂时保留原来代码中的愚蠢
                return 1;
        }
        while(!((c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F')));

        do{                         //读十六进制字符
            t[i++]=c;
            if( !(i<MAX))    //暂时保留原来代码中的愚蠢
                return 1; ;
            c=getchar();
            if(c=='\0')       //暂时保留原来代码中的愚蠢
                return 1;
        }
        while(c>='0'&&c<='9'||c>='a'&&c<='f'||c>='A'&&c<='F');

        t[i]='\0';
        printf("decimal number %d\n",htoi(t));    //输出转换结果

        printf("continue or not?");                    //询问是否继续
        i = 0;
        c=getchar();
    }
```

```
    while( !(c=='N'||c=='n') &&    printf("\n") );

    return 0;
}



#include <stdio.h>
#include <stdlib.h>

#define MAX 1000

#define YES 1
#define NO   0

int htoi(char s[]);
int be_hex( int );
char read_until_hex ( void ) ;
void read_hex ( char [] , unsigned );
int ask_continue(void);

int main( void )
{
    char t[MAX];

    do
    {
        t[0] = read_until_hex () ; //跳过非十六进制字符
        read_hex ( t , MAX ) ;        //读十六进制字符

        printf( "decimal number %d\n", htoi(t) );   //输出转换结果

        if( ask_continue() == NO )                          //询问是否继续
            break;
    }
    while( 1 );

    return 0;
}

int ask_continue(void)
{
     int c;
     printf("continue or not?");
```

```
    c = getchar() ;
    if(c=='N'||c=='n')
    {
        return NO;
    }
    printf("\n");
}

void read_hex ( char t[] , unsigned max )
{
    int i = 1 ;
    int c ;
    do
    {
        if( !( i < MAX ) )   //暂时保留原来代码中的愚蠢
            exit(1) ;
        c = getchar() ;
        if ( c=='\0' )          //暂时保留原来代码中的愚蠢
            exit(1) ;
        if( be_hex( c ) == YES )
            t [ i ++ ] = c ;
        else
        {
            t[i] = '\0';
            return ;
        }
    }
    while(1);

}

char read_until_hex ( void )
{
    int c;
    printf("input a HEX number:") ;
    do{
        c = getchar();
        if ( c =='\0' )       //暂时保留原来代码中的愚蠢
            exit(1) ;
    }
    while( be_hex(c) == NO );
    return c;
}
```

```
int be_hex( int c)
{
    return (c>='0'&&c<='9') ||
           (c>='a'&&c<='f') ||
           (c>='A'&&c<='F') ;
}
```

底子太差，大体上也只能改到这个地步了。
难怪有人说，不要试图修改垃圾代码，应该重写
如果可以用 ungetc()，可以改得更好些（不过老谭的书没介绍这个函数，这里就不写了）
if (c=='\0')　　exit(1) ;这样蠢话可以删去
此外老谭的代码还有一个很大的漏洞，就是压根没考虑到输入为负的情况

这个题目的代码可以这样写

首先,define MAX 为 1000 是极其愚蠢的
应该

```
#define HEX_BIT 4       //一个十六进制数占 4 位
#define MAX ( sizeof(int) * ( CHAR_BIT / HEX_BIT ) + 1 + 1 ) // +、-,\0
```

跳过前面的非十六进制形式字符
可以
scanf("%*[^0123456789ABCDEFabcdef]");
读入十六进制形式字符可以
scanf("%9[0123456789ABCDEFabcdef]%*[^\n]",hex_str);
这里的是 MAX-1（暂且不考虑前面的正负号）

这个代码可以很简单地写为

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

#define HEX_BIT 4       //一个十六进制数占 4 位
#define MAX ( sizeof(int) * ( CHAR_BIT / HEX_BIT ) + 1 + 1 ) // +、-,\0

int htoi(char []);

int main( void )
{
   char hex_str[MAX];
   char c;

   do
   {
```

```
        printf("input a HEX number:");
        scanf("%*[^0123456789ABCDEFabcdef]");
        scanf("%10[0123456789ABCDEFabcdef]%*[^\n]",hex_str);
        printf("decimal number %d\n",htoi(hex_str));
        printf("continue or not?");
        scanf(" %c",&c);
    }
    while(!(c=='N'||c=='n'));

    return 0;
}

int htoi(char s[])
{
    int i,n=0;

    for( i = 0 ; s[i] != '\0' ; i++ )
    {
        if( s[i] >= '0' && s[i] <= '9' )
            n =    n*16 + s[i] - '0' ;
        else if( s[i] >= 'a' && s[i] <= 'f' )
            n = n*16 + s[i] - 'a' + 10 ;
        esle if( s[i] >= 'A' && s[i] <= 'F' )
            n = n*16 + s[i] - 'A' + 10 ;
    }

    return n ;
}
```

P96

17.用递归法将一个整数 n 转换成字符串。例如，输入 483，应输出字符串"483"。n 的位数不确定，可以是任意位数的整数。

评："任意"两个字说明对计算机缺乏起码的认识
在计算机的词典中没有"任意"这两个字

P96~97

```
#include <stdio.h>
int main()
{ void convert(int n);
```

```
  int number;
  printf("input an integer:");
  scanf("%d",&number);
  printf("output: ");
  if(number<0)
    {putchar('-');putchar(' ');
     number=-number;
    }
  convert(number);
  printf("\n");
  return 0;
}

void convert(int n)
{int i;
if((i=n/10)!=0)
   convert(i);
putchar(n%10+'0');
putchar(32);
}
```

运行结果：

```
input an integer:2345678
output: 2 3 4 5 6 7 8

input an integer:-345
output: - 3 4 5
```

评：这是指东打西，打哪算哪

1.程序压根就没有按照要求“将一个整数 n 转换成字符串”

2. 题目要求“例如，输入 483，应输出字符串“483””，可结果呢，瞪着眼睛说瞎话

3.putchar(32); 是很恶心的写法

4.main()中的 printf("\n");很蹩脚应该写在 convert()中

P96~97

给出年、月、日，计算该日是该年的第几天

```
#include <stdio.h>
int main( )
{int sum_day(int month,int day);
int leap(int year);
int year,month,day,days;
printf ("input date(year,month,day):");
```

```
scanf("%d,%d,%d",&year,&month,&day);
days=sum_day(month,day);
if(leap(year)&&month>=3)
      days=days+1;
printf("is the %dth day in this year.\n",days);
return 0;
}


int sum_day(int month,int day)
{int day_tab[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
int i;
for(i=1;i<month;i++)
    day+=day_tab[ i];
return(day);
}


int leap(int year)
{int leap;
leap=year%4==0&&year%100!=0||year%400;
return(leap);
}
```

评：感觉最雷人的是这个

```
int leap(int year)
{int leap;
leap=year%4==0&&year%100!=0||year%400;
return(leap);
}
```

其实可以简单地写为

```
int leap(int year)
{
   return    year%4==0&&year%100!=0||year%400;
}
```

最终的 days 居然是在 main()求得，乖张

sum_day()的功能不可说

```
days=days+1;
```

业余

sum_day()中的

```
int day_tab[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
```

应为

static int const day_tab[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};

P99

```
#include <stdio.h>
int main()
{ void swap(int *p1,int *p2);
  int n1,n2,n3;
  int *p1,*p2,*p3;
  printf("input three integer n1,n2,n3:");
  scanf("%d,%d,%d",&n1,&n2,&n3);
  p1=&n1;
  p2=&n2;
  p3=&n3;
  if(n1>n2)swap(p1,p2);
  if(n1>n3)swap(p1,p3);
  if(n2>n3)swap(p2,p3);
  printf("Now,the order is:%d,%d,%d\n",n1,n2,n3);
  return 0;
}

void swap(int *p1,int *p2)
{int p;
p=*p1;*p1=*p2;*p2=p;
}
```

评：main()中的 p1,p2,p3 完全是多余的
写出这种啰嗦的代码多半是由于受到两个错误观念的影响：
& 是求地址运算
指针是一个变量

P99~100

```
/*
题目：输入 3 个字符串，按由小到大的顺序输出。
*/
#include <stdio.h>
#include <string.h>

int main()
{void swap(char *,char *);
char str1[20],str2[31],str3[20];
```

```
printf("input three line:\n");
gets(str1);
gets(str2);
gets(str3);
if(strcmp(str1,str2)>0)swap(str1,str2);
if(strcmp(str1,str3)>0)swap(str1,str3);
if(strcmp(str2,str3)>0)swap(str2,str3);
printf("Now,the order is:\n");
printf("%s\n%s\n%s\n",str1,str2,str3);
return 0;
}

void swap(char *p1,char *p2)
{char p[20];
strcpy(p,p1);strcpy(p1,p2);strcpy(p2,p);
}
```

运行结果：

```
input three line:
I study very hard.
C language is very interesting.
He is a professor.
Now,the order is:
C language is very interesting.
He is a professor.
I study very hard.
```

评：尽管这个运行结果似乎正确，然而却是瞎猫碰到死耗子。这个结果根本靠不住。
至少有两次数组越界错误

P106~107

7.有一字符串，包含 n 个字符。写一函数，将此字符串中从第 m 个字符开始的全部字符复制成为另一个字符串。

```
#include <stdio.h>
#include <string.h>
int main()
{void copystr(char *,char *,int );
int m;
char str1[20],str2[20];
printf("input string:");
gets(str1);
printf("which character that begin to copy?");
scanf("%d",&m);
if(strlen(str1)<m)
```

```
        printf("input error!");
else
{copystr(str1,str2,m);
   printf("result:%s\n",str2);
}
return 0;
}

void copystr(char *p1,char *p2,int m)
{int n;
n=0;
while(n<m-1)
   {n++;
     p1++;
   }
while(*p1!='\0')
   {*p2=*p1;
     p1++;
     p2++;
   }
*p2='\0';
}
```

评：代码的主要毛病就是拖泥带水，思路不清

```
int n;
n=0;
while(n<m-1)
   {n++;
      p1++;
   }
```

一段的功能
无非是 p+=m-1;而已
字符串的拷贝也无比啰嗦
在调用函数中已经处理了 m 不合理的情况
没有任何理由再把 m 作为参数

代码应改为

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

void copystr(char *,char *);

int main( void )
{
int m;
```

```
char str1[20],str2[20];
printf("输入字符串:");
gets(str1);
printf("从第几个字符开始拷贝?");
scanf("%d",&m);

if(strlen(str1)<m)
      printf("input error!");
else
{
      copystr(str2 , str1 + m - 1 );
      printf("result:%s\n",str2);
}

return 0;
}

void copystr(char *p1 , char *p2 )
{
    while( ( * p1 ++ = * p2 ++ ) != '\0' )
          ;
}
```

P107

```
#include <stdio.h>
int main()
{int upper=0,lower=0,digit=0,space=0,other=0,i=0;
 char * p,s[20];
 printf("input string:  ");
 while((s[ i ]=getchar())!='\n')i++;
 p=&s[0];
 while(*p!='\n')
  {if(('A'<=*p)&&(*p<='Z'))
      ++upper;
   else if(('a'<=*p)&&(*p<='z'))
      ++lower;
   else if(*p ==' ')
      ++space;
   else if((*p<='9')&&(*p>='0'))
      ++digit;
   else
      ++other;
   p++;
```

```
  }
 printf("upper case:%d       lower   case:%d",upper,lower);
 printf("        space:%d       digit:%d          other:%d\",space,digit,other);
 return 0;
}
```

```
    评：这段代码写得要多傻有多傻
    char * p,s[20];  简直是在无病呻吟
    根本用不着
    #include <stdio.h>
    int main( void )
    {
        int upper = 0 , lower = 0 , digit = 0 , space = 0 , other = 0 ;
        char c ;

        printf("输入一行字符\n");
        while(( c = getchar() )!='\n')
        {
            if( ( 'A' <= c ) && ( c <= 'Z' ) )
                    ++upper;
            else if( ( 'a' <= c ) && ( c <= 'z' ) )
                    ++lower;
            else if( c == ' ' )
                    ++space;
            else if( ( '0' <= c ) && ( c <= '9' ) )
                    ++digit;
            else
                    ++other;
        }
        printf("upper case:%d\tlower case:%d\t",upper,lower);
        printf("space:%d\tdigit:%d\tother:%d\n",space,digit,other);
        return 0;
    }
```

P108

将一个 5*5 的矩阵中最大的元素中最大的元素放在中心，4 个角分别放 4 个最小的元素（顺序为从左到右，从上到下依次从小到大存放），写一函数实现之，用 main 函数调用。

P109~110

```
#include <stdio.h>
int main()
```

```
{void change(int *p);
int a[5][5],*p,i,j;
printf("input matrix:\n");
for(i=0;i<5;i++)
    for(j=0;j<5;j++)
      scanf("%d",&a[i][j]);
p=&a[0][0];
change(p);
printf("Now matrix:\n");
for(i=0;i<5;i++)
    {for(j=0;j<5;j++)
       printf("%d ",a[i][j]);
     printf("\n");
    }
return 0;
}
void change(int *p)
{int i,j,temp;
int *pmax,*pmin;
pmax=p;
pmin=p;
for(i=0;i<5;i++)
    for(j=i;j<5;j++)
      {if(*pmax<*(p+5*i+j))pmax=p+5*i+j;
       if(*pmin>*(p+5*i+j))pmin=p+5*i+j;
      }
temp=*(p+12);
*(p+12)=*pmax;
*pmax=temp;
temp=*p;
*p=*pmin;
*pmin=temp;
pmin=p+1;
for(i=0;i<5;i++)
    for(j=0;j<5;j++)
      if(((p+5*i+j)!=p)&&(*pmin>*(p+5*i+j)))pmin=p+5*i+j;
temp=*pmin;
*pmin=*(p+4);
*(p+4)=temp;
pmin=p+1;
for(i=0;i<5;i++)
    for(j=0;j<5;j++)
      if(((p+5*i+j)!=(p+4))&&((p+5*i+j)!=p)&&(*pmin>*(p+5*i+j)))
          pmin=p+5*i+j;
```

```
temp=*pmin;
*pmin=*(p+20);
*(p+20)=temp;
pmin=p+1;
for(i=0;i<5;i++)
   for(j=0;j<5;j++)
      if(((p+5*i+j)!=p)&&((p+5*i+j)!=(p+4))&&((p+5*i+j)!=(p+20))&&
         (*pmin>*(p+5*i+j))) pmin=p+5*i+j;
temp=*pmin;
*pmin=*(p+24);
*(p+24)=temp;
}
```

评：从没见过比 change 更丑的函数
谭给出的测试为

运行结果：
input matrix:
35 34 33 32 31
30 29 28 27 26
25 24 23 22 21
20 19 18 17 16
15 14 13 12 11
Now matrix:
11 34 33 32 12
30 29 28 27 26
25 24 35 22 21
20 19 18 17 16
13 23 15 31 14

不过代码依然是错的

P112

```
char str[10][6];      //p 是指向由 6 个元素组成的一维数组的指针
```

评：这个注释雷人

P112

```
void sort(char s[10][6])
```

评：这个函数参数不全
那个 10 要多傻有多傻

p115

函数原型可写为

```
float integral(float a,float b,float(*fun)());
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p115
评：实际上代码中的函数原型为

```
float integral(float(*)(float),float ,float ,int);
```

P117

14.将 n 个数按输入时顺序的逆序排列，用函数实现。

评：这个题目本身就有毛病

“n 个数”是含糊不清的提法

P117

```
#include <stdio.h>
int main()
{ void sort(char *p,int m);
  int i,n;
  char *p,num[20];
  printf("input n:");
  scanf("%d",&n);
  printf("please input these number:\n");
  for(i=0;i<n;i++)
     scanf("%d",&num [ i ]);
  p=&num[0];
  sort(p,n);
  printf("Now ,the sequense is:\n");
  for(i=0;i<n;i++)
     printf("%d",num[ i ]);
  printf("\n");
  return 0;
}
```

```
void sort(char *p,int m)
{int i;
char temp,*p1,*p2;
for(i=0;i<m/2;i++)
{p1=p+i;
  p2=p+(m-1-i);
  temp=*p1;
  *p1=*p2;
  *p2=temp;
}
```

评：解答更是错误的惊人

```
char *p,num[20];
   for(i=0;i<n;i++)
      scanf("%d",&num[ i ]);
```

已经错的没边了

其他的错误就不提了

（注：测试一下输入

1234 234 8 7 6 5 4 3 2 1

不难发现代码是错误的）

P117~120

劣质代码评析——兼谈指针越界问题

【题目】

15.有一个班 4 个学生,5 门课程：①要求计算第一门课程的平均分；②找出两门以上课程不及格的学生，输出他们的学号和全部课程成绩及平均成绩；③找出平均成绩在 90 分以上或全部成绩在 85 分以上的学生。分别编写 3 个函数实现以上 3 个要求。

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p117

【评析】

题目前提基本充分，要求大体也还算合理，除了“输出他们的学号”让人感到有些突兀，因为前面从来没提到“学号”，看来是打算用 4 个学号表示“4 个学生”。但是“分别编写 3 个函数实现以上 3 个要求”这种要求，则属于僭越无理，这种要求是一种误导和教唆，在这种荒唐变态的要求下代码必然扭曲。

【原代码】

```
#include <stdio.h>
int main()
{void avsco(float *,float *);
void avscour1(char (*)[10],float *);
void fali2(char course[5][10],int num[],float *pscore,float aver[4]);
void good(char course[5][10],int num[4],float *pscore,float aver[4]);
int i,j,*pnum,num[4];
float score[4][5],aver[4],*pscore,*paver;
```

```
char course[5][10],(*pcourse)[10];
printf("input course\n");
pcourse=course;
for(i=0;i<5;i++)
     scanf("%s",course[i]);
printf("input NO. and scores:\n");
printf("NO.");
for(i=0;i<5;i++)
     printf(",%s",course[i]);
printf("\n");
pscore=&score[0][0];
pnum=&num[0];
for(i=0;i<4;i++)
     {scanf("%d",pnum+i);
       for(j=0;j<5;j++)
          scanf("%f",pscore+5*i+j);
     }
paver=&aver[0];
printf("\n\n");
avsco(pscore,paver);
avscour1(pcourse,pscore);
printf("\n\n");
fali2(pcourse,pnum,pscore,paver);
printf("\n\n");
good(pcourse,pnum,pscore,paver);
return 0;
}

void avsco(float *pscore,float *paver)
{int i,j;
float sum,average;
for(i=0;i<4;i++)
   {sum=0.0;
     for(j=0;j<5;j++)
       sum=sum+(*(pscore+5*i+j));
     average=sum/5;
     *(paver+i)=average;
   }
}

void avscour1(char (*pcourse)[10],float *pscore)
{int i;
float sum,average1;
sum=0.0;
```

```
for(i=0;i<4;i++)
     sum=sum+(*(pscore+5*i));
average1=sum/4;
printf("course 1:%s average score:%7.2f\n",*pcourse,average1);
}

void fali2(char course[5][10],int num[],float *pscore,float aver[4])
{int i,j,k,label;
printf("    ==========Student who is fail in two courses=======\n");
printf("NO.");
for(i=0;i<5;i++)
     printf("%11s",course[i]);
printf("       average\n");
for(i=0;i<4;i++)
{label=0;
   for(j=0;j<5;j++)
    if(*(pscore+5*i+j)<60.0)label++;
   if(label>=2)
    {printf("%d",num[i]);
     for(k=0;k<5;k++)
        printf("%11.2f",*(pscore+5*i+k));
     printf("%11.2f\n",aver[i]);
    }
}
}

void good(char course[5][10],int num[4],float *pscore,float aver[4])
     {int i,j,k,n;
   printf("    ==========Students whose score is good=======\n");
   printf("NO.");
   for(i=0;i<5;i++)
      printf("%11s",course[i]);
   printf("       average\n");
   for(i=0;i<4;i++)
    {n=0;
     for(j=0;j<5;j++)
        if(*(pscore+5*i+j)>85.0)n++;
     if(n==5||(aver[i]>=90))
     {printf("%d",num[i]);
      for(k=0;k<5;k++)
         printf("%11.2f",*(pscore+5*i+k));
      printf("%11.2f\n",aver[i]);
     }
}
```

}
——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p117~120
下面逐条评析。
【原代码】
02. int main()
【评析】
这是一个老生常谈的问题，定义函数而不写形参类型亦即非函数原型形式(not prototype-format)是一种过时的写法(an obsolescent feature)。main()函数的头部应该写为：
int main (void)
【原代码】
03.{void avsco(float *,float *);
04.void avscour1(char (*)[10],float *);
05.void fali2(char course[5][10],int num[],float *pscore,float aver[4]);
06.void good(char course[5][10],int num[4],float *pscore,float aver[4]);
【评析】
这个也是谭氏代码特有的一个毛病。把函数类型声明写在函数之内，不但使函数臃肿不堪，而且限制了函数的使用范围。应该从函数中移出，放在 main()之前为好。
【原代码】
07.int i,j,*pnum,num[4];
08.float score[4][5],aver[4],*pscore,*paver;
【评析】
int num[4];和 float score[4][5];是这段代码核心的数据结构，如果不考虑使用结构体数据类型的话，尚属合理。但把 i,j,pnum 和 num 放在一起定义，尤其是把 i,j 和 num 放在一起声明，则显得不伦不类。
把 score 数组元素设计为 float 类型并非不可，但矛盾的是，后面出现了
【原代码】
69. if(*(pscore+5*i+j)<60.0)label++;
89. if(*(pscore+5*i+j)>85.0)n++;
90. if(n==5||(aver[i]>=90))
【评析】
前两句中的表达式中有 float 与 double 的混合运算，后一句中出现了 float 与 int 的混合运算，这在一般情况下是应该竭力避免的。因为这可能会导致截断误差的警告，而且至少在理论上存在着不必要的类型转换导致程序低效。
至于“aver[i]>=90”，通常是对类型不清醒而写出的似是而非的表达式，正规的写法应该是：aver[i]>=90.0F。
【原代码】
10.printf("input course\n");
11.pcourse=course;
12.for(i=0;i<5;i++)
13. scanf("%s",course[i]);
【评析】
这几句话的作用是输入 5 门课程名称，但却是画蛇添足。因为题目并没有要求输入课程名称，所以课程名称应该视为已知。
不满足题目要求是一种错误，完成题目额外要求也是一种错误，这叫做多做之过。
即使是出于纯粹个人练习的目的故意多写一段不必要的代码，这段代码也应该写成一个函数。
其中最可笑的是 11.行那条语句，因为它和这段代码的功能毫无关系，不当不正地插在这里显得非常不伦

不类。后面可以看到这条语句不但位置是错误的，而且压根就是一句可以删除的废话。

【原代码】

```
14.printf("input NO. and scores:\n");
15.printf("NO.");
16.for(i=0;i<5;i++)
17.     printf(",%s",course[i]);
18.printf("\n");
```

【评析】

这段代码的作用是输出一段提示信息，并为输入提供一个表头。但是",%s"这个格式设计得很拙劣，它的输出的效果是：

NO.,English,Computer,Math,Physics,Chemistry

NO.与 English 之间的“,”不但不伦不类，而且这种样式容易让程序的使用者误认为输入数据之间应该用“,”分隔。

实际上这段代码没有必要，如果需要写也应该用函数完成。

【原代码】

```
19.pscore=&score[0][0];
20.pnum=&num[0];
```

【评析】

这是两句无厘头的废话。

首先它们完全可以写成更简洁的形式：

```
pscore=score[0];
pnum=num;
```

其次，这两句话完全可以不写。也就是说 pscore 和 pnum 这两个变量完全是多余的。

【原代码】

```
21.for(i=0;i<4;i++)
22.     {scanf("%d",pnum+i);
23.       for(j=0;j<5;j++)
24.           scanf("%f",pscore+5*i+j);
25.     }
```

【评析】

这段代码中的

```
scanf("%d",pnum+i);
```

一句应该写成

```
scanf("%d",num+i);
```

而

```
scanf("%f",pscore+5*i+j);
```

则等价于

```
scanf("%f", *score + 5 * i + j );
```

但是需要特别指出的是，这两者都是错误的写法。这个错误就是本文副题所提到的指针越界问题。

指针可以与整数做加、减运算是有前提的。前提之一是这个指针必须是指向数据对象(Object)。例如：

```
int i;
```

&i 这个指针可以+0、+1。但是指向函数的指针或指向 void 类型的指针没有加减法运算。

前提之二是这个指针必须指向数组元素（单个 Object 视同一个元素的数组）或指向数组最后一个元素之后的那个位置。例如：

int a[2];
&a[0]、&a[1]、&a[1]+1(即 a、a+1、a+2)这些指针可以进行加减法运算。
第三，指针进行加减法运算的结果必须也指向数组元素或指向数组最后一个元素之后的那个位置。例如，对于指向 a[0]的指针 a，只能+0、+1、+2，对于 a+2 这个指针，只能-0、-1、-2。如果运算结果不是指向数组元素或指向数组元素最后一个元素之后的位置的情况，C 语言并没有规定这种运算行为的结果是什么，换句话说这是一种未定义行为(Undefined Behavior，后面简称 UB)。
前面【原代码】中的 pscore 是一个 float *类型的指针，它指向的是
float score[4][5];
中的 score[0][0]，score[0][0]是 score[0]这个一维数组(float [5])的元素，因此 pscore 只可以+0、+1、+2、+3、+4、+5，但是绝对不能加其他的整数。而
pscore+5*i+j
在 i≥1，j≥0 时显然违背了这种规则。因此是一种未定义行为，亦即是一种错误的代码。
很多人以为，既然
pscore+5*i+j
没有超出二维数组的范围，因此就没有问题。这种看法是错误的。构成二维数组的元素并非是 float 类型的数据对象，而是 float [5]类型的一维数组数据对象。
在这个例子中，float 类型的数据对象是构成一维数组的对象，指向 float 类型的指针最多只能+5。否则就是一种 UB，这种行为是一种严重的软件安全隐患。
【原代码】
```
21.for(i=0;i<4;i++)
22.     {scanf("%d",pnum+i);
23.       for(j=0;j<5;j++)
24.           scanf("%f",pscore+5*i+j);
25.     }
```
【评析】
应该写为
```
for( i = 0 ; i < 4 ; i++ )
{
    scanf("%d" , num + i );
    for( j = 0 ; j < 5 ; j++ )
        scanf("%f" , score[i] + j );
}
```
pnum 和 pscore 都是完全用不着的。当然，用函数完成这个功能更好。
【原代码】
```
26.paver=&aver[0];
27.printf("\n\n");
28.avsco(pscore,paver);
```
【评析】
26.行完全是多余的。
27.行很丑陋，应该写在相应的函数内。从效率上来说不如写 puts("\n");
28.行参数不完整，缺少两个数组第一维的维度；pscore,paver 是两个完全不必要的变量，因为他们完全等价于*score,aver。
下面再看 avsco()函数的定义
【原代码】

```
37.void avsco(float *pscore,float *paver)
38.{int i,j;
39.float sum,average;
40.for(i=0;i<4;i++)
41.  {sum=0.0;
42.   for(j=0;j<5;j++)
43.     sum=sum+(*(pscore+5*i+j));
44.   average=sum/5;
45.   *(paver+i)=average;
46.  }
47.}
```

【评析】

首先函数的参数不完整，导致函数内有两个 Magic Number :4,5。

43.行的“*(pscore+5*i+j)”这种写法是错误的。前面提到过，这是一种 UB。

假使不考虑这个错误 43.行也应该写为

```
sum += *(pscore+5*i+j) ;
```

这样显然更简洁。

44.行混合运算，前面提过，这里不再赘述。

39.行的 average 是一个完全多余的变量，因为 44.行和 45.行可以简单地写为：

```
*(paver+i) = sum/5.0F;
```

下面再回到 main()。

【原代码】

```
29.avscour1(pcourse,pscore);
```

【评析】

这句很垃圾。

首先，pcourse、pscore 这两个变量是多余的，因为这两个实参无非是 course 和*score 而已。

其次，这个函数的功能并不满足“计算第一门课程的平均分”而是输出第一门课程的平均分。输出是 main() 的事情，而不是“计算”平均分的功能。

第三，“计算第一门课程的平均分”这个功能要求提得很愚蠢。难道计算二门课程的平均分还要另外写一个函数不成？如果可以写一个计算第 n 门课程的平均分，没有人会愚蠢地提出“计算第一门课程的平均分”这样的函数功能要求。

第四，作为“计算第一门课程的平均分”的函数，pcourse 这个参数是多余的，因为这个参数对于计算是没有用处的。另一方面这个函数又缺少数组长度的参数。

再来看一下 avscour1()的函数定义：

【原代码】

```
49.void avscour1(char (*pcourse)[10],float *pscore)
50.{int i;
51.float sum,average1;
52.sum=0.0;
53.for(i=0;i<4;i++)
54.    sum=sum+(*(pscore+5*i));
55.average1=sum/4;
56.printf("course 1:%s average score:%7.2f\n",*pcourse,average1);
57.}
```

【评析】
52.行，多余，应在定义 sum 时顺便初始化。
53.行，这里有一个 Magic Number:4，这是因为函数参数不完整造成的。
54.行，这里同样存在数组越界的错误：“*(pscore+5*i)”
54.行，不考虑越界错误，这句话应该写成

```
sum +=*(pscore+5*i) ;
```

这样代码更简洁。
55.行，混合运算。
56.行，从这里不难看出 average1 这个变量是多余的。因为这句话可以写为

```
printf("course 1:%s average score:%7.2f\n",*pcourse,sum/4.0F );
```

29.行及 49.行~57.行应写为：

```
int main( void )
{
    /*      */
    printf("course %s average score:%7.2f\n",course[1-1]
                    ,average(score,5,1-1));//average(score , sizeof score / sizeof *score,1-1)
  /*    */
}

float    average( float p_score[][5],int    num_stu , int n)
{
    float    sum = 0.0 ;
    int    num_stu_ = num_stu ;
    while( num_stu_ -- > 0 )
    {
        sum += *(*p_score++ + n ) ;
    }
    return sum / (float) num_stu ;
}
```

【原代码】
31.fali2(pcourse,pnum,pscore,paver);
【评析】
这个函数的 4 个参数同样很垃圾。很多初学者都容易犯这种幼稚病，他们偏执地只使用变量做实参，却不懂得实参是一个表达式。事实上那些变量是多余的。
与此函数调用相应的函数类型声明：
【原代码】
05.void fali2(char course[5][10],int num[],float *pscore,float aver[4]);
【评析】
同样丑陋不堪，其中的 5 和 4 是概念不清的表现，对应的函数定义也存在同样的毛病：
【原代码】

```
59.void fali2(char course[5][10],int num[],float *pscore,float aver[4])
60.{int i,j,k,label;
61.printf("    ==========Student who is fail in two courses=======\n");
62.printf("NO.");
```

```
63.for(i=0;i<5;i++)
64.     printf("%11s",course[i]);
65.printf("       average\n");
66.for(i=0;i<4;i++)
67.{label=0;
68.   for(j=0;j<5;j++)
69.     if(*(pscore+5*i+j)<60.0)label++;
70.   if(label>=2)
71.     {printf("%d",num[i]);
72.       for(k=0;k<5;k++)
73.           printf("%11.2f",*(pscore+5*i+k));
74.       printf("%11.2f\n",aver[i]);
75.     }
76.}
77.}
```

【评析】

这个函数还有很多毛病：

比如过于复杂；因为任务离数据过于遥远而导致参数过多。

这是由于 main()无所作为，没有对任务做适当的分解，只是把数据一股脑地推给了函数的缘故。

同时也说明了问题要求把“找两门课程以上课程不及格的学生，输出他们的学号和全部课程成绩及平均成绩”写成一个函数十分荒谬愚蠢。

61.~64.行和 71.~74.行的代码完全应该写在这个函数的外部由单独的函数完成。

67.行的 label 是莫名奇妙的名字。

69.行的*(pscore+5*i+j)是错误的表达式，存在混合运算问题。

70.行的代码效率低下，它应该与前面的 if 语句一起成为前面 for 循环的内部语句，即应写为：

```
for(j=0;j<5;j++)
{
          if(*(pscore+5*i+j)<60.0F)
               label++;
          if(label==2)
          {
             printf("%d",num[i]);
             for(k=0;k<5;k++)
                  printf("%11.2f",*(pscore+5*i+k));
             printf("%11.2f\n",aver[i]);
             break ;
          }
}
```

顺便说一句，这里的*(pscore+5*i+k)也是错误的表达式。

此外，这个函数的函数名 fali2 也非常怪异，不清楚是哪国英语这样表述不及格。

【原代码】

```
33.good(pcourse,pnum,pscore,paver);
```

【评析】

和前面 fali2()函数调用存在同样的毛病，不再赘述。

【原代码】

```
79.void good(char course[5][10],int num[4],float *pscore,float aver[4])
80.    {int i,j,k,n;
81.   printf("    ==========Students whose score is good=======\n");
82.   printf("NO.");
83.   for(i=0;i<5;i++)
84.      printf("%11s",course[i]);
85.   printf("       average\n");
86.   for(i=0;i<4;i++)
87.     {n=0;
88.      for(j=0;j<5;j++)
89.         if(*(pscore+5*i+j)>85.0)n++;
90.      if(n==5||(aver[i]>=90))
91.      {printf("%d",num[i]);
92.        for(k=0;k<5;k++)
93.           printf("%11.2f",*(pscore+5*i+k));
94.        printf("%11.2f\n",aver[i]);
95.      }
96.}
97.}
```

【评析】

这个函数的毛病和 fali2()的函数定义一样

值得一提的是这段代码中的 81.行~85.行居然和 61.行~65.行完全一致，91.行~55.行居然和 71.行~75.行完全一致。对任何程序员来说，这种代码都是一种耻辱。

下面是对该问题及代码的修正。

————————————————————————————————————

【修正】

对题目的修正：

有一个班 4 个学生,5 门课程：

①要求计算第一门课程的平均分；

②找出两门以上课程不及格的学生，输出他们的学号和全部课程成绩及平均成绩；

③找出平均成绩在 90 分以上或全部成绩在 85 分以上的学生。

注：原来要求“编写 3 个函数实现以上 3 个要求”实在太愚蠢。

【重构】

```
#include <stdio.h>

#define NUM_STUDENT 4
#define NUM_COURSE 5

int main( void )
{
   int num[NUM_STUDENT] = { 101 , 102 , 103 , 104 };
   char    * course[NUM_COURSE] = {"English","Computer","Math","Physics","Chemistry"} ;
   double    score[NUM_STUDENT][NUM_COURSE]
```

```
            = {
                { 34.0 , 56.0 , 88.0 , 99.0 , 89.0 },
                { 27.0 , 88.0 , 99.0 , 67.0 , 78.0 },
                { 99.0 , 90.0 , 87.0 , 86.0 , 89.0 },
                { 78.0 , 89.0 , 99.0 , 56.0 , 77.0 }
              };


   return 0;
}
```

注：因为题目并没有要求输入学号、课程名称、成绩，所以这些数据应该视同已知。此外如果使用结构体类型，数据结构将更为合理。

【分析】

首先解决“①要求计算第一门课程的平均分;”

这个问题的一般提法是，求 4 个学生的 5 门课程中第 1 门课程的平均分。

程序员应该学会根据特殊问题提出具有一般性的问题并加以解决，而不能就事论事地盯着单独的具体问题。这样写出的代码才具有较好的通用性性、可复用性和可维护性。

因此这个函数的函数原型应该是

```
double get_average( double score[][5] , int num_student , int order_course );
```

其中,score:存储成绩的二维数组；num_student:学生人数；order_course ：课程的序号。

这样的函数，显然不但可以解决计算第一门课程的平均分这样的问题，也可以计算第 n (1<=n<=5)门课程的平均分这样的问题;不但可以解决 4 个学生情况下的这类问题，也可以解决 k（k>0）个学生情况下的这类问题。

【重构】

```
#include <stdio.h>

#define NUM_STUDENT 4
#define NUM_COURSE 5
double get_average( double [][NUM_COURSE] , int    , int    );
int main( void )
{
   int num[NUM_STUDENT] = { 101 , 102 , 103 , 104 };
   char    * course[NUM_COURSE] = {"English","Computer","Math","Physics","Chemistry"} ;
   double    score[NUM_STUDENT][NUM_COURSE]
            = {
                { 34.0 , 56.0 , 88.0 , 99.0 , 89.0 },
                { 27.0 , 88.0 , 99.0 , 67.0 , 78.0 },
                { 99.0 , 90.0 , 87.0 , 86.0 , 89.0 },
                { 78.0 , 89.0 , 99.0 , 56.0 , 77.0 }
              };

    //①计算第一门课程的平均分;
   printf("第%d 门课程%s 的平均分为%7.2f\n" ,
                1 , course[ 1 - 1 ] , get_average( score , NUM_STUDENT , 1 - 1 ) );
```

```
   return 0;
}
double get_average( double score[][NUM_COURSE] , int n , int k )
{
    double sum = .0 ;
    int i ;

    for( i = 0 ; i < n ; i ++ )
        sum += score[i][k];

    return sum/(double)n;
}
```

【分析】

再看第二个要求：②找出两门以上课程不及格的学生，输出他们的学号和全部课程成绩及平均成绩；

“找出”和“输出”是两件事情，不宜作为一个函数完成。此外“两门以上课程不及格的学生”是一个集合，除非构造出“集合”这种数据类型，否则单纯用函数无法完成，因为函数只能返回一个值。所以在“找出”这件事上 main()不应该无所作为。

【重构】

```
#include <stdio.h>

#define NUM_STUDENT 4
#define NUM_COURSE 5
double get_average( double [][NUM_COURSE] , int    , int    );
void print_head(char    *([]), int );
int less_than(double [], int    , double , int);
void output(double [], int);
int main( void )
{
int num[NUM_STUDENT] = { 101 , 102 , 103 , 104 };
char    * course[NUM_COURSE] = {"English","Computer","Math","Physics","Chemistry"} ;
double    score[NUM_STUDENT][NUM_COURSE]
          = {
               { 34.0 , 56.0 , 88.0 , 99.0 , 89.0 },
               { 27.0 , 88.0 , 99.0 , 67.0 , 78.0 },
               { 99.0 , 90.0 , 87.0 , 86.0 , 89.0 },
               { 78.0 , 89.0 , 99.0 , 56.0 , 77.0 }
             };

//①计算第一门课程的平均分；
printf("第%d 门课程%s 的平均分为%7.2f\n" ,
               1 , course[ 1 - 1 ] , get_average( score , NUM_STUDENT , 1 - 1 ) );

//②找两门课程以上课程不及格的学生，输出他们的学号和全部课程成绩及平均成绩；
```

```
{
        int i;
        puts("\n 两门课程以上课程不及格的学生");
        print_head( course , NUM_COURSE );        //输出表头
        for( i = 0 ; i < NUM_STUDENT ; i ++ )
            if( less_than(score[i],NUM_COURSE,60.,2) ) //有 2 科成绩低于 60.0
            {
                printf("%d    ",num[i]);            //学号
                output(score[i],NUM_COURSE); //输出各科成绩及平均分
            }
}

return 0;
}

//输出一名学生各科成绩及平均分
void output(double score[], int n)
{
    double sum = 0.0 ;
    int i ;
    for( i = 0 ; i < n ; i ++ )
    {
        printf("%11.2f",score[i]);
        sum += score[i] ;
    }
    printf( "%11.2f\n" , sum / (double) n );
}

//score 中有 amount 科成绩低于 fail 返回 1，否则返回 0
int less_than(double score[], int n , double fail , int amount )
{
    int i , num = 0 ;
    for( i = 0 ; i < n ; i ++ )
    {
        if( score[i] < fail )
            num ++ ;

        if( num == amount )
            return 1 ;
    }
    return 0;
}

//输出表头
```

```
void print_head(char    *(course[]), int n )
{
    int i;
    printf("学号    ");
    for( i = 0 ; i < n ; i ++ )
    {
        printf("%11s",course[i]);
    }
    printf("      平均分\n");

}

// 返回 score[n][5]中第 k 科平均分
double get_average( double score[][NUM_COURSE] , int n , int k )
{
    double sum = .0 ;
    int i ;

    for( i = 0 ; i < n ; i ++ )
        sum += score[i][k];

    return sum/(double)n;
}
```

【分析】

继续解决第三个问题：③找出平均成绩在 90 分以上或全部成绩在 85 分以上的学生。

由于“平均成绩在 90 分以上或全部成绩在 85 分以上的学生”是一个集合，所以解决这个问题同样不适合用一个函数完成，而应该由 main()和其他函数配合完成。但是这里判断“平均成绩在 90 分以上”及“全部成绩在 85 分以上”应该分别由两个函数实现。

而“全部成绩在 85 分以上”显然可以由前面写过的函数 less_than() 来表达：!less_than(score[i],NUM_COURSE,85.0 , 1)。

此外增加了一个求一名学生平均成绩的函数，并对前面的 output()函数做了适当修改。

最后的代码为：

【重构】

```
#include <stdio.h>

#define NUM_STUDENT 4
#define NUM_COURSE 5
double get_average( double [][NUM_COURSE] , int    , int    );
void print_head(char    *([]), int );
int less_than(double [], int    , double , int);
void output(double [], int);
double get_average_student(double [], int);
int main( void )
{
```

```
int num[NUM_STUDENT] = { 101 , 102 , 103 , 104 };
char    * course[NUM_COURSE] = {"English","Computer","Math","Physics","Chemistry"} ;
double    score[NUM_STUDENT][NUM_COURSE]
          = {
              { 34.0 , 56.0 , 88.0 , 99.0 , 89.0 },
              { 27.0 , 88.0 , 99.0 , 67.0 , 78.0 },
              { 99.0 , 90.0 , 87.0 , 86.0 , 89.0 },
              { 78.0 , 89.0 , 99.0 , 56.0 , 77.0 }
            };

//①计算第一门课程的平均分；
printf("第%d 门课程%s 的平均分为%7.2f\n" ,
              1 , course[ 1 - 1 ] , get_average( score , NUM_STUDENT , 1 - 1 ) );

{//②找两门课程以上课程不及格的学生，输出他们的学号和全部课程成绩及平均成绩；
        int i ;
        puts("\n 两门课程以上课程不及格的学生");
        print_head( course , NUM_COURSE );        //输出表头
        for( i = 0 ; i < NUM_STUDENT ; i ++ )
            if( less_than(score[i],NUM_COURSE,60.,2) ) //有 2 科成绩低于 60.0
            {
                printf("%d    ",num[i]);            //学号
                output(score[i],NUM_COURSE); //输出各科成绩及平均分
            }
}

{//③找出平均成绩在 90 分以上或全部成绩在 85 分以上的学生。
        int i ;
        puts("\n 平均成绩在 90 分以上或全部成绩在 85 分以上的学生有：");
        for( i = 0 ; i < NUM_STUDENT ; i ++ )
            if(    get_average_student(score[i],NUM_COURSE) > 90.0//平均成绩在 90 分以上
                ||!less_than( score[i],NUM_COURSE , 85.0 , 1 )     //全部成绩在 85 分以上
              )
                printf("第%d 号\n",num[i]);            //学号
}

return 0;
}

//求一名学生各科成绩平均分
double get_average_student(double score[], int n)
{
    double sum = 0.0 ;
    int i ;
```

```
    for( i = 0 ; i < n ; i ++ )
        sum += score[i] ;

    return    sum / (double) n ;
}

//输出一名学生各科成绩及平均分
void output(double score[], int n)
{
    int i ;
    for( i = 0 ; i < n ; i ++ )
        printf("%11.2f",score[i]);

    printf( "%11.2f\n" , get_average_student( score , n) );
}

//score 中有 amount 科成绩低于 fail 返回 1，否则返回 0
int less_than(double score[], int n , double fail , int amount )
{
    int i , num = 0 ;
    for( i = 0 ; i < n ; i ++ )
    {
        if( score[i] < fail )
            num ++ ;

        if( num == amount )
            return 1 ;
    }
    return 0;
}

//输出表头
void print_head(char    *(course[]), int n )
{
    int i;
    printf("学号    ");
    for( i = 0 ; i < n ; i ++ )
        printf("%11s",course[i]);

    printf("      平均分\n");

}

//  返回 score[n][5]中第 k 科平均分
```

```
double get_average( double score[][NUM_COURSE] , int n , int k )
{
    double sum = .0 ;
    int i ;

    for( i = 0 ; i < n ; i ++ )
        sum += score[i][k];

    return sum/(double)n;
}
```

P120~121

输入一个字符串，串内有数字和非数字字符。例如：

A123x456 17960?302tab5876

将其中连续的数字作为一个整数，依次存放到一数组 a 中。例如，123 放在 a[0]，456 放在 a[1]……，统计出共有多少个整数，并输出这些数。

评：这个题目本身就有问题。

首先是“输入一个字符串”。字符串是 C 代码语境下的术语和概念，而不是问题语境下的。

其次没有提到长度，这是很不严谨的。

“将其中连续的数字作为一个整数”，同样没有规定范围，不是任何长度连续的数字都可以转换为整数存入数组的。这些数字的进制也应该提一下

总共有多少“连续的数字”也没有提，这同样是不严谨的

P121~122

```
01.#include <stdio.h>
02.int main()
03.{
04.char str[50],*pstr;
05.int i,j,k,m,e10,digit,ndigit,a[10],*pa;
06.printf("input a string\n");
07.gets(str);
08.pstr=&str[0];
09.pa=&a[0];
10.ndigit=0;
11.i=0;
12.j=0;
13.while(*(pstr+i)!='\0')
```

```
14.    {if((*(pstr+i)>='0')&&(*(pstr+i)<='9'))
15.       j++;
16.     else
17.        {if(j>0)
18.           {digit=*(pstr+i-1)-48;
19.            k=1;
20.            while(k<j)
21.               {e10=1;
22.                for(m=1;m<=k;m++)
23.                    e10=e10*10;
24.                digit=digit+(*(pstr+i-1-k)-48)*e10;
25.                k++;
26.               }
27.              *pa=digit;
28.              ndigit++;
29.              pa++;
30.              j=0;
31.           }
32.        }
33.       i++;
34.    }
35.if(j>0)
36.{digit=*(pstr+i-1)-48;
37.    k=1;
38.    while(k<j)
39.       {e10=1;
40.        for(m=1;m<=k;m++)
41.            e10=e10*10;
42.        digit=digit+(*(pstr+i-1-k)-48)*e10;
43.        k++;
44.       }
45.     *pa=digit;
46.     ndigit++;
47.     j=0;
48.}
49.printf("There are %d numbers in this line,they are:\n",ndigit);
```

50.j=0;

51.pa=&a[0];

52.for(j=0;j<ndigit;j++)

53. printf("%d ",*(pa+j));

54.printf("\n");

55.return 0;

56.}

评：仅有一个 main()函数，从这点就能足以判定这是一段垃圾代码

04.char str[50],*pstr;

05.int i,j,k,m,e10,digit,ndigit,a[10],*pa;

这两行中的 str、a、和 ndigit 是这段代码所采用的基本数据结构。其中的 char str[50]尚属差强人意，权且接受，就当做输入的字符串不超过 50 个字符。

但是，int a[10]就绝对莫名其妙了。因为在 char str[50]中最多可以存储 50/2 个整数。所以，如果输入为：

1a2a3a4a5a6a7a8a9a0a1a2a3a4a5a6a7a8a9a0a

的情况下，程序得到的结果必然是错误的。这一点连运行测试都不必做就可以确定。

04.char str[50],*pstr;

05.int i,j,k,m,e10,digit,ndigit,a[10],*pa;

把变量不分主次地定义在一起，尤其是把那些 i,j,k,m 与程序的主要数据结构眉毛胡子一把抓地定义在一起，也是一种恶劣的编程习惯。

此外还要说明的是，这里定义的 pstr 和 pa 都是毫无意义的变量。在一个函数内操作数组，绝对可以不通过指针变量。

i,j,k,m 这种垃圾命名法这里就不赘了。

08.pstr=&str[0];

09.pa=&a[0];

10.ndigit=0;

11.i=0;

12.j=0;

喧宾夺主。垃圾

应该在定义变量时初始化

13.while(*(pstr+i)!='\0')

(pstr+i)无非就是 a[i]而已，写成(pstr+i)毫无必要，显得十分做作。在后面的代码中 pstr 这个变量从没有改变，所以这个变量根本就是多余的。

此外对这个问题而言，从头到尾地读取字符串中的字符属于无策。因为一旦*(pstr+i)!='\0'，很可能前面还有没有处理的数字。所以 13.行的写法是事先缺乏思考的随手写法。

一句话

根本就不应该 while(*(pstr+i)!='\0')

```
14.    {if((*(pstr+i)>='0')&&(*(pstr+i)<='9'))
15.       j++;
```

完全不知所云

j 那个名字太拙劣之故

17.行~32.行

35.行~48.行

一样的垃圾代码居然写了两次

垃圾的平方

```
17.         {if(j>0)
18.            {digit=*(pstr+i-1)-48;
19.             k=1;
20.             while(k<j)
21.                {e10=1;
22.                 for(m=1;m<=k;m++)
23.                    e10=e10*10;
24.                 digit=digit+(*(pstr+i-1-k)-48)*e10;
25.                 k++;
26.                }
27.              *pa=digit;
28.              ndigit++;
29.              pa++;
30.              j=0;
31.            }
32.         }
```

这是典型的垃圾代码。

首先，18.行：digit=*(pstr+i-1)-48;这句，其实完全可以写到后面的循环语句中：

```
01.             k=1;
02.             digit =0;
03.             while(k<j)
04.                {e10=1;
05.                 for(m=1;m<=k;m++)
06.                    e10=e10*10;
07.                 digit=digit+(*(pstr+i-1-k)-48)*e10;
08.                 k++;
09.                }
```

现在不难看出，这个 while 语句其实应该用 for 语句实现：

```
01.                 for(k=0,digit=0;k<j;k++)
02.                 {
03.                     e10=1;
04.                     for(m=1;m<=k;m++)
05.                         e10=e10*10;
06.                     digit += (*(pstr+i-1-k)-48)*e10;
07.                 }
```

现在会发现内部那个循环很 2，因为这句话可以写成

```
01.                 for(k=0,digit=0,e10=1;k<j;k++,e10*=10)
02.                 {
03.                     digit += (*(pstr+i-1-k)-48)*e10;
04.                 }
```

其中的 48 是很糟糕的写法，应该写成'0'。因此 18.~20.行无非是

```
01.             for(k=0,digit=0,e10=1;k<j;k++,e10*=10)
02.             {
03.                 digit += (*(pstr+i-1-k)-'0')*e10;
04.             }
```

而已。也可以写为

```
01.             for(k=0,digit=0,e10=1;k<j;k++,e10*=10)
02.             {
03.                 digit += ( str[i-1-k] -'0' ) * e10 ;
04.             }
```

所以说 pstr 这个变量是完全多余的。

```
01.27.              *pa=digit;
02.28.              ndigit++;
03.29.              pa++;
```

因为有了 ndigit，所以变量 pa 完全是多余的。这几句话可以简单地写为：

a[ndigit++]=digit;

```
01.35.if(j>0)
02.36.{digit=*(pstr+i-1)-48;
03.37.  k=1;
04.38.  while(k<j)
05.39.      {e10=1;
06.40.       for(m=1;m<=k;m++)
07.41.          e10=e10*10;
08.42.       digit=digit+(*(pstr+i-1-k)-48)*e10;
```

```
09.43.          k++;
10.44.        }
11.45.      *pa=digit;
12.46.      ndigit++;
13.47.      j=0;
14.48.}
```

这种代码居然能在一个函数中出现两次，实在令人瞠目结舌。

值得一说的是其中的 j=0 不知所云，毫无意义。因此代码可改为：

```
    if( j > 0 )
    {
        for(k=0,digit=0,e10=1;k<j;k++,e10*=10)
        {
            digit += (str[i-1-k] -'0') * e10 ;
        }
        a[ndigit++]=digit;
    }
50.j=0;
51.pa=&a[0];
52.for(j=0;j<ndigit;j++)
53.      printf("%d ",*(pa+j));
54.printf("\n");
```

前两句完全多余，代码可改写为

```
for(j=0;j<ndigit;j++)
    printf("%d ",a[j] );
putchar('\n');
```

下面稍微整理一下，修改后代码为：

```
#include <stdio.h>
int main( void )
{
    char str[50];
    int ndigit = 0 , a[10] = {0} ;
    int i , j = 0 , k , e10 ;

    printf("input a string\n");
    gets(str);
```

```
    for( i = 0 ; str[i] != '\0' ; i ++ )
        if(( str[i] >= '0' ) && ( str[i] <= '9' ) )
            j++;
        else
            if( j > 0 )
            {
                for( k = 0 , e10 = 1 ; k < j ; k++ , e10 *= 10 )
                    a[ndigit] += (str[i-1-k] -'0') * e10 ;
                ndigit++;
                j = 0 ;
            }

    if( j > 0 )
    {
        for( k = 0 , e10 = 1 ; k < j ; k++ , e10 *= 10 )
            a[ndigit] += ( str[i-1-k] -'0' ) * e10 ;
        ndigit++;
    }

    printf("There are %d numbers in this line,they are:\n",ndigit);
    for( j = 0 ; j < ndigit ; j++ )
        printf("%d " , a[j] );
    putchar('\n');

    return 0;
}
```

下面重写这个题目的代码

首先需要对题目做适当修正

输入一行字符（少于 80 个），其中含有十进制数字字符和其他字符。例如：

A123x456 17960?302tab5876

其中连续的数字的个数皆不超过 9 个。将其中各连续的数字均作为一个整数依次存储。

统计共有多少个整数，并输出这些数。

```
01.#include <stdio.h>
02.#define LEN 80
03.int main( void )
```

```
04.{
05.     char str[LEN];
06.
07.     puts("输入一行字符");
08.     fgets(str,LEN,stdin);
09.
10.     return 0;
11.}
```

这是对输入的基本考虑

其中的 stdin 表示从标准输入设备输入

这段代码没有考虑输入出现错误的情况，因此从更严格的意义上讲有些瑕疵。

更安全的写法是：

```
01.#include <stdio.h>
02.#define LEN 80
03.int main( void )
04.{
05.     char str[LEN];
06.
07.     puts("输入一行字符");
08.     if( fgets( str , LEN , stdin ) == NULL )
09.     {
10.          return 1;
11.     }
12.
13.     return 0;
14.}
```

这可以保证程序不至于处理 str 没有正确输入的情况。

求解问题的关键在于找到数字开始和结束的位置，所以

```
01.     char * begin = NULL , //数字开始位置
02.               * end     = str       ; //数字结束位置
```

由于每次寻找数字开始位置总是从上次数字结束位置开始，所以设 end 的初值为 str。

通过函数查找字符串中数字开始的位置和结束位置。

```
#include <stdio.h>
#define LEN 80
char * find_begin ( char * );
```

```
char * find_end      ( char * );
int main( void )
{
    char str[LEN];
    char * begin = NULL , //数字开始位置
           * end      = str    ; //数字结束位置

    puts("输入一行字符");
    if( fgets( str , LEN , stdin ) == NULL )
        return 1;

    while( ( begin = find_begin ( end ) )!= NULL    )
    {
        end = find_end ( begin ) ;

        {     //测试
            char *p = begin ;
            while( p < end )
            {
                putchar(*p++);
            }
        }
    }

    return 0;
}
//返回数字结束位置
char * find_end (    char *s    )
{
    while( '0' <= *s && *s <= '9' )
    {
        s++ ;
    }
    return s ;
}
```

```
//返回数字开始位置。若无数字，返回 NULL
char * find_begin ( char *s )
{
    while( ! ( '0' <= *s && *s <= '9' )   )
    {
        if( *s == '\0')
        {
            return NULL;
        }
        s ++ ;
    }
    return s;
}
```

存放整数的数组的尺寸至少应该为 LEN/2：

```
    unsigned value[LEN/2]   ,
                num = 0                ;
```

P122

17 编写一个函数，实现两个字符串的比较。即自己写一个 strcmp 函数，函数原型为：
strcmp(char *p1,char*p2)
设 p1 指向字符串 s1，p2 指向字符串 s2。要求当 s1=s2 时，返回值为 0；若 s1≠s2，返回它们二者第一个不同字符的 ASCII 码差值（如"BOY"与 BAD，第 2 个字母不同，"O"与"A"之差为 79-65=14）；如果 s1>s2，则输出正值，如果 s1<s2，则输出负值

评：首先要说的是“strcmp(char *p1,char*p2)”，这不是函数原型。因为函数原型包括返回值类型。

其次“自己写一个 strcmp 函数”这个要求过分，因为这个函数是一个库函数。库函数的名字是保留的，自定义函数不应该使用库函数的名字。

此外“s1=s2”、“s1<s2”这种描述方法不严格。至少在 C 代码层面来看是错误的。

P122

```
strcmp(char *p1,char *p2)
{.....
}
```

评：这个名字与标准库相冲突

此外没有说明函数返回值类型也是陈腐过时的写法

P122~123

```
#include <stdio.h>
int main( )
{int m;
char str1[20],str2[20],*p1,*p2;
printf("input two strings:\n");
scanf("%s",str1);
scanf("%s",str2);
p1=&str1[0];
p2=&str2[0];
m=strcmp(p1,p2);
printf("result:%d,\n",m);
return 0;
}
strcmp(char *p1,char *p2)
{int i;
i=0;
while(*(p1+i)==*(p2+i))
    if(*(p1+i++)=='\0')return(0);
return(*(p1+i)-*(p2+i));
}
```

主要有三个问题

1.没写函数类型声明

2.使用库函数名 strcmp

3.函数定义没写函数返回值类型，这在 C99 中是错误的

其他

01.p1=&str1[0];

02.p2=&str2[0];

怎么看都觉得无聊

m 更无聊

p123

18.编一程序，输入月份号，输出该月的英文月名.例如，输入“3”，则输出“March”，要求用指针数组处理。

```
01.#include <stdio.h>
02.int main()
03.{char *month_name[13]={"illegal month","January","Februry","March","April","May","June",
04.          "July","August","Septemper","October","November","December"};
05.int n;
```

```
06.printf("input month:\n");
07.scanf("%d",&n);
08.if(n<=12&&n>=1)
09.     printf("It is %s.\n",*(month_name+n));
10.else
11.     printf("It is wrong.\n");
12.return 0;
13.}
```

评：这段代码的毛病在于 month_name 数组有 13 个元素，而第一个元素没用

p124

19.(1)编写一个函数 new，对 n 个字符开辟连续的存储空间，此函数应返回一个指针(地址)，指向字符串开始的空间。new(n)表示分配 n 个字节的内存空间，见图 8.4。

(2)写一函数 free，将前面用 new 函数占用的空间释放。free(p)表示将 p(地址)指向的单元以后的内存段释放。

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p124

评：所谓返回指向“字符串”开始的空间是错误的说法，因为为 n 个字符申请空间并不意味着存储的就是字符串。

要求写一个名为 free 函数也是错误的，因为这是保留的库函数的名字。

p124

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

02.#define NEWSIZE 1000

03.char newbuf[NEWSIZE];

04.char *newp=newbuf;

05.char *new( int n )

06.     {if(newp+n<=newbuf+NEWSIZE)

07.          {newp+=n;

08.           return(newp-n);
```

```
09.          }

10.          else

11.              return(NULL);

12.    }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p124

评：

01.#include <stdio.h>

只有一个作用，后面代码使用了 NULL。

感觉很别扭

其实写为

01.#include <stddef.h>

更地道些

01.03.char newbuf[NEWSIZE];

02.04.char *newp=newbuf;

照猫画虎

可惜画虎不成反类犬

作者完全不懂得应该把它们封装起来

06.   {if(newp+n<=newbuf+NEWSIZE)

这是一个似是而非的错误写法。

正确的写法应该是

01.{

02.   if(n <= newbuf + NEWSIZE - newp)

p124~125

```
#include <stdio.h>
#define NEWSIZE 1000
cha newbuf[NEWSIZE];
char *newp=newbuf;
void free(char *p)
    {if(p>=newbuf&&p<newbuf+NEWSIZE)
        newp=p;
```

```
    }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p124~125

评：

free()函数定义居然另写了一段，完全丧失了意义。更滑稽的是又重新写了一遍

01.01.#include <stdio.h>

02.02.#define NEWSIZE 1000

03.03.cha newbuf[NEWSIZE];

04.04.char *newp=newbuf;

这就使人完全不清楚到底在写什么东西了。

此外 free()函数与库函数重名

这也是一个错误。

P125

用指向指针的指针的方法对 5 个字符串排序并输出

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p125

不看代码

无论如何猜不出到底要求做什么

P125

```
13.p=pstr;
14.sort(p);
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p125

巨无聊

P125

```
21.void sort(char ** p)
22.{ int i,j;
23.  char * temp;
24.  for(i=0;i<5;i++)
25.    {for(j=i+1;j<5;j++)
26.      {if(strcmp(*(p+i),*(p+j))>0)
27.         {temp=*(p+i);
28.         *(p+i)=*(p+j);
```

```
29.         *(p+j)=temp;
30.         }
31.     }
32.   }
33.}
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p125

参数不全
算法古怪
有点像选择法
但却大量交换

126

用指向指针的指针的方法对 n 个整数排序并输出。要求将排序单独写成一个函数。n 个整数在主函数中输入，最后在主函数中输出.——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p126

“n 个整数在主函数中输入，最后在主函数中输出”这种要求非常荒诞无理，仿佛是在要求只能把代码写烂，不许写好一样。除了绑架代码写作者写出糟糕别扭的代码之外，没有任何价值。作为一个习题提出这样的要求，是误导学习者。

126

```
01.04.int i,n,data[20],**p,*pstr[20];
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p126

很显然，代码把 data 这个具有 20 个 int 类型元素的数组作为输入的“n 个整数”的存储空间。然而题目并没有明确输入的整数不超过 20 个，所以这个属于自欺欺人的做法。

126

```
01.04.int i,n,data[20],**p,*pstr[20];
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p126

pstr 这个名字荒唐的很够层次。字面上看这应该是一个与字符串有关的指针，但实际上这个标识符和字符串八竿子打不着。这比那种简单的、无意义的单字符变量名更烂。单字符变量名仅仅会让人费解，但 pstr 这种名字是直接让人误解。

126

```
07.for(i=0;i<n;i++)
08.    pstr[i]=&data[i];
09.printf("input %d integer numbers:",n);
10.for(i=0;i<n;i++)
11.    scanf("%d",pstr[i]);
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p126

这几行代码其实可以简单地通过一个循环实现。

```
printf("input %d integer numbers:",n);
for( i = 0 ; i < n ; i ++ )
    scanf("%d",pstr[i] =&data[i] );
```

126

```
12.p=pstr;
13.sort(p,n);
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p126

无聊。

127

```
21.void sort(int **p,int n)
22.{int i,j,*temp;
23.for(i=0;i<n-1;i++)
24.    {for(j=i+1;j<n;j++)
25.        {if(**(p+i)>**(p+j))
26.            {temp=*(p+i);
27.             *(p+i)=*(p+j);
28.             *(p+j)=temp;
29.            }
30.        }
31.    }
32.}
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p127

这段代码的主要问题是算法低效。

128 第九章

1．定义一个结构体变量（包括年、月、日）。计算该日在本年中是第几天，注意闰年问题。

仅仅“定义一个结构体变量”是无法计算“该日在本年中是第几天”的

题目没有交代这个变量值的来源

128

```
#include<stdio.h>
struct
{int year;
int month;
int day;
}date;
int main()
{int days;
printf("input year,month,day:");
scanf("%d,%d,%d",&date.year,&date.month,&date.day);
switch(date.month)
{case 1: days=date.day;                break;
   case 2: days=date.day+31;           break;
   case 3: days=date.day+59;           break;
   case 4: days=date.day+90;           break;
   case 5: days=date.day+120;          break;
   case 6: days=date.day+31;           break;
   case 7: days=date.day+181;          break;
   case 8: days=date.day+212;          break;
   case 9: days=date.day+243;          break;
   case 10: days=date.day+273;         break;
   case 11: days=date.day+304;         break;
   case 12: days=date.day+334;         break;
}
if((date.year%4==0&&date.year%100!=0
```

```
        ||date.year%400==0)&&date.month>=3) days+=1;
    printf("%d/%d is the %dth day in%d.",date.month,date.day,days,date.year);
    return 0;
    }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p128

这段代码的结构给人的感觉是莫名其妙。它定义了一个外部变量 date，但是整个程序却只有一个 main()。换句话说，这个 date 从定义位置来看貌似是给许多函数使用的，但是实际上却只有一个函数在使用它。这种自相矛盾显得很滑稽。

128

```
11.switch(date.month)
12.{ case 1: days=date.day;          break;
13.  case 2: days=date.day+31;       break;
14.  case 3: days=date.day+59;       break;
15.  case 4: days=date.day+90;       break;
16.  case 5: days=date.day+120;      break;
17.  case 6: days=date.day+151;      break;
18.  case 7: days=date.day+181;      break;
19.  case 8: days=date.day+212;      break;
20.  case 9: days=date.day+243;      break;
21.  case 10: days=date.day+273;     break;
22.  case 11: days=date.day+304;     break;
23.  case 12: days=date.day+334;     break;
24.}
```

这段代码写得极其笨拙，这不是让计算机算，是人自己在算

129

```
25.if((date.year%4==0&&date.year%100!=0
26.         ||date.year%400==0)&&date.month>=3) days+=1;
```

正常的思路应该是

```
if( date.month > 2    &&
    (date.year%4==0&&date.year%100!=0
       ||date.year%400==0) ) days+=1;
```

这个表达式过于臃长复杂

判断是否闰年应该由一个函数或宏实现

129

```
解法二
#include<stdio.h>
struct
        {int year;
         int month;
         int day;
        }date;
int main()
    {int i , days;
     int day_tab[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
     printf("input year,month,day:");
     scanf("%d,%d,%d",&date.year,&date.month,&date.day);
     days=0;
     for(i=1;i<date.month;i++)
         days=days+day_tab[i];
     days=days+date.day ;
     if((date.year%4==0&&date.year%100!=0|date.year%400==0)&&date.month>=3)
         days=days+1;
     printf("%d/%d is the %dth day in%d.\n",date.month,date.day,days,date.year);
     return 0;
    }   ——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p129
```

因为偷懒把变量定义写在不恰当的位置会导致代码整体结构失和，因小失大。
存储 12 个月每个月的天数，只需要 12 个元素的数组。毫无意义地定义有 13 个元素的数组是因为对 C 语言数组下标从 0 开始的特性掌握不够，但又不肯认真掌握，而强迫 C 语言迁就自己对 C 的不理解。

129~130

写一个函数 days，实现第 1 题的计算。由主函数将年、月、日传递给 days 函数，计算后将日子输出传回主函数输出。

```
02.struct y_m_d
03.     {/* ……*/
06.     }date;   //由于变量定义的位置，这个 date 在说，其他函数都可以直接使用我
```

```
07.int main()
08.             {/* ……*/
                 days(date) //这里在说，由我来告诉 days()函数 date 的值
                 /* ……*/
12.             }
13.
14.int days(struct y_m_d date1)//无意义地准备了一个 date1，并费时地接受 main()转发的 date
                                //的值。但其实这个函数是可以直接使用 date 变量的。
15.{ /* ……*/
33.}   ——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p129~130
```

这个代码结构让人哭笑不得。

```
08.          {int days(struct y_m_d date1);  //定义 date1 为结构体变量，类型为 struct y_m_d
```

这里是一个函数类型声明，说明的是 days 这个标识符的含义。根本不存在 date1 的定义。实际上这里的 date1 写不写都可以。

main()函数明显缺少 return 0;语句。如果是在 C89 标准下编译，main()的返回值是不确定的。在需要用的这个返回值的场合，这个错误极其严重。

130~131

```
        #include<stdio.h>
struct y_m_d
    {int year;
     int month;
     int day;
    }[color=Red]date[/color];   //位置不当
int main()
    {[color=Red]int days(int year,int month,int day);
     int days(int ,int ,int );[/color]                          //唠唠叨叨，同一件事反复说了两遍
    [color=Red] int day_sum;[/color]                            //多余的变量
     printf("input year,month,day:");
     scanf("%d,%d,%d",&date.year,&date.month,&date.day);
     day_sum=days(date.year,date.month,date.day);
     printf("%d/%d is the %dth day in%d.\n",date.month,date.day,day_sum,date.year);
     return 0;
    }
```

```
int days(int year,int month,int day) //点金成铁的接口设计，糟蹋结构体类型的范例，不辞辛苦地把代码弄糟
{int day_sum,i;
 int day_tab[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};//多了一个没有用的元素
 day_sum=0;
 for( i = 1 ; i < month ; i++ )
     day_sum+=day_tab[ i ];
 day_sum+=day ;
 if((year%4==0&&year%100!=0||year%400==0)&&month>=3)
     day_sum+=1;
 return(day_sum);
}
```

P131

3.编写一个函数 print,输出一个学生的成绩数组，该数组中有 5 个学生的数据记录，每个记录包括 num、name、sore[3]，用主函数输入这些记录，用 print 函数输出这些记录。

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p131

没人搞得清“每个记录包括 num、name、sore[3]”是在说什么。

P 131~132

这是一个错误的代码。首先

```
01.06.          int score[4];
```

这个“4”莫名其妙，后面代码根本没有用到这数组成员的第四个元素。而且这与题目中的“sore[3]”也根本对不上（尽管题目本身也有问题）。

```
01.07.          }stu[N];
```

这里随手定义了一个外部数组，这个数组在后面任何地方都可以使用。效果是把所有的代码都紧密地连接了起来，这通常是一种恶劣的风格

```
01.09.{void print(struct student stu[6]);
```

这个函数类型声明中的“6”同样莫名其妙不知所云。比石头里跳出个孙悟空还让人惊诧。

```
10.int i,j ;
11.for(i=0;i<N;i++)
12.     {printf("\ninput score of student %d:\n",i+1);
13.      printf("NO.:");
```

```
14.        scanf("%s",stu[i].num);
15.        printf("name:");
16.        scanf("%s",stu[i].name);
17.        for(j=0;j<3;j++)
18.           {printf("score%d:",j+1);
19.             scanf("%d",&stu[i].score[j]);
20.           }
21.         printf("\n");
22.        }
```

在 main()中输入数据是一个馊主意。

```
23.print(stu);
```

这条语句不但有煞有介事的成分，也有装傻充愣的成分。

```
26.void print(struct student stu[6])
```

如果这里的“6”写得是“5”或“N”，说明代码作者概念不清，但这里居然写的是“6”，只能说明代码作者很 2。
如果写“5”或“N”说明代码作者的意图可能是向这个函数传递一个具有 5 个元素或 N 个元素的数组。
但是这个愿望是会落空的，因为编译器根本对形参后面第一个[]内的数视而不见。
但是这里居然写的是“6”，搞不清这个“6”是哪儿来的“天外飞仙”。

```
29.for(i=0;i<N;i++)
30.       {printf("%5s%10s",stu[i].num,stu[i].name);
31.        for(j=0;j<3;j++)
32.        printf("%9d",stu[i].score[j]);
33.        printf("\n");
34.       }
```

这里的 N 和 3 都属于天外飞仙，不知道从哪里来的。

P 133~134

```
canf("%s",stu[i].num);
rintf("name:")
```

可是在 134 页居然声称

```
运行情况与第 3 题相同。
```

公然扯谎

P 134~136

```
#include <stdio.h>
#define N 10

struct student
     {char num[6];
      char name[8];
      float score[3];
      float avr;
     }stu[N];

int main()
{int i,j,maxi;
float sum,max,average;
   //输入数据
for(i=0;i<N;i++)
   {printf("input scores of student %d:\n",i+1);
     printf("NO.:");
     scanf("%s",stu[i].num);
     printf("name:");
     scanf("%s",stu[i].name);
     for(j=0;j<3;j++)
        {printf("score %d:",j+1);
         scanf("%f", &stu[i].score[j]);
        }
     }
//计算
average=0;
max=0;
maxi=0;
for(i=0;i<3;i++)
     {sum=0;
      for(j=0;j<3;j++)
         sum+=stu[i].score[j];
```

```
        stu[i].avr=sum/3.0;
        average+=stu[i].avr;
        if(sum>max)
            {max=sum;
             maxi=i;
            }
        }
    average/=N;
    //输出
    printf("NO. name score1 score2 score3 average\n");
    for(i=0;i<N;i++)
      {printf("%5s%10s",stu[i].num, stu[i].name);
       for(j=0;j<3;j++)
            printf("%9.2f",stu[i].score[j]);
            printf("%8.2f\n",stu[i].avr);
      }
    printf("average=%5.2f\n",average);
    printf("The highest score is : student %s,%s\n",stu[maxi].num,stu[maxi].name);
    printf("his scores are:%6.2f,%6.2f,%6.2f,average:%5.2f.\n",
              stu[maxi].score[0],stu[maxi].score[1],stu[maxi].score[2],stu[maxi].avr);
    return 0;
    }
```

P136 页的运行结果是伪造的
这个代码不可能得到 136 页的运行结果
代码的结构莫名其妙，因为整段代码只有一个函数 main()，没有任何理由设置外部变量。

```
01.24.          //计算
02.25.          average=0;
03.26.          max=0;
04.27.          maxi=0;
05.28.          for(i=0;i<3;i++)
06.29.          {sum=0;
07.30.          for(j=0;j<3;j++)
08.31.          sum+=stu[i].score[j];
09.32.          stu[i].avr=sum/3.0;
10.33.          average+=stu[i].avr;
11.34.          if(sum>max)
```

```
12.35.          {max=sum;
13.36.           maxi=i;
14.37.           }
15.38.          }
16.39.          average/=N;
```

这段代码能够记住第一个 sum 最高值，并用 maxi 记录其序号 i。但问题在于如果后面某个 sum 等于 max 的话应该如何处理呢？代码的功能是把第二个忽视，但这并非是题目的要求。

```
01.40.          //输出
02.41.          printf("NO. name score1 score2 score3 average\n");
03.42.          for(i=0;i<N;i++)
04.43.          {printf("%5s%10s",stu[i].num, stu[i].name);
05.44.          for(j=0;j<3;j++)
06.45.          printf("%9.2f",stu[i].score[j]);
07.46.          printf("%8.2f\n",stu[i].avr);
08.47.          }
09.48.          printf("average=%5.2f\n",average);
10.49.          printf("The highest score is : student %s,%s\n",stu[maxi].num,stu[maxi].name);
11.50.          printf("his scores are:%6.2f,%6.2f,%6.2f,average:%5.2f.\n",
12.51.          stu[maxi].score[0],stu[maxi].score[1],stu[maxi].score[2],stu[maxi].avr);
```

输出部分更为混乱。41.行~47.行根本不是问题要求。这属于“多做之过”。程序应该完成且只应该完成所要求的功能，擅自增加功能是画蛇添足，也是一种错误。

49．行~50.行不可能是正确的，原因前面已经解释过了。需要说一下的是

```
01.50.          printf("his scores are:%6.2f,%6.2f,%6.2f,average:%5.2f.\n",
02.51.          stu[maxi].score[0],stu[maxi].score[1],stu[maxi].score[2],stu[maxi].avr);
```

用这种方法（stu[maxi].score[0],stu[maxi].score[1],stu[maxi].score[2]）输出数组的各个元素，可谓笨拙之极，这使代码完全丧失了可维护性。

P 147

```
#include <malloc.h>
```

古老过时已经腐朽的写法

P 148~149

```
{struct student student *creat();
 struct student student *del(student *,long);
 struct student student *insert(student *,student *);
```

```
    void print(struct student *head);
```

这种地沟油代码根本无法通过编译
可老谭在 150 页居然说有运行结果
这是在欺骗读者

struct student *creat()
这是一个很不规范的写法。
n=0;
它用了一个蹩脚的外部变量来表示节点个数。n 蹩脚在它与链表不是紧密结合为一体，而是分崩离析各自为政，如果 n 被意外改变，程序将错得一塌糊涂。

```
    p1=p2=(struct student*)malloc(LEN);
```

这句有些昏头昏脑。它不分青红皂白地申请了一块内存——malloc(LEN)，但是却还没弄清楚是否真的需要这块内存，然后就向结构体中写数据
scanf("%ld,%f",&p1->num,&p1->score);
如果输入的数据使 p1->num 为 0，会导致内存泄露
p1=p2=(struct student*)malloc(LEN);
和
p1=(struct student*)malloc(LEN);
没考虑 malloc()函数返回值为 NULL 即没考虑内存可能失败的情况也是错误的
head=NULL;
这句不应该写，应该在定义 head 变量时初始化。

scanf("%ld,%f",&p1->num,&p1->score);
/*……*/
while(p1->num!=0)
{/*……*/
scanf("%ld,%f",&p1->num,&p1->score);
这种重复表明代码书写者根本不会写 C 代码。
p1=(struct student*)malloc(LEN);
这句最后一定会导致内存泄露。

P 149

```
    //删除结点的函数
    struct student *del(struct student *head,long num)
        {struct student *p1,*p2;
         if(head==NULL)
             {printf("\nlist null\n");
              return(head);
```

```
        }
    p1=head
    while(num!=p1->num&&p1->next!=NULL)
                                        //p1 指向的不是所要找的结点且后面还有结点
      {p2=p1;p1=p1->next;}
    if(num==p1->num)
    {if(p1==head)head=p1->next;
     else p2->next=p1->next;
     printf("delete:%ld\n",num);
     n=n-1;
    }
    else
     printf("%ld not been found!\n",num);
    return(head);
  }
```

这个函数风格更烂，而且竟然有语法错误：

```
    p1=head
```

很显然，这里缺少一个“;”。

```
while(num!=p1->num&&p1->next!=NULL)
                                        //p1 指向的不是所要找的结点且后面还有结点
      {p2=p1;p1=p1->next;}
```

这个注释更是奇葩，它居然把一个完整的语句彻底割裂开来，不知道是为了让人看得懂还是为了让人更加看不懂？通常注释即使不成功也不会对代码照成损害，但这个注释却是空前绝后地破了这个惯例。

```
                if( head == NULL )
                {printf("\nlist null\n");
                  return(head);
                }
```

这是多此一举的，也是自相矛盾的。因为调用函数自己可以判断链表是否为空，而且 head==NULL 和 n==0 都可以作为判断链表是否为空的条件，再次证明 n 那个外部变量是多余的累赘。

即使由 del()可能接受空链表，后面的语句也完全能够完成正确的功能。

```
01.     09.     while(num!=p1->num&&p1->next!=NULL)
02.10.                                  //p1 指向的不是所要找的结点且后面还有结点
03.11.        {p2=p1;p1=p1->next;}
```

体现初学者常见的怯懦，不敢更进一步让 p1 的值为 NULL，并以此作为循环的结束条件。结果只好在后面写出很猥琐的代码

```
01.19.                          if( p1 == head )
02.20.                                  head=p1->next;
```

```
03.21.                    else
04.22.                        p2->next=p1->next;
```

这里有个严重的错误，就是没有把删除节点所占用的内存释放，自己不用，也不让别人用，随手乱丢垃圾，造成内存泄露。函数调用结束后 p1 指向的内存块就像太空中的垃圾一样，无论是操作系统还是自己的程序，谁都够不着。

P 150

```
//定义输出链表的 print 函数
void print(struct student *head)
    {struct student *p;
     printf("\nNow,These %d records are:\n",n);
     p=head;
     if(head!=NULL)
        do
        {printf("%ld%5.1f\n",p->num,p->score);
         p=p->next;
        }while(p!=NULL)
    }
```

评：很滑稽的写法
首先由于 if(head!=NULL) 之前 p=head;
所以 if(head!=NULL)可以写成 if(p!=NULL)
这时再来看一下代码

```
//定义输出链表的 print 函数
void print(struct student *head)
    {struct student *p;
     printf("\nNow,These %d records are:\n",n);
     p=head;
     if(p!=NULL)
        do
        {printf("%ld%5.1f\n",p->num,p->score);
         p=p->next;
        }while(p!=NULL);
    }
```

不难发现

```
     if(p!=NULL)
```

```
        do
        {printf("%ld%5.1f\n",p->num,p->score);
          p=p->next;
        }while(p!=NULL);
```

其实不过是 while 语句的一种变态写法
这句应该写为

```
    while(p!=NULL)
    {
        printf("%ld%5.1f\n",p->num,p->score);
        p=p->next;
    }
```

再回头看一下这个函数

```
//定义输出链表的 print 函数
void print(struct student *head)
{   struct student *p;
    printf("\nNow,These %d records are:\n",n);
    p=head;
    while(p!=NULL)
    {
        printf("%ld%5.1f\n",p->num,p->score);
        p=p->next;
    }
}
```

会发现 p 这个变量是多余的，函数应该写为

```
void print(struct student *p)
{
    printf("\nNow,These %d records are:\n",n);
    while( p != NULL )
    {
        printf("%ld%5.1f\n",p->num,p->score);
        p = p -> next ;
    }
}
```

这里的 n 是一个外部变量，完全不入流。应该删除

P 154~155

```
05.struct student
06.     {int num;
07.      char name[8];
08.      struct student *next;
09.     }a[LA],b[LB];
11.int main( )
12.{struct student a[LA]={{101,"Wang"},{102,"LI"},{105,"zhang"},{106,"Wei"}};
13.    struct student b[LB]={{103,"Zhang"},{104,"Ma"},{105,"Chen"},{107,"Guo"},{108,"Lui"}};
```

写出这种代码说明头脑不清，糊涂混乱
这种代码连下流都谈不上
只能说是不入流

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

这行也很无厘头

```
18.head2=b;
```

位置很傻，显得东一榔头西一棒子。起码应该写在 29 行

```
17.head1=a;


19.printf("list A :\n");
20.for(p1=head1,i=1;i<=LA;i++)
21.     {if(i<LA)
22.          p1->next=a+i;
23.      else
24.          p1->next=NULL;
25.      printf("%4d%8s\n",p1->num,p1->name);
26.      if(i<LA)
27.          p1=p1->next;
28.     }
```

这几行代码很煞有介事。其实它的主要功能无非是

```
for( i = 0 ; i < sizeof a/sizeof *a - 1 ; i++ )
   a[ i ].next=a[ i + 1 ]

a[ i ].next=NULL;
```

head1=a;

```
30.for(p2=head2,i=1;i<=LB;i++)
31.     {if(i<LB)
32.          p2->next=b+i;
33.      else
34.          p2->next=NULL;
35.      printf("%4d%8s\n",p2->num,p2->name);
36.      if(i<LB)
37.          p2=p2->next;
38.     }
```

完全重复 20~28 行，丑陋得惨不忍睹

```
40.//对 a 链表进行删除操作
41.p1=head1;
42.while(p1!=NULL)
43.     {p2=head2;
44.      while((p1->num!=p2->num)&&(p2->next!=NULL))
45.          p2=p2->next;
46.          //使 p2 后移直到发现与 a 链表中当前的结点的学号相同或已到 b 链表最后一个结点
47.      if(p1->num==p2->num)
48.           {if(p1==head1)
49.                head1=p1->next;
50.            else
51.                {p->next=p1->next;
52.                                    //使 p->next 指向 p1 的下一个结点，即删去 p1 当前指向的节点
53.                 p1=p1->next;
54.                }
55.           }
56.      else
57.           {p=p1;p1=p1->next;}
58.     }
```

一团乱麻一样复杂的代码，源自颠三倒四的混乱思维。

它的想法是逐个查看 a 链表的节点，如果该节点的学号在 b 链表中存在，则删除该节点。这样成功地把简单的问题复杂化了——把两个链表交织在了一起。

P 156

一上来就是一只苍蝇。

```
02.#include <malloc.h>
```

这条预处理命令的用意是给出代码中调用 malloc()函数的函数类型声明。然而这是几十年前的编译器的写法。现代 C 语言中 malloc()函数的函数类型声明由 stdlib.h 提供。

P 156~157

```
04.struct student
05.{char num[6];
06.char name[8];
07.char sex[2];
08.int age;
09.struct student* next;
10.}stu[10];
```

这是代码中的第二只苍蝇——stu[10]。这是一个外部变量，但是自相矛盾的是整个源程序只有一个 main()，更滑稽的是这个数组在整个源程序中根本就没有被用到。

P 157

```
14.int i,length,iage,flag=1;
15.int find=0;
```

每次见到 flag 都仿佛能闻到代码中有种馊味。后面的 find 显然也是同样性质的变量。

P 157

```
41.p=head;
42.printf("\n NO.  name  sex  age\n");
43.while(p!=NULL)
44.    {printf("%4s%8s%6s%6d\n",p->num, p->name, p->sex, p->age);
45.     p=p->next;
46.    }
```

这段代码几乎和

```
72.p=head;
73.printf("\n NO.name sex age\n");
74.while(p!=NULL)
75.{printf("%4s%8s",p->num,p->name);
76.  printf("%6s%6d\n",p->sex,p->age);
77.  p=p->next;
78.}
```

一模一样，是一种拙劣的败笔，甚至是比一模一样还要拙劣的败笔。

P 157

```
23.//建立链表
24.for(i=0;i<length;i++)
25.{p=(struct student *)malloc(LEN);
26.    if(i==0)
27.        head=pt=p;
28.    else
29.        pt->next=p;
30.    pt=p;
31.    printf("NO:");
32.    scanf("%s",p->num);
33.    printf("name:");
34.    scanf("%s",p->name);
35.    printf("sex:");
36.    scanf("%s",p->sex);
37.    printf("age:");
38.    scanf("%d",&p->age);
39.}
40.p->next=NULL;
```

正如前面所指出的那样，链表根本就不应该在建立之前指定 length。因此从根本上来说，建立链表根本就不应该使用 for 语句。这里之所以出现如此怪异的建立方法是因为代码作者不懂得正确的建立方法。

在代码细节上这段代码也有很多问题，例如没有处理 malloc()返回值为 NULL 的情况，此外

```
26.    if(i==0)
27.        head=pt=p;
28.    else
29.        pt->next=p;
30.    pt=p;
```

中的 27.行的 pt=p 是多此一举的画蛇添足，因为在 30.行有一句 pt=p;，无论如何都会被执行。

P 157

```
48.//删除结点
49.printf("input    age:");
50.scanf("%d",&iage);
51.pt=head;
52.p=pt;
53.if(pt->age==iage)
54.        {p=pt->next;
```

```
55.        head=pt=p;
56.        find=1;
57.      }
58.else
59.      pt=pt->next;
```

这一段处理的是第一个节点。第一个节点可能需要被删除也可能不需要被删除。紧接着的一段代码

```
60.while(pt!=NULL)
61.  {if(pt->age==iage)
62.    {p->next=pt->next;
63.    find=1;
64.    }
65.    else
66.    p=pt;
67.    pt=pt->next;
68.  }
```

处理的是第一个节点之后亦即非 head 所指向的各个节点。然而在第一个节点被删除的情况下，第二个节点就会成为第一个结点，这种情况下必须继续把第二个节点作为 head 所指向的那个节点来处理。这时如果把第二个节点作为第一个节点之后的后续节点就会发生错误。

所以，不言而喻，这段关于删除的代码是绝对错误的。这一点不需要运行测试就能发现。

P159

从键盘输入一个字符串，将其中的小写字母全部转换成大写字母，然后输出到一个磁盘文件“test”中保存。输入的字符串以“！”结束。

评：“ 字符串以“!”结束”，算是奇闻

P159

题目的要求是“输出到一个磁盘文件‘test’中保存”，可是代码中却是

```
8.            if((fp=fopen("a1","w"))==NULL)
```

显然“码”不对题。

P159

```
10.                exit(0);
```

那个 0 似是而非。

P159

13. gets(str);

这是一个错误的写法。因为 gets()函数的作用是输入一行字符，但是题目中并没有提到输入“!”之前没有换行。

P159

16. str[i]=str[i]-32;

32 看着很扎眼

P159

21. fp=fopen("a1","r");
22. fgets(str,strlen(str)+1,fp);
23. printf("%s\n",str);
24. fclose(fp);

这属于多做之过。完成问题根本没要求的功能也是一种错误。其中的第 22.行更是错得荒谬无边。

P160

有两个磁盘文件”A”和”B”,各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件”C”中。

——谭浩强 ， 《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p160

题目不是一般的无聊
是挖空心思的无聊

P160~161

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
FILE *fp;
```

```
int i,j,n,i1;
char c[100],t ,ch;
if((fp=fopen("a1","r"))==NULL)
{printf("can not open the file\n");
exit(0);
}
printf("\n file A:\n");
for(i=0;(ch=fgetc(fp))!=EOF;i++)
{
c[i]=ch;
putchar(c[i]);
}
fclose(fp);

i1=i;
if((fp=fopen("b1","r"))==NULL)
  {printf("\n can not ipen the file");
   exit(0);
  }
printf("\nfile B:\n");
for(i=i1;(ch=fgetc(fp))!=EOF;i++)
{c[i]=ch;
putchar(c[i]);
}
fclose(fp);

n=i;
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
if(c[i]>c[j])
{t=c[i];
c[i]=c[j];
c[j]=t;
}
printf("\n file C:\n");
fp=fopen("c1","w");
```

```
for(i=0;i<n;i++)
  {putc(c[i],fp);
   putchar(c[i]);
  }
  printf("\n");
  fclose(fp);
  return 0;
}
```

02.#include <stdlib.h>

典型的无病呻吟。

07. char c[100],t ,ch;

这里的“100”毫无依据，不但是天真的也是武断的。写代码不能自说自话地假设问题的条件。那个 t 根本就不应该在这里定义。ch 声明为 char 类型是有问题的。

```
08. if((fp=fopen("a1","r"))==NULL)
09.    {printf("can not open the file\n");
10.      exit(0);
11.    }
```

文不对题货不对板。题目说的是文件“A”，这里却自作主张地打开了“a1”。另外只写文件名不写文件的位置也是一种陋习。

23. exit(0);

外行写法

20. i1=i;

这句看起来没有问题，然而结合

26. for(i=i1;(ch=fgetc(fp))!=EOF;i++)

来看就能看出其中的滑稽了，因为从 20.行到 26.行无论是 i 的值还是 i1 的值都没有改变，因而 20.行的 i1=i 与 26.行的 i=i1 就如同一个人把右手中的筷子换到了左手，然后又把左手中的筷子换到了右手一样，其实筷子原本就在右手。

```
32. n=i;
33. for(i=0;i<n;i++)
34.    for(j=i+1;j<n;j++)
35.       if(c[ i ]>c[j])
36.          {t=c[ i ];
37.           c[ i ]=c[j];
38.           c[j]=t;
39.          }
```

这段代码的算法既非冒泡也非选择，有点非驴非马的味道。

41. fp=fopen("c1","w");

这行代码有两处错误，其一是打开的文件与题目要求不符，其二是没有考虑到文件打不开的情形

P161~162

```
#include <stdio.h>
struct student
{char num[10];
char name[8];
int score[3];
float ave;
}stu[5];
int main( )
    {int i,j,sum;
     FILE *fp;
     for(i=0;i<5;i++)
        {printf("\n input score of student%d:\n",i+1);
         printf("NO.:");
         scanf("%s",stu[i].num);
         printf("name:");
         scanf("%s",stu[i].name);
         sum=0;
         for(j=0;j<3;j++)
            {printf("score %d :",j+1);
             scanf("%d",&stu[i].score[j]);
             sum+=stu[i].score[j];
            }
         stu[i].ave=sum/3.0;
        }
        //将数据写入文件
        fp=fopen("stud","w");
        for(i=0;i<5;i++)
           if(fwrite(&stu[i],sizeof(struct student),1,fp)!=1)
              printf("File write error\n");
        fclose(fp);
        fp=fopen("stud","r");
        for(i=0;i<5;i++)
           {fread(&stu[i],sizeof(struct student),1,fp);
            printf("\n%s,%s,%d,%d,%d,%6.2f\n",stu[i].num,stu[i].name,stu[i].score[0],
            stu[i].score[1], stu[i].score[2] ,stu[i].ave);}
```

```
    system("PAUSE");return 0;
}
```

```
1.          struct student
2.          {char num[10];
3.           char name[8];
4.           int score[3];
5.           float ave;
6.          }stu[5];
```

许多初学者之所以在声明结构体类型时顺手定义外部变量是因为偷懒，不愿意再写一次“struct student”这样的结构体类型名称。这是贪小便宜的行为。俗话说贪小便宜吃大亏，在这里也是如此。顺手定义外部变量的后果是把整个代码结构弄得不堪入目。

```
8.              {int i,j,sum;
```

这里定义的 j 和 sum 都很无聊。从 main()全局来看，它们与问题无关。

```
25.             fp=fopen("stud","w");
```

这里没有处理文件无法打开的情况，更严重的是，由于后面使用的是 fwrite()函数写文件，打开模式明显应该是"wb"而不是"w"。

```
26.             for(i=0;i<5;i++)
27.               if(fwrite(&stu[ i ],sizeof(struct student),1,fp)!=1)
28.                 printf("File write error\n");
```

这段代码最大的问题是它检查了写入不正常的情况(fwrite(&stu[i],sizeof(struct student),1,fp)!=1)，但除了输出“File write error”之外没有做其他任何处理。实际上在这种情况下后面的代码以及失去意义。
&stu[i]这个写法其实可以更简洁地写为 stu + i 。另外，循环语句在这里是根本没有必要的，fwrite()函数可以写入成组的数据：
fwrite(stu+i,sizeof(struct student),5,fp);

下面这段代码的作用是进行测试。

```
01.30.          fp=fopen("stud","r");

02.31.          for(i=0;i<5;i++)

03.32.            {fread(&stu[ i ],sizeof(struct student),1,fp);

04.33.             printf("\n%s,%s,%d,%d,%d,%6.2f\n",stu[ i ].num,stu[ i ].name,stu[ i ].score[0],

05.34.             stu[ i ].score[1], stu[ i ].score[2] ,stu[ i ].ave);}
```

它有这样一些问题：
首先

```
01.34.             stu[i].score[1], stu[i].score[2] ,stu[i].ave);}
```

把“}”写在行末，就如同把袜子穿在了手上，寻常人等是没有这般勇气的。

01.32. {fread(&stu[i],sizeof(struct student),1,fp);
同样根本用不着放在循环语句中，可以直接
fread(stu , sizeof(struct student),5,fp);
就足以完成任务。
01.33. printf("\n%s,%s,%d,%d,%d,%6.2f\n",stu[i].num,stu[i].name,stu[i].score[0],

02.34. stu[i].score[1], stu[i].score[2] ,stu[i].ave);
把所有的东西都塞在一个函数调用之中是懒婆娘的作风，就如同把内衣外衣袜子一股脑地塞进一只箱子。score 这样数组，居然分别写出然后一字排开 stu[i].score[0] ,stu[i].score[1], stu[i].score[2]，更是笨拙之中尤其之笨拙之写法。
最后要说的是，这段测试代码是无效的。因为在
01.27. if(fwrite(&stu[i],sizeof(struct student),1,fp)!=1)
之前，stu 数组中已经写好了数据，即使在 fwrite()没起作用的情况下，比如一个数据都没写进去，那么后面的
01.32. {fread(&stu[i],sizeof(struct student),1,fp);
同样不起作用。但是这时数组中的数据还在，依然能够输出。所以这段测试代码是一种自欺欺人的伪测试。

P163~164

```
#include <stdio.h>
#define SIZE 5
struct student
{char name[10];
int num;
int score[3];
float ave;
}stud[SIZE];

int main( )
    {void save(void);
     int i;
     float sum[SIZE];
     FILE *fp1;
     for(i=0;i<SIZE;i++)
        {scanf("%s%d%d%d%d",stud[i].name,&stud[i].num,&stud[i].score[0],
                    &stud[i].score[1],&stud[i].score[2]);
         sum[i]=stud[i].score[0]+stud[i].score[1]+stud[i].score[2];
         stud[i].ave=sum[i]/3;
        }
     save();
     fp1=fopen("stu.dat","rb");
     printf("\n name NO. score1 score2 score3 ave\n");
     printf("-------------------------------------------------------\n");
```

```
                                        //输出表头
    for(i=0;i<SIZE;i++)
       {fread(&stud[i],sizeof(struct student),1,fp1);
        printf("%-10s%3d%7d%7d%7d%8.2f\n",stud[i].name,stud[i].num,
        stud[i].score[0],stud[i].score[1],stud[i].score[2] ,stud[i].ave);
       }
    fclose(fp1);
    system("PAUSE");return 0;
}

void save(void)
{
    FILE *fp;
    int i;
    if((fp=fopen("stu.dat","wb"))==NULL)
      {printf("The file can not open\n");
       return ;
      }
     for(i=0;i<SIZE;i++)
       if(fwrite(&stud[i],sizeof(struct student),1,fp)!=1)
            {printf("File write error\n");
             return ;
            }
     fclose(fp);
}
```

这是同一问题的另一写法。这个写法更成问题。

```
2.   struct student
3.   {char name[10];
4.    int num;
5.    int score[3];
6.    float ave;
7.   }stud[SIZE];
```

这个外部变量也是贪小便宜吃大亏，为了取暖不惜烧掉房子。

```
9.              {void save(void);
```

位置不当。

```
11.             float sum[SIZE];
```

这个数组定义得莫名其妙，后面会看到它完全是多余的。

```
14.                {scanf("%s%d%d%d%d",stud[ i].name,&stud[ i].num,&stud[ i].score[0],
15.                   &stud[ i].score[1],&stud[ i].score[2]);
```

这个调用实在有些太雷人了，居然一口气写出了 3 个数组元素实参&stud[i].score[0] ，&stud[i].score[1] ，

&stud[i].score[2]。看来 C 语言根本就不需要发明循环语句。
另外，这里的 scanf()调用是“裸体”的，对用户没有做任何提示，这种对用户一点也不友好的编程作风非常不值得效法。

16. sum[i]=stud[i].score[0]+stud[i].score[1]+stud[i].score[2];

02.17. stud[i].ave=sum[i]/3;
16.行完全多余，因为 17.可以写为
01.stud[i].ave=(stud[i].score[0]+stud[i].score[1]+stud[i].score[2])/3.0F;
完全没必要为记录，stud[i].score[0]+stud[i].score[1]+stud[i].score[2]而定义一个数组，因为 stud[i].score[0]+stud[i].score[1]+stud[i].score[2]这个值只是临时性的一次性使用。难道有人会为一次性使用的卫生纸制作一个精致且耐用的包装吗？

32. void save(void)
这个函数第一个错误是没有任何参数，它究竟如何知道应该打开哪个文件并向其中写什么呢？看来是生而知之的。凡是这种生而知之的函数基本上都是废品函数，因为只能打开某个特定文件并向其中写特定的数据（而且必须是外部变量），那么如果需要写到另一个文件就只能再克隆一个几乎一模一样的函数。从这个意义上来说，这种函数是废品函数，因为它没有任何适应能力和通用性。

36. if((fp=fopen("stu.dat","wb"))==NULL)
这句显然文不对题。题目要求写入的文件并非 stu.dat。当无法打开文件时
37. {printf("The file can not open\n");
38. return ;
39. }
诡异的是那个 return;。因为无法打开文件应该视为程序执行过程中的一种错误或异常，这时应该对此进行必要的处理之后程序才能继续执行，如果无法处理则直接体面地结束程序。可是现在错误已经发生了，但是这句 return;却像什么事都没发生一样心平气和脸不变色心不跳地返回 main()，显然后面将无法保证程序的正确运行。这时程序往往“死”得很难看。

01. 40. for(i=0;i<SIZE;i++)

02.41. if(fwrite(&stud[i],sizeof(struct student),1,fp)!=1)

03.42. {printf("File write error\n");

04.43. return ;

05.44. }
这里再次重复了一次前面所说的错误：在异常情况下假装正常，返回调用处。

回到 main()之后是一段测试用的代码，这与题目要求无关。
20. fp1=fopen("stu.dat","rb");
再度打开文件，但却不对打不开的情况进行处理。

01.21. printf("\n name NO. score1 score2 score3 ave\n");

02.22. printf("----------\n");

03.23. //输出表头

在 main()这么重要的地方尽弄些鸡毛蒜皮。因而弄出

01.24. for(i=0;i<SIZE;i++)

02.25. {fread(&stud[i],sizeof(struct student),1,fp1);

03.26. printf("%-10s%3d%7d%7d%7d%8.2f\n",stud[i].name,stud[i].num,

04.27. stud[i].score[0],stud[i].score[1],stud[i].score[2] ,stud[i].ave);

05.28. }

这样的一地鸡毛。除此之外还有不少鸡屎：

01.fread(&stud[i],sizeof(struct student),1,fp1);

，没有处理读入异常情况，而且由于 stud 是数组，这里本来就不需要循环语句。至于

01.26. printf("%-10s%3d%7d%7d%7d%8.2f\n",stud[i].name,stud[i].num,

02.27. stud[i].score[0],stud[i].score[1],stud[i].score[2] ,stud[i].ave);

这样的语句，稍有自尊的程序员都是写不出手的。把数组元素一字排开挤在一条 scanf()调用中，这简直是丑疯了。

这段代码本质上根本无法检测数据是否正确写入文件。因为若没写入文件，一般也无法正确地读出，但是由于这里输出的是 stud 数组，数组中原本就有数据，即使没有写入和再度读出，依然能输出相同的数据。

P164

将上题 stud 文件中的学生数据按平均分进行排序处理，并将已排序的学生数据存入一个新文件 stu-sort 中。

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p164

题目多少有些无聊，本质上无非还是读写文件。因为无聊，所以要加点料，这回加的作料是排序。

P165~166

```
#include <stdio.h>
#include <stdlib.h>
#define N 10
```

```
struct student
{char num[10];
char name[8];
int score[3];
float ave;
}st[N],temp;

int main( )
{FILE *fp;
int i,j,n;

//读文件
if((fp=fopen("stud","r"))==NULL)
   {printf("can not open the file");
    exit(0);
   }
  printf("\n File 'stud':");
for(i=0;fread(&st[i],sizeof(struct student),1,
fp)!=0;i++)
{printf("\n%8s%8s",st[i].num,st[i].name);
for(j=0;j<3;j++)
  printf("%8d",st[i].score[j]);
printf("%10.2f",st[i].ave);
}
printf("\n");
fclose(fp);
n=i;

//排序
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
if(st[i].ave<st[j].ave)
{temp=st[i];
st[i]=st[j];
st[j]=temp;
}

//输出
printf("\nNow:");
fp=fopen("stu-sort","w");
for(i=0;i<n;i++)
{fwrite(&st[i],sizeof(struct student),1,fp);
printf("\n%8s%8s",st[i].num,st[i].name);
for(j=0;j<3;j++)
```

```
  printf("%8d",st[i].score[j]);
printf("%10.2f",st[i].ave);
}
printf("\n");
fclose(fp);
return 0;
}
```

2. #define N 10

这个 10 不知道是从哪来的，题目中根本就没有提。自作主张地添加条件是一种恶习。看到这里可以得到一个结论，要么题目条件不完全因而是错误的，要么代码是错误的。

```
3.  struct student
4.  {char num[10];
5.   char name[8];
6.   int score[3];
7.   float ave;
8.  }st[N],temp;
```

纵观整段代码，只有一个 main()函数，这本身就是一种初学者的幼稚病。在只有一个函数的情况下，因为偷懒顺手定义外部变量，这是一种恶习。令人大跌眼镜的是这里居然定义了一个八竿子打不着的只是在千里之外的角落里才用得着的 temp，实在令人匪夷所思。这太雷人了也。

```
12.  //读文件
13.  if((fp=fopen("stud","r"))==NULL)
14.     {printf("can not open the file");
15.      exit(0);
16.     }
```

从后面对 fp 的使用来看，这里用"r"模式打开是错误的，应该用"rb"模式打开。
exit(0)似是而非。

```
18.  for(i=0;fread(&st[i],sizeof(struct student),1,
19.  fp)!=0;i++)
20.     {printf("\n%8s%8s",st[i].num,st[i].name);
21.      for(j=0;j<3;j++)
22.         printf("%8d",st[i].score[j]);
23.      printf("%10.2f",st[i].ave);
24.     }
25.  printf("\n");
26.  fclose(fp);
27.  n=i;
```

把读入数据和输出搅在一起，不是明智写法。
再有，这里的 i 的作用是记数，用 for 语句极不自然。最后再把 i 的值赋给 n，是很笨拙的写法。

```
01.28.  //排序
```

```
02.29.          for(i=0;i<n;i++)
03.30.            for(j=i+1;j<n;j++)
04.31.              if(st[i].ave<st[j].ave)
05.32.                {temp=st[i];
06.33.                 st[i]=st[j];
07.34.                 st[j]=temp;
08.35.                }
```

排序通常都是指从小到大排，除非另外特别说明。但是这里的排序确是一反常规地从大到小，而题目中并没有说明要求从大到小排序。

这里的算法也不够好，是一种效率很低的交换法。其中的“i<n”是一个不够精准的条件，应该写为“ i < n － 1 ”。

```
01.36.          //输出
02.37.          printf("\nNow:");
03.38.          fp=fopen("stu-sort","w");
04.39.          for(i=0;i<n;i++)
05.40.            {fwrite(&st[i],sizeof(struct student),1,fp);
06.41.             printf("\n%8s%8s",st[i].num,st[i].name);
07.42.             for(j=0;j<3;j++)
08.43.                printf("%8d",st[i].score[j]);
09.44.             printf("%10.2f",st[i].ave);
10.45.            }
11.46.          printf("\n");
12.47.          fclose(fp);
```

这段代码的问题较多，首先"w"这种打开模式是错误的，二进制文件应该用"wb"模式打开。其次，没考虑无法打开文件时如何处理。第三，用不着使用循环语句向文件写数据。第四，没有考虑写入出错情况下的处理。最后，41.~44.行完全是 20.~23.行的重复，这是烂代码的一个标志。

P168~169

```
#include <stdio.h>
#include <stdlib.h>
struct student
{char num[10];
   char name[8];
   int score[3];
   float ave;
}st[10],s;

int main( )
{FILE    *fp, * fp1 ;
```

```
int i,j,t,n;
printf("\n NO.:");
scanf("%s",s.num);
printf("name:");
scanf("%s",s.name);
printf("score1,score2,score3:");
scanf("%d,%d,%d",&s.score[0], &s.score[1], &s.score[2]);
s.ave=(s.score[0]+s.score[1]+s.score[2])/3.0;

    //从文件读数据
if((fp=fopen("stu_sort","r"))==NULL)
    {printf("can    not open file.");
      exit(0);
    }
printf("original data:\n");
     for(i=0;fread(&st[i],sizeof(struct student),1,fp)!=0;i++)
         {printf("\n%8s%8s",st[i].num,st[i].name);
            for(j=0;j<3;j++)
                 printf("%8d",st[i].score[j]);
            printf("%10.2f",st[i].ave);
         }

    n=i;
    for(t=0;st[t].ave>s.ave&&t<n;t++);

     //向文件写数据
  printf("\nNow:\n");
    fp1=fopen("sort1.dat","w");
    for(i=0;i<t;i++)
      {fwrite(&st[i],sizeof(struct student),1,fp1);
         printf("\n%8s%8s",st[i].num,st[i].name);
         for(j=0;j<3;j++)
              printf("%8d",st[i].score[j]);
         printf("%10.2f",st[i].ave);
      }
fwrite(&s,sizeof(struct student),1,fp1);
printf("\n%8s%7s%7d%7d%7d%10.2f",s.num,s.name,s.score[0],
                s.score[1],s.score[2],s.ave);

    for(i=t;i<n;i++)
      {fwrite(&st[i],sizeof(struct student),1,fp1);
         printf("\n %8s%8s",st[i].num,st[i].name);
         for(j=0;j<3;j++)
              printf("%8d",st[i].score[j]);
```

```
        printf("%10.2f",st[i].ave);
    }


printf("\n");
fclose(fp);
fclose(fp1);

return 0;
}
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p168~169

```
2.          struct student
3.          {char num[10];
4.           char name[8];
5.           int score[3];
6.           float ave;
7.          }st[10],s;
```

看来是认准外部变量 Only You 了。

```
35.    for(t=0;st[t].ave>s.ave&&t<n;t++);
```

这里明显存在一个逻辑错误

```
01.25.              {printf("\n%8s%8s",st[i].num,st[i].name);
02.26.               for(j=0;j<3;j++)
03.27.                  printf("%8d",st[i].score[j]);
04.28.               printf("%10.2f",st[i].ave);
05.29.              }
```

与

```
01.37.            printf("\n%8s%8s",st[i].num,st[i].name);
02.38.            for(j=0;j<3;j++)
03.39.               printf("%8d",st[i].score[j]);
04.40.            printf("%10.2f",st[i].ave);
```

以及

```
01.47.            printf("\n %8s%8s",st[i].num,st[i].name);
02.48.            for(j=0;j<3;j++)
03.49.               printf("%8d",st[i].score[j]);
04.50.            printf("%10.2f",st[i].ave);
```

甚至

```
01.43.          printf("\n%8s%7s%7d%7d%7d%10.2f",s.num,s.name,s.score[0],
02.44.                s.score[1],s.score[2],s.ave);
```

是重复的

重复是万恶之源

最后，169 页的运行结果纯粹是伪造的，这很恶劣

P171~172

```
0.    #include<stdio.h>
1.    #include <stdlib.h>
2.    #include <string.h>
3.    struct employee
4.    {char num[6];
5.    char name[10];
6.    char sex[2];
7.    int age;
8.    char addr[20];
9.    int salary;
10.     char health[8];
11.     char class[10];
12.     }em[10];

13.     struct emp
14.     {char name[10];
15.     int salary;
16.     }em_case[10];

17.     int main( )
18.       {FILE *fp1, *fp2;
19.         int i,j;
20.       if ((fp1=fopen("employee","r"))==NULL)
21.         {printf("can not open the file.");
22.           exit(0);
23.         }
24.       printf("\n NO. name sex age addr salary health class\n");
25.       for(i=0;fread(&em[i],sizeof(struct employee),1,fp1)!=0;i++)
26.         {printf("\n%4s%8s%4s%6d%10s%6d%10s%8s",em[i].num,em[i].name,em[i].sex,
27.               em[i].age, em[i].addr, em[i].salary, em[i].health, em[i].class);
28.         strcpy(em_case[i].name, em[i].name);
29.         em_case[i].salary=em[i].salary;
30.         }
31.         printf("\n\n*********************************************");
32.         if((fp2=fopen("emp_salary","wb"))==NULL)
33.           {printf("can not open the file.");
34.             exit(0);
35.           }
```

```
36.             for(j=0;j<i;j++)
37.               {if(fwrite(&em_case[j],sizeof(struct emp),1,fp2)!=1)
38.                  printf("error!");
39.                printf("\n %12s%10d",em_case[j].name,em_case[j].salary);
40.               }
41.           printf("\n*****************************************");
42.           fclose(fp1);
43.           fclose(fp2);
44.           return 0;
45.         }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p171~172

```
3.            struct employee
4.            {char num[6];
5.              char name[10];
6.              char sex[2];
7.              int age;
8.              char addr[20];
9.              int salary;
10.               char health[8];
11.               char class[10];
12.             }em[10];

13.             struct emp
14.             {char name[10];
15.               int salary;
16.             }em_case[10];
```

定义了两个外部数组不但非常可笑，而且也是一种作茧自缚的行为。因为在文件中数据超过 10 个的时候程序失效。

此外既然这里可以把 name 和 salary 聚合成一个结构体类型的数据，那么当初写文件的时候就应该把 struct employee 类型定义为

```
struct emp
{
        char name[10];
        int salary;
};

struct employee
{
        struct emp name_salary;
        char num[6];
        char sex[2];
```

```
        int age;
        char addr[20];
        char health[8];
        char class[10];
}
```
这样代码要简单很多。

```
20.             if ((fp1=fopen("employee","r"))==NULL)
```
这里的错误地把"rb"写成了"r"，这会导致读取数据提前结束的异常。

```
32.          if((fp2=fopen("emp_salary","wb"))==NULL)
```
这次打开模式意外地写对了，应该表扬一下。然而
```
34.                  exit(0);
```
这里又错了。

```
36.              for(j=0;j<i;j++)
37.                 {if(fwrite(&em_case[j],sizeof(struct emp),1,fp2)!=1)
38.                     printf("error!");
39.                  printf("\n %12s%10d",em_case[j].name,em_case[j].salary);
40.                 }
```
这里的错误是出现异常不做任何处理，还当作平安无事地继续执行程序，勇敢向前走，尽管灾难在招手，全然不觉已经大事不好了。

P173

```
#include<stdio.h>
#include <stdlib.h>
struct employee
{char num[6];
char name[10];
char sex[2];
int age;
char addr[20];
int salary;
char health[8];
char class[10];
}em[10];

int main( )
   {
    FILE *fp;
```

```
    int i;
    printf("input NO.,name,ex,age,addr,salary,health,class\n");
    for(i=0;i<4;i++)
    scanf("%s%s%s%d%s%d%s%s",em[i].num,em[i].name,em[i].sex,
              &em[i].age,em[i].addr,&em[i].salary, em[i].health,em[i].class);

          //将数据写入文件
    if((fp=fopen("employee","w"))==NULL)
       {printf("can not open the file.");
        exit(0);
       }

    for(i=0;i<4;i++)
      if(fwrite(&em[i],sizeof(struct employee),1,fp)!=1)
          printf("error!");
    fclose(fp);
    return 0;
  }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p173

这段代码用于建立文件

```
2.          struct employee
3.          {char num[6];
4.           char name[10];
5.           char sex[2];
6.           int age;
7.           char addr[20];
8.           int salary;
9.           char health[8];
10.            char class[10];
11.          }em[10];
```

描述职工数据的数据结构不合理。例如职工号用 char [6]，性别用 char [2]以及工资用 int 类型。

最荒唐的是外部数组 em。如果像代码中那样从键盘输入数据建立文件的话根本不需要外部变量，更不需要数组。更何况这个具有 10 个元素的数组完全把程序的功能给无理地限定非常狭仄。

```
17.           for(i=0;i<4;i++)
18.           scanf("%s%s%s%d%s%d%s%s",em[i].num,em[i].name,em[i].sex,
19.                   &em[i].age,em[i].addr,&em[i].salary, em[i].health,em[i].class);
```

这个“i<4”，进一步限制了程序功能

```
21.           if((fp=fopen("employee","w"))==NULL)
```

```
22.                 {printf("can not open the file.");
23.                  exit(0);
24.                 }
```
文件打开模式错误；exit(0)似是而非。

```
25.             for(i=0;i<4;i++)
26.                 if(fwrite(&em[i],sizeof(struct employee),1,fp)!=1)
27.                     printf("error!");
```
这里就更奇怪了。已经发生了错误，只输出了一个"error!"，然后就像没事人一样若无其事地继续执行，就像开车发现发动机漏油，只是跟乘客打了声招呼但依旧全速继续开一样。

P174~175

```
0.          #include <stdio.h>
1.          #include <stdlib.h>
2.          #include <string.h>
3.          struct employee
4.          {char name[10];
5.          int salary;
6.          }emp[20];

7.          int main( )
8.             {FILE *fp;
9.              int i,j,n,flag;
10.             char name[10];
11.             if((fp=fopen("emp_salary","rb"))==NULL)
12.                {printf("can not open file.");
13.                 exit(0);
14.                }
15.             printf("\noriginal data:");
16.             for(i=0;fread(&emp[i],sizeof(struct
17.                 employee),1,fp)!=0;i++)
18.                 printf("\n %8s %7d",emp[i].name,emp[i].salary);
19.             fclose(fp);
20.             n=i;
21.             printf("\n input name deleted:\n");
22.             scanf("%s",name);
23.             for(flag=1,i=0;flag&&i<n;i++)
24.              {if(strcmp(name,emp[i].name)==0)
25.                  {for(j=i;j<n-1;j++)
26.                   {strcpy(emp[j].name,emp[j+1].name);
```

```
27.                         emp[j].salary=emp[j+1].salary;
28.                       }
29.                       flag=0;
30.                     }
31.                   }
32.               if(!flag)
33.                   n=n-1;
34.               else
35.                   printf("\nnot found!");
36.               printf("\n Now,the content of file:\n");
37.               if((fp=fopen("emp_dalary","wb"))==NULL)
38.                   {printf("can not open file.");
39.                    exit(0);
40.                   }
41.               for(i=0;i<n;i++)
42.                   fwrite(&emp[i],sizeof(struct employee),1,fp);
43.               fclose(fp);
44.               fp=fopen("emp_salary","r");
45.               for(i=0;fread(&emp[i],sizeof(struct employee),1,fp)!=0;i++)
46.                   printf("\n%8s%7d",emp[i].name,emp[i].salary);
47.               printf("\n");
48.               fclose(fp);
49.               return 0;
50.           }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p174~175

```
3.          struct employee
4.          {char name[10];
5.          int salary;
6.          }emp[20];
```

这个让人喷饭。因为在前面的题目中

```
struct emp
{char name[10];
 int salary;
}em_case[10];
```

已经引刀自宫地限定了文件中最多有 10 个数据，现在居然企图人为地把数据数目加大为 20，这是自欺欺人。

```
9.              int i,j,n,flag;
```

看见 flag 就仿佛能嗅出代码中的馊味。不信？等着瞧。

```
11.              if((fp=fopen("emp_salary","rb"))==NULL)
12.                  {printf("can not open file.");
```

```
13.                     exit(0);
14.                   }
```
这里的 exit(0);是错误的，打开方式是错误的。"rb"只表示想要读文件，但程序显然不仅仅需要读这个文件，还需要写

```
16.                 for(i=0;fread(&emp[i],sizeof(struct
17.                     employee),1,fp)!=0;i++)
18.                     printf("\n %8s %7d",emp[i].name,emp[i].salary);
19.                 fclose(fp);
20.                 n=i;
```
题目并没有要求做输出数据这件事，这叫多管闲事，也叫多做之过。
这段代码的另一个功能是统计文件中有多少个数据，它设计在 fread()返回值为 0 时停止循环，然后指望根据 i 的值来了解数据文件中有多少数据。但是这个得意算盘的失算之处在于，fread()并非仅仅在读到文件结尾时返回值为 0，当读入过程中发生错误 fread()的返回值也同样为 0。

```
23.                 for(flag=1,i=0;flag&&i<n;i++)
24.                   {if(strcmp(name,emp[ i ].name)==0)
25.                       {for(j=i;j<n-1;j++)
26.                         {strcpy(emp[j].name,emp[j+1].name);
27.                           emp[j].salary=emp[j+1].salary;
28.                         }
29.                         flag=0;
30.                       }
31.                   }
```
这段代码不伦不类非常蹩脚，实际上它表达的无非是

```
01.     for( i = 0 ; i < n ; i++ )

02.         if(strcmp(name,emp[i].name)==0)

03.         {

04.             for( j = i ; j < n - 1 ; j++ )

05.             {

06.                 strcpy(emp[j].name,emp[j+1].name);

07.                 emp[j].salary=emp[j+1].salary;

08.             }

09.             break;

10.         }
```

复制代码而已。原来代码中的 flag 像正在发炎的阑尾，除了带出毛病，本身是没有什么用的。这段代码更简单的写法是

```
01.     for( i = 0 ; i < n ; i++ )

02.         if(strcmp(name,emp[i].name)==0)

03.             break;

04.

05.     for( j = i ; j < n - 1 ; j++ )

06.     {

07.         strcpy(emp[j].name,emp[j+1].name);

08.         emp[j].salary=emp[j+1].salary;

09.     }
```

复制代码原来那种复杂的循环嵌套显然是源自一种变态的想法，是把简单问题复杂化。

```
35.                     printf("\nnot found!");
```

这行代码的问题是，既然明明知道没找到，那么数据文件不用做任何处理，直接退出就可以了。可是它没有退出，表现出一副非常留恋的样子，继续执行后面在这种情况下其实完全没有意义的操作。

```
44.                 fp=fopen("emp_salary","r");
45.                 for(i=0;fread(&emp[i],sizeof(struct employee),1,fp)!=0;i++)
46.                     printf("\n%8s%7d",emp[i].name,emp[i].salary);
47.                 printf("\n");
48.                 fclose(fp);
```

很不幸，打开文件的模式"r"又是错的，应该是"rb"。而且一相情愿地假设读入不会发生错误。此外问题根本没有要求重新打开文件输出数据。

P175

从键盘输入若干行字符（每行长度不等），输入后把它们存储到一磁盘文件中。再从该文件中读入这些数据，将其中小写字母转换成大写字母后在显示屏上输出。——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p175

题目不明确的地方在于没有说明输入如何结束，而所谓“每行长度不等”又是毫无意义的废话。

P175~176

```
0.      #include <stdio.h>
1.      int main( )
2.          {int i,flag;
3.           char str[80],c;
4.           FILE *fp;
5.                fp=fopen("text","w");
6.                flag=1;
7.                while(flag==1)
8.                  {printf("\n Input string:\n");
9.                   gets(str);
10.                   fprintf(fp,"%s",str);
11.                   printf("\nContinue?");
12.                   c=getchar();
13.                   if((c=='N')||(c=='n'))
14.                   flag=0;
15.                   getchar();
16.                  }
17.               fclose(fp);
18.               fp=fopen("text","r");
19.               while(fscanf(fp,"%s",str)!=EOF)
20.                  {for(i=0;str[i]!='\0';i++)
21.                   if((str[i]>='a')&& (str[i]<='z'))
22.                   str[i]-=32;
23.                   printf("%s\n",str);
24.                  }
25.               fclose(fp);
26.               return 0;
27.           }
```

——谭浩强 ，《C 程序设计（第四版）学习辅导》，清华大学出版社，2010 年 7 月，p175~176

由于问题并没有限制每行输入字符数不超过 80，所以

```
3.           char str[80],c;
```

这种数据结构毫无依据。如果用户输入的某行字符数超过 80，程序可能面临灾难性的崩溃。

```
5.      fp=fopen("text","w");
```

这里有两个错误，第一没有检查文件是否打开，即没有考虑 fp 的值为 NULL 的情况；此外没有对 fopen() 的类型进行声明，亦即代码缺少

```
#include <stdlib.h>
```

```
6.                flag=1;
7.                while(flag==1)
```

```
8.                          {printf("\n Input string:\n");
9.                           gets(str);
10.                            fprintf(fp,"%s",str);
11.                            printf("\nContinue?");
12.                            c=getchar();
13.                            if((c=='N')||(c=='n'))
14.                            flag=0;
15.                            getchar();
16.                           }
```

这通常是 BASIC、FORTRAN 等语言的一种笨拙写法，由于语言的表达能力不强，在那些语言中只好这样写。但在 C 语言中这样写就笨得有些可笑了。那个 flag 怎么看怎么别扭。

这里使用的 gets(str)函数调用也很成问题，第一，由于前面限定了

```
3.                  char str[80],c;
```

所以可能发生越界的可能。第二，gets()函数并不将读入的回车换行写入数组，造成信息丢失，这样将来也无法将回车换行符写入文件，以后再从文件读出的内容根本分不清究竟是哪行的，成了一锅粥。

```
18.                         fp=fopen("text","r");
```

这里的错误同样是没有检查 fopen()返回值为 NULL 的情况。

```
19.                         while(fscanf(fp,"%s",str)!=EOF)
20.                            {for(i=0;str[i]!='\0';i++)
21.                             if((str[i]>='a')&& (str[i]<='z'))
22.                             str[i]-=32;
23.                             printf("%s\n",str);
24.                            }
```

这段代码的主要问题是，没有检查 fscanf()返回 EOF 是到达文件结尾还是读入过程出错。此外“str[i]-=32;”这种写法非常差劲，32 是一个莫名其妙的常数。

最为恶劣的是
176 页的运行结果完全是伪造的
这和错误不同
是欺骗

第 11 章 预处理命令

P177

第 11 章 预处理命令
在预处理阶段，预处理器把程序中的注释全部删除；
评：不是“删除”
“程序”这个术语也不妥当
应该是“源程序”

P177

将程序中的符号常量用指定的字符串代替
评：“字符串”这个术语是错误的

P177

最后再由编译程序对预处理后的源程序进行实际的编译处理，得到可供执行的目标代码
评：“预处理后的源程序”，应该是翻译单元
“实际的编译处理”,看来还有虚幻的编译处理了
“得到可供执行的目标代码”，得到的并不是可供执行的目标代码

P177

C 语言与其他高级语言的一个重要区别是可以使用预处理指令和具有预处理的功能
评：胡扯

P178

#define 标识符 字符串
评：“字符串”术语错用

P178

这就是已经介绍过的定义符号常量
评：只能说符号常量是用这种方式定义的
这种宏并不是定义符号常量专用的

P178

#define PI 3.1415926
……
将程序中……所有的“PI”都用“3.1415926”
评：不是“所有”

P178

这种方法使用户能以一个简单的名字代替一个长的字符串，因此把这个标识符(名字)称为“宏名”
评：“因此”二字万分可笑

谭从来搞不清什么叫因什么叫果
因此，“因此”二字在他那里只是一种语气词而已
阅读时务请特别注意小心

P179

C 语言把所有的实数都作为双精度数处理
评：扯淡
事实是
谭使用 float 在新的编译器中遇到了“警告”
但并不知道如何处理
只好把所有的数据都改成了 double 类型

P179

由于变量为 double 型，故在输出时用%lf 格式符（在 f 之前加小写字母 l），否则会输出不正确的数字。读者可以试一下。
评：读者只要试一下就会知道这绝对是在胡扯

P179

使用宏名代替一个字符串，可以减少程序中重复书写某些字符串的工作量
评：“字符串”术语错用
此外这不是宏最重要的用意

P179

#define array_size 1000
评：忘记在同页刚刚说过“宏名习惯用大写”

P179

宏定义只是用宏名代替一个字符串
评：根本没搞清楚什么叫“宏定义”

P180

#define 指令出现在程序中的函数的外面
评：没有这回事

P181

对程序中用双撇号括起来的字符串内的字符，即使与宏名相同，也不进行置换
评：这没错
　　但和老谭前面自己说过的“所有的都……代替”自相矛盾

P181

#define 宏名(参数表) 字符串
评：字符串，是程序处理的对象，其含义是在尾部包含\0 字符的字符序列。用在这里明显是关公战秦琼

P181

带参数的宏定义不是进行简单的字符串替换，还要进行参数替换
评：这和 179 页所说的“宏定义只是用宏名代替一个字符串”明显是自相矛盾的

P181

在程序中如果有带实参的宏（如 S(3,2)），则按#define 指令行中指定的字符串从左到右进行置换
评：并不存在什么“从左到右进行置换”
　　这个说法也没有任何实际意义

P181

#define S(r) PI*r*r
评：这是一个不合格的宏定义
　　在例题中写出这种东西是一个误导

P182

实参字符串
评：术语错误

P182

将 S(a)中的实参字符 a 代替宏定义中的字符串"PI*r*r"中的形参 r，
评：前面说“实参字符串”
　　这里说“实参字符”
　　自相矛盾，而且都属于错误的术语

“字符串"PI*r*r"中的形参 r”，错的更没边了，这里哪来的形参呢

P182

字符串中的形式参数
评：同前

P182

#define S　(r) PI*r*r　//在 S 后有一空格
系统会认为 S 是符号常量（不带参数的宏名）
评：这不是“符号常量”
　　“符号常量”和“不带参数的宏名”也并非等价的说法

P182

#define S　(r) PI*r*r　//在 S 后有一空格
……
area=S(a);
则被置换为
aArea=(r) PI*r*r(a);
评：aArea:明显的错误

P183

实参字符“a+b”
评：这是“字符”吗？

P183

对函数中的实参和形参都要定义类型
评：对“实参”“定义类型”？
　　怎么定义？

P183

定义宏时，字符串可以是任何类型的数据
评：简直不敢相信我的眼睛
哪怕喝高了也不至于说出这种胡话吧

P183

#define CHAR1 CHINA　　（字符）
#define A 3.6　　（数值）
……在程序中凡遇“CHAR1”均以字符“CHINA”代之；凡遇“A”均以字符"3.6"代之
评：字符“CHINA”
　　字符"3.6"

　　一个字：雷人

P183

调用函数只可得到一个返回值，而用宏可以设法得到几个结果
评：前半句是对的，
　　后半句纯粹是胡扯

P183

#define CIRCLE(R,L,S,V) L=2*PI*R;S=PI*R*R;V=4.0/3.0*PI*R*R*R
评：如果用意是展示一下什么是垃圾
　　我同意

P183

double r,……
scanf("%f",&r);
评：不妨留给初学者作为改错题吧

P184

程序中只给出一个实参 r 的值，就可以从宏 CIRCLE 的置换中得到 3 个值(l,s,v)
评：牛！
　　金庸应该来学习学习怎样写武侠

P184

使用宏次数多时，宏展开后源程序变长，因为每展开一次都使程序增长，而函数调用不会使源程序变长
评：没有根据的说法
　　而且这种讨论没有意义

P184

#define MAX(x,y) (x)>(y)?(x) : (y)
评：这是一个不合格的宏定义

P184~185

可以事先将程序中的“输出格式”定义好，以减少在输出语句中每次都要写出具体的输出格式的麻烦。
评：这是书房里的异想天开

P185

#define PR printf
#define NL "\n"
#define D "%d"
#define D1 D NL
……
PR(D1,a);
……
评：如果想让老板和同事的讥笑和痛斥
    你不妨这样写好了

    建议学习者完全无视 185 页，直接跳过为好

P185

把它们单独编成一个文件，它相当于一个“格式库”
评：Too simple,sometimes naive
    对开发过程完全无知
    对“库”的概念也一无所知

P186

图 11.2（a）为文件 file1.c，该文件中有一个#include <file2.c>指令
评：笑喷了
    参见
    《谭浩强的书我就不说什么了，居然教学生 include 一个.c 文件》
    http://bbs.chinaunix.net/thread-1603177-1-1.html

    其中有一种观点认为，在某些特定情况下需要 include .c 文件

starwing83 网友对此发表了精彩的见解：（3140 楼）

我看了一下那个帖子，发现根本就没有说到重点上。include .c，如果.c 声明的是非 static 的函数，那么根本就没法解决重名问题。比如说两个库都有 OpenFile，而且不是 static 的，那么无论你是不是 include .c 都会链接错误，因为.o 里面的名字相同，这个没法骗人的。

所以这种情况下，要么在.c 里面写 static 函数并且 include .c（后缀名不重要），这样即使.c 被意外编译也无所谓，要么就是脱了裤子放屁，照样得死。唯一的解决方案只有加前缀。

如果.c 里面的是 static 函数，用这个后缀名也是不合适的，.c.inc 就好多了。

最后，如果实在懒得加前缀，可以考虑 zlib 的做法，即写个头文件，写一堆的
#define OpenFile A_OpenFile

然后用在.c 文件里面（绝对不能被包含到.h 文件中），然后.c 里面就可以用 OpenFile 了，当然这样还是不需要 include .c，直接分离编译即可。

否则，所有内部函数都要加 static，这个工作量看不出来和加前缀有什么区别，有功夫给所有的内部函数加 static，不如用刚才说的技术，写个那种头文件。

当然如果还是嫌麻烦，先 ctags 搞出一个列表，然后 vim 处理一下做个脚本，用 sed 处理一遍，也很方便。

总的来说，include .c 完全是没有必要的。而且是危险的、不能解决问题的。

再说说：以上这堆估计村长是知道的，但是他：
- 就是不想让函数被除了本模块的人以外的人知道（那么加前缀就没用了）
- 希望编辑器能高亮（那就必须.c 结尾）
- static 只有在一个文件内才有用（那就只有 include 了）

这种情况下，.c.inc 至少对 Vim 和 Emacs 来说也是有用的（点分隔的扩展名只要有一个有效就会默认高亮），另外，还有一个非常困难的解决方案：
-- 究竟是为什么，一个模块会有几十万行代码，而且连接紧密？能不能分解为几个模块？

村长的答案是不行，我觉得这个很匪夷所思，就算有 static，如果要将一个模块分解成几个模块也不是很难的——最简单的思路，如果模块有几十万行，我很难想像其对外接口会只有一个公开函数，那么如果有多个公开函数，那么很容易就可以让一个公开函数一个文件，然后这个只有公开函数需要的功能写成 static 放到这个文件中，对于模块内部需要使用的公共服务，这个是绝对可以抽象成一个独立模块的（因为都是服务，就肯定能做事，因为共有了，就肯定有协议，有协议又做实事的，是绝对可以作为一个独立模块而存在的），最终会形成一个模块的树，然后依次都可以简单地.h .c 了，当然这就是重构。

这里仍然有个问题没法解决：假设 A，B 两个模块都变成了模块树，那么底层的小模块就相当于透明

了，一旦小模块的 API 相同，那么仍然逃不过链接错误的命运，而且实际上这样仍然是没有隐藏掉底层小模块的接口的。

所以，实际的解决方法是——让函数的实现和命名有必然关系，让一个功能只能会有一个名字，让一个名字一定是某个特定的功能而不会重复，这仍然是重构，在这个基础上，是根本就不可能出现名字相同而功能不同的情况的——这种情况是设计缺陷。很多时候，名字重复而功能不同是概念定义模糊的结果，Open 还是 Create？NetFile 还是 LocalFile？如果真的严格区分了项目所有的概念，就不大可能出现重名问题了，就算都是 Matrix，你也可以命名成 m3x3 或者 m4x4。

这里仍然有个问题，假设两个模块都有 OpenFile，但是具体的配置不一样（比如一个立即打开，一个延缓打开），这样这个 OpenFile 是可以抽出作为独立模块的，然后包括所有的配置（这是模块的责任）然后分别给重名的位置使用——这仍然是重构。

那么，如果在设计之初函数名和功能没有本质上的联系（都叫 OpenFile，一个是打开本地文件，一个是打开网络文件），而且又不愿意重构，那就只能 include .c 了，然而，即使是这种极端情况，include .c.inc 也好很多，有 include .c 的所有优点，又没有缺陷。

总的来说，include .c 是一个信号——“你的模块划分有问题，你的模块 API 命名有问题，你项目里的概念不够清晰”，这是需要解决的，include .c 绝对是一个红灯，当然闯红灯是可以的——只要你爸是李刚即可，但是大多数情况，我们还是要为自己的生命安全着想。

P186

“文件包含”指令是很有用的，它可以节省程序设计人员的重复劳动
评：前半句是废话
后半句表明作者根本不清楚其实际使用方法
另：“节省”“重复劳动”是病句

P186

例 11.6 将例 11.5 的格式宏做成头文件，把它包含在用户程序中
评：错上加错，而且极其无聊
建议初学者直接无视

P187

……“头文件”，常以“.h”为后缀……当然不用“.h”为后缀而用“.c”为后缀或者没有后缀也是可以的
评：这是在教人如何在项目中捣乱

P188

如果文件 1 需要包含文件 2，而在文件 2 中又要用到文件 3 的内容，则可在文件 1 中用两个#include 指令

分别包含文件 2 和文件 3，而且文件 3 应出现在文件 2 之前，即在 file1.c 中定义：
#include "file3.h"
#include "file2.h"
评：这是完全不懂代码管理的外行话
“在 file1.c 中定义”：居然能冒充“定义”两个字

P188

头文件除了可以包含函数原型和宏定义外，也可以包括结构体类型定义（见第 10 章）和全局变量定义
评：“全局变量”本身是错误的概念
在.h 中定义变量是极其错误的做法

P189

#ifdef COMPUTER_A
#define INTEGER 16
#else
#define INTEGER_SIZE 32
#endif
评：#define INTEGER
明显是
#define INTEGER_SIZE
之误

P192

此时运行结果为
C language
评：这又是一个伪造的结果

P192

对这个问题完全可以不用条件编译处理而用 if 语句处理，但那样做，目标程序长（因为所有语句都编译），运行时间长……
评：这属于不懂装懂
if 语句和条件编译所要解决的问题是截然不同的
用条件编译根本不是为了解决“目标程序长，运行时间长”这样的问题

P192

只是为了说明怎样使用条件编译，有人会觉得其优越性不太明显

评：自己都不懂得为什么使用条件编译
怎么可能说明怎样使用条件编译呢
以其昏昏使人昭昭
是不可能的

P192

预处理功能是 C 语言特有的
评：是井底之蛙
还是没话找话？
而且自相矛盾
因为在 177 页说过“它不是 C 语言本身的组成部分”
怎么又成了“C 语言特有的”的呢

P192

善于利用预处理命令，对提高程序的质量会有好处的
评：程序质量不高不可能是因为“善于利用预处理命令”
但像谭这样滥用预处理命令会败坏程序质量则是必然的

第 12 章 位运算

P193

评：开篇第一句就很有气势
位运算是 C 语言的重要特色，是其他计算机高级语言所没有的
评：什么话都敢说
C++er 对此不知作何感想

P193

所谓位运算是指以二进制位为对象的运算
评：不好意思谭先生
C 语言里没有以位对运算对象的运算

P193

在系统软件中，常要处理二进制位的问题

评：有这事儿吗？

P193

例如，将一个存储单元的各二进制位左移或右移一位，两个数按位相加等
评：这个“例如”难道是在佐证“在系统软件中，常要处理二进制位的问题”吗？
依据不足啊

两个数“按位相加”是什么意思？恐怕又是谭先生的“发明”吧

P193

C 语言提供位运算的功能，与其他高级语言相比，它显然具有很大的优越性。
评：有挑起语言战争之嫌
我认为 Ritchie 是绝对不敢这样讲的

P193

指针运算和位运算往往是编写系统软件所需要的
评：不写 OS 就不必学了吧

P193

在计算机检测和控制领域也要用到位运算的知识
评：嗯、长见识了

P193

参加位运算的对象只能是整型或字符型的数据
评：从来搞不清谭先生的“整型”是什么个概念

P193

如果参加“&”运算的是负数（如-7&-5），则以补码形式表示为二进制
评：假定计算机采用补码是不正确的

P194

按位与有一些特殊的用途
(1)清零。如果想将一个单元清零，即使其全部二进制为 0，只要找一个二进制数，其中各个位符合以下条件：原来的数中为 1 的位，新数中相应位为 0.然后使二者进行&运算，即可达到清零目的。
例如，原有数为 00101011，另找一个数，设它为 10010100，它符合以上条件，即在原数为 1 的位置上，它的位值均为 0，将两个数进行&运算：

```
    00101011
(&) 10010100
------------------------
    00000000
```

其道理是显然的。当然也可以不用 10010100 这个数，而用其他数（如 01000100）也可以，只要符合上述条件即可。
评：这样实际上是无法达到“将一个单元清零”的目的的（且不说一个单元这个概念在这里是模糊不清的）
因为这只是得到了一个某种类型的 0 值而已
必须还要经过赋值才能达到“将一个单元清零”的目的
既然如此，还不如直接赋值
即使是希望通过&得到这个 0 值
也完全没必要去寻找什么“原来的数中为 1 的位，新数中相应位为 0”这样的二进制数
用 0 就可以了

P194

（2）取一个数中某些指定为……
评：基本上把&运算解释成了一种算术运算而不是一种 C 语言的运算，因为是完全脱离数据类型来讲的，这个问题在这一章非常明显

P194

将八进制数 60 与八进制数 17 进行按位与运算
评：问题的提法就不正确
根本不存在八进制数按位与运算

C 语言中没有任何一种运算的运算对象是八进制数

P195

若参见位运算的两个二进制位异号，则得到 1（真），若同号，则结果为 0（假）
评：位运算哪来的什么真假？

P195

012^00=012
评：00

P197

~运算符的优先级别比算术运算符、关系运算符、逻辑运算符的其他运算符都高
评：谭一定是忘记他把 ！也说成是逻辑运算符了

P197

“<<”用来将一个数的各二进制位全部左移若干位
评：将“一个数”是误导

P197

假设以一个字节（8 位）存一个整数，若 a 为无符号整型变量，则 a=64 时，左移一位时溢出的是 0，而左移 2 位时，溢出的最高位中包含 1.
由表 12.2 可以看出，若 a 的值为 64，在左移 1 位后相当于乘 2，左移 2 位后，值等于 0。
评：不考虑措辞的不严谨
仅就一个字节（8 位）的无符号整数类型变量 a 而言(只能是 unsigned char 或 char 类型)
上面所描述的情形在任何编译器中都不可能发生

P198

例如：a 是 short 型变量，用两个字节存放数，若 a 值为八进制数 113755，即最高位为 1，对它进行右移两位的运算：a>>1。请看结果
a: 1001011111101101
a>>1: 01001011111101101 （算术右移时）
a>>1: 11001011111101101 （逻辑右移时）
评：右移两位的运算：a>>1

a>>1 的结果与具体实现有关
未必是列出的那两种样子

P198

如果两个数据长度不同（例如 short 和 int 型）进行位运算时（如 a&b，而 a 为 short，b 为 int 型），系统会将二者按右端对齐。如果 a 为正数，则左侧 16 位补满 0；若 a 为负数，左端应补满 1;如果 a 为无符号型，则左侧填满 0.

评：这个绝对是大错特错，胡说八道

所谓“右端对齐”绝对是捏造出来的

P198

例 12.1 从一个整数 a 中把右端开始的 4~7 位取出来

评：这个问题本身就成问题

“整数 a”，“右端”，都是非计算机领域的很不规范的说法

P199

(a>>4)&~(~0<<4)

评：所谓“取出”，也没有要求>>4

程序运行结果甚至输出了这 4 位的值，更是莫名其妙的做法

P199

例 12.2 循环移位。要求将 a 进行右循环移位，见图 12.4。图 12.4 表示将 a 右循环移位 n 位，即将 a 中原来左面（16-n）位右移 n 位，原来右端 n 位移到最左面 n 位。今假设用两个字节存放一个短整数(short int 型)。

评：题目出的就稀里糊涂

a 是什么根本没有交代

16-n 中的 16 是哪里来的？

“今假设用两个字节存放一个短整数(short int 型)”更是莫名其妙

P200

程序如下：

```
#include <stdlib.h>
int main()
{unsigned a,b,c;
int n;
printf("please enter a & n"\n);
scanf("a=%o,n=%d",&a,&n);
b=a<<(16-n);
c=a>>n;
```

```
c=c|b;
printf("a:\nc",a,c);
return 0;
}
```

评：如果把这叫做程序，那垃圾应该叫什么？

代码写得令人作呕不说

（unsigned a,b,c;

scanf("a=%o,n=%d",&a,&n);

b=a<<(16-n);

）

甚至错得连编译都无法通过

（printf("a:\nc",a,c);

printf("please enter a & n"\n);

）

可书中竟然给出了运行结果

而且这里面压根就不干题目中所提到的“短整数(short int 型)”什么事

P201

```
data=data&~(15<<4)|(n&15)<<4
```

评：目的是将 n(=12)写到 data 的第 4~7 位

其中的 &15 毫无必要

P201

```
struct Packed_data
    {unsigned a:2;
      unsigned b:2;
      unsigned c:2;
      unsigned d:2;
      short i;
    }data;
```

……其中的 a，b，c，d 段分别占 2 位、6 位、4 位、4 位，i 为 short 型，以上共占 4 个字节。

评：“占 4 个字节”的说法是武断的

P202

请注意位段允许的最大值的范围。如果写成

```
data.a=8;
```

就错了。因为 data.a 只占两位，最大值为 3。在此情况下，系统会自动取赋予它的数的低位。

评：是错了

但“在此情况下，系统会自动取赋予它的数的低位。”也错了
如果自动取低位，data.a=8;也就不成为错误了

P202

位段成员的类型可以指定为 unsigned int 或 int 型。
评：错
对于 C90 来说还可以指定为 signed int 类型
对于 C99 来说还可以指定为_Bool 等类型

P202

“宽度”……必须小于或等于指令类型的位长
评：“指令类型的位长”？
莫名其妙的说法
不知所云

P202

对位段组（……，至少占一个存储单元（即一个机器字，4 个字节），即使实际长度只占一个字节，但也得分配 4 个字节
评：一连串错误：
至少占一个存储单元
（即一个机器字，4 个字节）
即使实际长度只占一个字节，但也得分配 4 个字节

P203

一个位段必须存储在同一存储单元中，不能跨越两个单元
评：错

P203

位段的长度不能大于存储单元的长度
评：不存在这种问题

P203

位段中的数可以用整型格式输出，例如
printf("%d,%d,%d",data.a,,data.b,data.c);

当然，也可以用%u,%o,%x
评：这是有条件的
不是所有的“位段中的数”（应该是位段成员）都可以随意用这几种转换格式

P203

位段可以在数值表达式中引用，它会被系统自动地转换成整型数
评：到底转变成什么类型？
这里的说法是模棱两可的
总结：12.3 位段部分几乎到处是错，尤其是后面说明部分，一共 7 条，全都是错的

P203

位段可以在数值表达式中引用，它会被系统自动地转换成整型数
评：到底转变成什么类型？

P203

位段可以在数值表达式中引用，它会被系统自动地转换成整型数
评：到底转变成什么类型？

P203

位段可以在数值表达式中引用，它会被系统自动地转换成整型数
评：到底转变成什么类型？

P203

位段可以在数值表达式中引用，它会被系统自动地转换成整型数
评：到底转变成什么类型？

P203

位段可以在数值表达式中引用，它会被系统自动地转换成整型数
评：到底转变成什么类型？

316-3159

《C 程序设计（第三版）》部分

目录

评：结构性缺陷。现在我终于知道他为什么写“#include <file2.c>”了。

(前言)pIX

算法是程序的核心、是灵魂，语法是外壳、是工具。不应把学习重点放在语法规则上，……一定要把重点放在解题思路上，通过大量的例题学习怎样设计一个算法，构造一个程序。
评：典型的企图不学走就学跑式的一相情愿。事实上，书中根本就没什么象样的算法。这是一个误导，也是这本书在指导思想上的根本错误。数据类型都整不明白，侈谈什么算法

PS：每当看到有初学者说“算法才是王道”，我都忍不住想笑

(前言)pIX

学习程序设计课程时，应该把精力放在最基本、最常用的内容上，学好基本功。
评：我认为这话是对的。但却是与前面过分强调渲染算法是自相矛盾的，而且这本书对“最基本、最常用的内容”讲的稀里糊涂，甚至是错误连篇的。

(前言)pX

衡量这门课的好坏，不是看你“知不知道”，而是“会不会干”

评：我就奇怪了，不知道怎么干？怎么可能“会干”不知道只可能导致胡来瞎干

(前言) p VII

……十多年来，该书累计发行了 700 多万册，平均每年印刷 50 万册，居全国同类书的首位。全国大多数高校把本书作为正式教材。许多高校的研究生入学考试都指定本书为必读教材，国内许多介绍 C 语言的书籍以本书为蓝本，……本书成为广大初学者学习 C 语言程序设计的主流用书

评：

“我发现很多非专业的也都用谭的 c 教材。是不是楼主对”主流教材“要求过高了。”

“您觉得是否“过高了”呢？（我发现很多非专业的也都用谭的 c 教材。是不是楼主对”主流教材“要求过高了。）”

“现在发行量已经突破 1000 万册了，

也就是说，祸害了 1000 万人”

“很多错误非常低级，以至于不值得批驳

但是，初学者是无法察觉这些错误的

而且每一个错误都要乘以 1000 万

ps:总能想到三鹿”

p VIII

用 C++编译系统时，对程序要求更加规范。例如，在定义和声明函数时，必须指定函数类型；程序中如果用到系统提供的库函数（包括 printf 和 scanf 函数），都必须在程序文件的开头用#include 命令将有关头文件包含进来。因此，本书的程序基本上采用下面的形式：

```
#include <stdio.h>
void main()
{
.
.
.
}
```

评：原以为老谭终于懂得了为什么#include，遗憾的是不是这样，他只是被迫这样的。

这种代码极不规范(void main())，违背 C 标准（C89），最多算一种 C 方言而已。

p 2

第一章 C 语言概述 1.2 C 语言的特点

对变量的类型使用比较灵活，例如，整型量与字符型数据以及逻辑型数据可以通用。

评：误导。C 语言根本就没有逻辑“型”数据。况且数据类型和变量是正交的概念，“对变量的类型使用比较灵活”，难道对常量的类型使用就死板？“通用”这个说法更成问题

p 2

1983 年，美国国家标准化协会（ANSI）根据 C 语言问世以来各种版本对 C 语言的发展和扩充，公布了第一个 C 语言标准草案（'83 ANSI C)

评：我从来就没听说过这个所谓的“'83 ANSI C”。不知道作者这样说的根据何在？据我所知，1983 年 X3J11 刚刚成立。刚成立就能抛出一个标准草案？

信口开河的例子在这本书里不是一处两处。

CU 的 wiki 上也说：“1983 年，美国国家标准委员会（ANSI）对 C 语言进行了标准化，于 1983 年颁布了第一个 C 语言标准草案（83 ANSI C)”- http://wiki.chinaunix.net/index.php/C

在这里我郑重地请教一下该词条的编写者，“83 ANSI C”是怎么回事情？

p 2

1990 年，国际标准化组织 ISO（International Standard Orgnization）接受 C89……

评：ISO 啥时候改名了？

我记得这个组织的正式名称是 International Organization for Standardization

难怪书这么“水”

ISO 都被山寨了

p 3

C 语言的这种双重性，使它既是成功的系统描述语言，又是通用的程序设计语言。。

评：“什么叫“系统描述”，什么叫“系统程序设计”？人们都都说 C 是一种“系统程序设计语言”，而“系统描述语言”则完全是另一种东西。想必谭先生对它们之间的差异和关系都不清楚。在这种情况下就来下断言，一上马就露了怯。”——引自《谭浩强《C 语言程序设计（第二版）》的前 50 页中的错误分析》

p 3

```
#include<stdio.h>
void main()
{
   printf("This is a C program.\n");
```

```
}
```

本程序的作用是输出以下一行信息：

This is a C program

评：本书号称“以 C89 为基础”

对照着 C89 来看

【The function called at program startup is named main. The implementation declares no prototype for this function. It can be defined with no parameters:int main(void) { /* ... */ }

or with two parameters (referred to here as argc and argv, though any names may be used, as they are local to the function in which they are declared): int main(int argc, char *argv[]) { /* ... */ }】——ISO/IEC 9899: 1990 ， 5.1.2.2.1 Program startup 小节

程序输出

This is not a C program.

才对。

p4~p5

```
#include <stdio.h>
void main()
{
int max(int x,int y);
int a , b , c ;
scanf("%d,%d",&a,&b);
c=max(a,b);
printf("max=%d\n",c);
}
int max(int x,int y)
{
int z;
if (x>y) z=x ;
else z=y;
return(z)
}
```

评：很垃圾的代码风格

P6

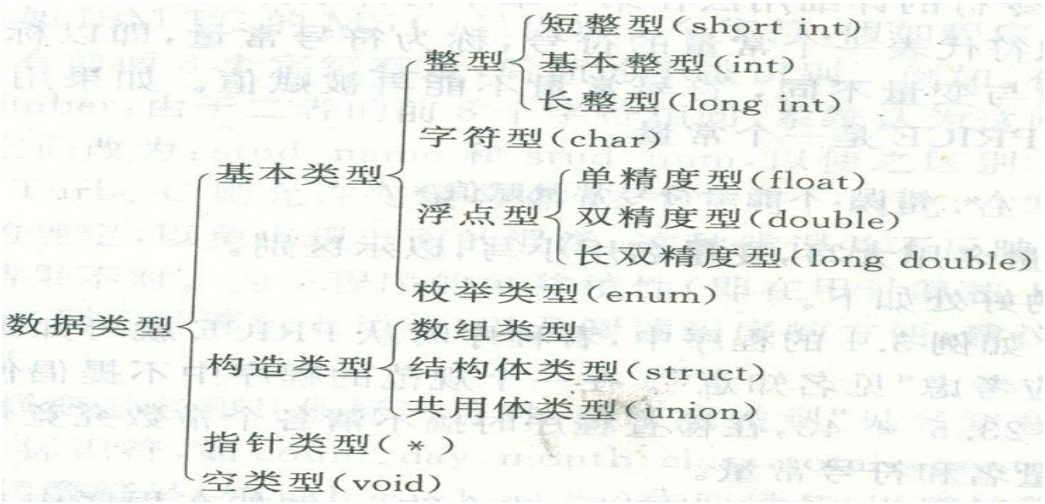
C 的函数库十分丰富，ANSI C 提供一百多个库函数，Turbo C 提供三百多个库函数。
评：关公战秦琼

1966 年，Bohra 和 Jacopini 提出了以下 3 种基本结构,……
评：老谭也有给科学家改名的爱好

P6

C99 标准不支持一行内写多个语句
评：无端臆测
信口开河

P37



评：这个分类，1.不全，2.错误，3.滑稽

P38

变量名实际上是以一个名字代表一个地址
评：int i;
i 是地址，那&i 是什么？

P40

int i; /*定义为整型变量*/
评：从 P37 图中可以看到，“整型”是一个泛称
但是在 p40
到底什么是“整型”？

P40

在 C 语言中，整数常量可用以下 3 种形式表示。
（1）十进制整数，如 123，-456、4。
……
评：-456 是个常量表达式而并不是常量。因为十进制整数常量只容许有 0~9 这十个数字。

P49

3.5 字符型数据
3.5.1 字符常量
C 语言的字符常量是用单撇号括起来的一个字符。如'a'、……
……
3.5.2 字符变量
字符型变量用来存放字符常量，……
……
字符变量的定义形式如下：
char c1,c2;
……
评：这根本就是两种类型？

P50

字符数据以 ASCII 码存储，它的存储形式就与整数的存储形式类似。这样使字符型数据和整型数据之间可以通用。
评：最基本的数据类型都没整明白

P54

图 3-10 中横向向左的箭头表示必定的转换，如字符数据必定先转换为整数，short 型转换为 int 型，float 型数据在运算时一律先转换成双精度型，以提高运算精度（即使是两个 float 型数据相加，也先都化成 double 型，然后再相加）。

纵向的箭头表示当运算对象为不同类型时转换的方向。例如 int 型与 double 型数据进行运算，先将 int 型的数据转换成 double 型，然后在两个同类型（double 型）数据间进行运算，结果为 double 型。

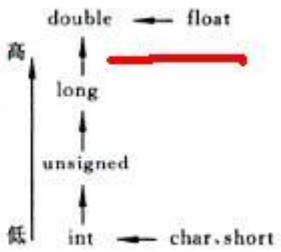


图 3-10

注意：箭头方向只表示数据类型级别的高低，由低向高转换。不要理解为 int 型先转换成 unsigned int 型，再转成 long 型，再转成 double 型。如果一个 int 型数据与一个 double 型数据运算，是直接将 int 型转成 double 型。同理，一个 int 型与一个 long 型数据运算，先将 int 型转换成 long 型。

换言之，如果有一个数据是 float 型或 double 型，则另一数据要先转换为 double 型，运算结果为 double 型。如果参加运算的两个数据中最高级别为 long 型，则另一数据先转换为 long 型，运算结果为 long 型。其他依此类推。

评：如果从 1990 开始算，这个错误存在了 20 年
如果从 1995 年（GB 颁布执行）开始算，这个错误存在了 15 年
我不只在一本大学的教材里看到这个错误
很想知道，那些大学教授究竟是干什么吃的？

P56

在表达式求值时，先按运算符的优先级别高低次序执行，例如先乘除后加减。

评：
不再“求解”而是求值，有进步。
好象有语病。不过咱“语文不好”，暂且存疑。
“先乘除后加减”，太也荒唐。

P56

如果一个运算对象两侧的运算符的优先级别相同，如 a-b+c，则按规定的“结合方向”处理。

C 语言规定了运算符的结合方向（结合性），算术运算符的结合方向为“自左至右”，既先左后右，因此 b 先与减号结合，执行 a-b 的运算，再执行加 c 的运算。“自左至右的结合方向”又称“左结合性”，即运算对象先与左面的运算符结合。

评：荒唐之极

P56

用算术运算符和括号将运算对象（也称为操作数）连接起来、符合 C 语法规则的式子，称为算术表达式。运算对象包括常量、变量、函数等。

评：算术表达式的这个定义相当于什么都没说。更重要的是对可以参与算术运算的操作数没讲清楚。

这里的“函数”不知道是指什么，如果是函数调用表达式，那么把它和常量、变量并列是欠妥的；如果是函数名，那就更加驴唇不对马嘴（见过有人问函数名如何做加法这样荒唐的问题，对此这本书是否有责任呢？），函数名不可能参加算术运算。

即使忽视这个错误，“运算对象包括常量、变量”也是一句废话。试想，参加运算的除了常量、变量还能有什么别的东西吗？

是否可以成为算术运算的运算对象的决定性条件是数据的类型而不是别的什么东西

P58

自增运算符（++）和自减运算符只能用于变量，而不能用于常量和表达式

评：变量本身就是表达式。

P63

……

+=,-=,*=,/=,%=,<<=,>>=,&=,^=,|=

……

C 语言采用这种复合运算符，一是为了简化程序，使程序精练，二是为了提高编译效率（这样写法与“逆波兰”式一致，有利于编译，能产生质量较高高的目标代码。学过编译原理的读者对此容易理解，其他读者可不必深究）。

评：老谭你就不怕读者日后真的学习编译原理？

P64

赋值表达式也可以包括复合的赋值运算符。例如：

a+=a-=a*a

也是一个赋值表达式。如果 a 的初值为 12，此赋值表达式的求解步骤如下：

①先进行“a-=a*a”的运算，它相当于 a=a-a*a，a 的值为 12-144=132。

②再进行“a+=-132”的运算，相当于 a=a+(-132)，a 的值为-132-132=-264。

评：“a+=a-=a*a”这个表达式本身就是错误的，怎么居然还能分析出求解步骤呢？

【Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression.】——ISO/IEC 9899:1999，67 页，6.5 Expressions 之第 2 款；ISO/IEC 9899: 1990 ，6.3 Expressions 小节

【The question is whether the subscript is the old value of i or the new. Compilers can interpret this in different ways, and generate different answers depending on their interpretation. The standard intentionally leaves most such matters unspecified. When side effects (assignment to variables) take place within an expression is left to the discretion of the compiler, since the best order depends strongly on machine architecture. (The standard does specify that all side effects on arguments take effect before a function is called, but that would not help in the call to printf above.)

The moral is that writing code that depends on order of evaluation is a bad programming practice in any language. Naturally, it is necessary to know what things to avoid, but if you don't know how they are done on various machines, you won't be tempted to take advantage of a particular implementation.】K&R(第 2 版),2.12 Precedence and Order of Evaluation

首先我支持一下楼主：这种写法的确不好，但是你必须得告诉初学者：这样不好，并且还要说出不好的原因，这样才有进步，我最开始学 C 语言的时候也是看的谭书，书里面经常写“这样是错误的，那样是错误的”却不给出原因，到底是这么写不好，还是会有编译错误？不得而知。

推荐一下《C 语言常见问题集》和《C 专家编程》两本书，有很多内容很深刻。

其次帮楼主澄清一下：首先并不是能被编译通过的代码就一定没有错误，C 语言标准里面有个东西叫做“未定义”，意思是如果你这么写了，那么编译器无论怎么处理都是正确的：因为 C 语言标准根本就没有定义这么做会怎样！显然“编译通过”也算是处理方式之一，但那不代表这么写是正确的。

a+=a*=a*a;

这个表达式的确是错误的，原因上面楼主说了。顺带说一下，楼主的英文不是随便写的。要注意 C 语言毕竟是老外发明的，而且 C 语言的权威标准也是用英文写的，楼主贴的是 C 语言的标准文档，什么叫标准？意思是如果你不照着那个文档来，你实现的语言根本就不配叫“C 语言”这个名字！这个文档是受法律保护的，因此最好别随便翻译，大家能看懂就看，看不懂可以让别人解释一下下，但是直接翻译是不好的。

楼上有人问顺序点的详细定义，这在标准文档的附录里面写的非常详细，摘录如下：

Annex C
(informative)
Sequence points
1The following are the sequence points described in 5.1.2.3:
—The call to a function, after the arguments have been evaluated (6.5.2.2).
—The end of the first operand of the following operators: logical AND && (6.5.13);
logical OR || (6.5.14); conditional ? (6.5.15); comma , (6.5.17).
—The end of a full declarator: declarators (6.7.5);
—The end of a full expression: an initializer (6.7.; the expression in an expression
statement (6.8.3); the controlling expression of a selection statement (if or switch)
(6.8.4); the controlling expression of a while or do statement (6.8.5); each of the
expressions of a for statement (6.8.5.3); the expression in a return statement

(6.8.6.4).
—Immediately before a library function returns (7.1.4).
—After the actions associated with each formatted input/output function conversion specifier (7.19.6, 7.24.2).
—Immediately before and immediately after each call to a comparison function, and also between anycall to a comparison function and anymovement of the objects passed as arguments to that call (7.20.5).

由此看出，这个表达式在一对顺序点之间改变了 a 两次，因此是个 C 语言根本没定义的操作，换言之：是错误的。

P64

请分析下面的赋值表达式：

(a=3*5)=4*3

先执行括号内的运算，将 15 赋值给 a，然后执行 4*3 的运算，得 12，再把 12 赋给 a。最后 a 的值为 12。读者可以看到：(a=3*5)出现在赋值运算符的左侧，因此赋值表达式(a=3*5)是左值。请注意，在对赋值表达式(a=3*5)求解后，变量 a 得到值 15，此时赋值表达式(a=3*5)=4*3 相当于(a)=4*3，在执行(a=3*5)=4*3 时，实际上是将 4*3 的积 12 赋给变量 a，而不是赋给 3*5。正因为这样，赋值表达式才能够作为左值。

赋值表达式作为左值应加括号，如果写成下面这样就出现语法错误：

a=3*5=4*3

……

评：这论述错的也太离谱了吧，简直令人无法评说

赋值表达式也可以包括复合的赋值运算符。例如：
a+=a-=a*a
也是一个赋值表达式。如果 a 的初值为 12，此赋值表达式的求解步骤如下：
①先进行“a-=a*a”的运算，它相当于 a=a-a*a，a 的值为 12-144=132。
②再进行“a+=-132”的运算，相当于 a=a+(-132)，a 的值为-132-132=-264。
评：“a+=a-=a*a”这个表达式本身就是错误的，怎么居然还能分析出求解步骤呢？

P69~71

C 语句分为以下 5 类。
(1)控制语句。
(2)函数调用语句。
(3)表达式语句。
(4)空语句
(5)复合语句

评：C 语言的最新发展，多出了一种语句

P72

C 语言本身不提供输入输出语句，……例如，printf 函数和 scanf 函数。读者在使用它们时，千万不要误认为他们是 C 语言提供的“输入输出语句”。
P219 例如有输出语句　printf("%d",i);
评：到底是“这个可以有”，还是“这个真没有”？

P73

4.4.1 putchar 函数
putchar 函数（字符输出函数）的作用是向终端输出一个字符。其一般形式为
putchar(c)
它输出字符变量 c 的值，c 可以是字符型变量或整型变量。
评：有网友说见过几乎和老谭的书一模一样的英文书。如果的确是这样，老谭的英语也不过关

P73

下面代码也可能因为“c 是字符型变量或整型变量”才写得那么费劲吧，但 putchar('\n');又与“c 是字符型变量或整型变量”相矛盾

```
例 4.1 输出单个字符
#include <stdio.h>
void main()
  {
    char a,b,c;
    a ='B';b='O';c='Y';
    putchar(a);putchar(b);putchar(c);putchar('\n');
  }
```

评：char a,b,c; 这风格，具有强烈的教唆色彩
　　a ='B';b='O';c='Y';
　　putchar(a);putchar(b);putchar(c);putchar('\n'); 节俭又奢侈，节俭是省纸，奢侈是浪费内存和时间

P85

如果是
scanf("a=%d,b=%d,c=%d",&a,&b,&c);
输入应为以下形式：
a=12,b=24,c=36
采用这种形式是为了使用户输入数据时添加必要的信息，使含义清楚，不易发生输入数据的错误。

评：采用这种形式是为了让用户输入数据时添加不必要的麻烦，使含义似是而非，以便发生输入数据的错误。

P91

在 C 语言中选择结构是用 if 语句实现的。if 语句最常用的形式如下：

if (关系表达式) 语句 1 else 语句 2

例如：

if (x>0)y=1;else y=-1;

评：句句带伤，且多为硬伤。最后一处是很成问题的风格。

P94

5>3&&8<4-!0

表达式自左至右扫描求解。首先处理“5>3”（因为关系运算符优先于&&）。在关系运算符两侧的 5 和 3 作为数值参加关系运算，“5>3”的值为 1（代表真）。再进行“1&&8<4-!0”的运算，8 的左侧为“&&”，右侧为“<”运算符，根据优先规则，应先进行“<”的运算，即先进行 8<4-！0”的运算。现在 4 的左侧为“<”，右侧为“-”运算符，而“-”优先于“<”，因此应先进行“4-!0”的运算，由于“!”的级别最高，因此先进行“！0”的运算，得到结果 1。然后进行“4-！0”的运算，得到结果 3，再进行“8-3”的运算，得 0，最后进行“1&&0”的运算，得 0

评：1.充满错误的想当然

2.即使是作为“想当然”式的幻觉，其本身也是逻辑混乱、自相矛盾的

3.C 语言中从来没有什么“扫描求解”

4.作者压根不懂得“优先级”和“结合性”的含义

”扫描求解“

谭犯得错误把 c 语言语法表达式和 编译过程中的词法分析和语法分析混淆了，估计他根本没搞明白。

P95，96

5.3.1 if 语句的 3 种形式

C 语言提供了 3 种形式的 if 语句。

评：从目录里看到这个标题被吓了一跳，还以为我 C 语言没学全呢。赶忙翻开一瞧

3. if (表达式 1) 语句 1

else if (表达式 2) 语句 2

else if (表达式 3) 语句 3

.

.

.

else if (表达式 m) 语句 m

else 语句 n

评：原来这就是 C 语言提供的第 3 种形式的 if 语句啊

p105

C 语言提供 switch 语句直接处理多分支选择，它的一般形式如下：
switch(表达式)
{
case 常量表达式 1： 语句 1
case 常量表达式 2： 语句 2
.
.
.
case 常量表达式 n： 语句 n
default : 语句 n+1
}p105，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

评：C 语言从来没说过这是“它的一般形式”

P113

6.2 goto 语句以及用 goto 语句构成循环
评：标题就够雷人的了

P113

……但也不是绝对禁止使用 goto 语句。一般来说，有两种用途：
（1）与 if 语句一起构成循环结构。
（2）从循环体中跳转到循环体外，但在 C 语言中可以用 break 语句和 continue 语句（见 6.8 节）跳出本层循环和结束本次循环。goto 语句的使用机会已大大减少，只是需要从多层循环的内层循环跳到外循环时才用到 goto 语句。但是这种用法不符合结构化原则，一般不宜采用。
评：彻底无语。

P120

可见 for 语句功能强，可以在表达式中完成本来应该在循环体内完成的操作。
评：误导。过分夸大 for 语句的作用，后果是许多人只会使用 for 语句而不懂得使用 while 语句或 do-while 语句。

实际上并非所有“在循环体内完成的操作”都可以“在表达式中完成”。试问如果在循环体内的是一 switch 语句或循环语句，又如何能使之在表达式中完成呢？

各种循环语句都在一定条件最为自然，应该选择最适合表达程序思想的循环语句，没什么强不强的。

P156

C 程序的执行是从 main 函数开始的，如是在 main 函数中调用其他函数，在调用结束后流程返回带 main 函数，在 main 函数中结束整个程序的运行。
评：看不懂。有误导之嫌。

P156

函数间可以互相调用，但不能调用 main 函数。main 函数是系统调用的。

评：从没见过这种说法的依据。

P162~163

8.4.2 函数调用的方式

按函数在程序中出现的位置来分，可以有以下 3 种函数调用方式。

1.函数语句

把函数调用作为一个语句。……

2.函数表达式

函数出现在一个表达式中，这种表达式称为函数表达式。……

3.函数参数

函数调用作为一个函数的实参。……

评：这是二年级小学生给一年级小学生讲课。现在我终于知道“通俗易懂”的原因了。

P163~164

（3）如果使用用户自己定义的函数，而该函数的位置在调用它的函数（即主调函数）的后面（在同一个文件中），应该在主调函数中对被调用的函数作声明。p163，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

评：误导

……

```
void main()
   {
    float add(float x,float y);
    ……
   }
float add(float x,float y)
```

{

……

}p164，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

评：只有外行才这样写

P171

……C 语言的特点就在于允许函数的递归调用。例如：

```
int f(int x)
    {
      int y,z;
      z=f(y);
      return(2*z);
    }
```

评：这例子举得也太过随意了吧。是否应该注意一下影响？

形参没用，y 没初值，z 变量多余，无限递归，return(2*z);永远不会执行

好书应该注意处处给人以好的影响，小处不可随便

P179

例 8.11

```
float average(float array[10])
{
……
}
```

P193

1.在一个文件内声明外部变量

……

例 8.20 用 extern 声明外部变量，扩展它在程序文件中的作用域。

```
#include <stdio.h>
void main()
{
int max(int,int);
extern A,B;                    /*外部变量声明*/
printf("%d\n",max(A,B)):
}
```

```
int A=13,B=-8;              /*定义外部变量*/
int max(int x,int y)     /*定义 max 函数*/
{
int z;
z=x>y?x,y;
return (z);
}
```
评：纯吃饱了撑的型

P194

用 extern 声明外部变量时，类型名可以写也可以不写。
评：我很想评价得客气点。但是很遗憾，我最多只能在“信口开河”和“想当然”这两个词中选择一个。

P204

用一个指定的标识符（即名字）来代表一个字符串，它的一般形式为

```
#define 标识符 字符串
```

这就是已经介绍过的定义符号常量，……
评：误导

P219

其实程序经过编译以后已经将变量名转换为变量的地址，对变量值的存取都是通过地址进行的。
评：register int i;i 的地址是什么？

P229

引用数组元素可以用下标法，也可以用指针法，即通过指向数组元素的指针找到所需的元素。使用指针法能使目标程序质量高（占内存少，运行速度快）。
评：看不出这种说法有任何依据。无法证实也无法证伪。

P245

例 10.10

……

```
#define FORMAT "%d,%d\n"
```

……

```
int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
printf(FORMAT,a,*a);
```

……

评：对指针一相情愿的曲解。

P255

```
#include <stdio.h>
void main()
{void copy_string(char from[],char to[]);
   char from[]="I am a teacher.";
   char to[]="You are a    student.";
   char *a=from,*b=to;
   printf("string a=%s\nstring b %s\n",a,b);
   printf("copy string a to string b :\n");
   copy_string(a,b);
   printf("string a=%s\nstring b %s\n",a,b);
}

void copy_string(char from[],char to[])
{int i=0;
   while(from[ i]!='\0')
     {to[ i]= from[ i];i++;}
   to[ i]='\0';
}
```

评：1.a,b 太多余太扎眼了，老谭明显不知道数组名究竟是什么。

2.那个 copy_string()函数定义写得拖泥带水，也就个初学者的水平。

3.void copy_string(char from[],char to[]);，老问题了，不再重说了

P257

虽然用字符数组和字符指针变量都能实现字符串的存储和运算，……

评：我还真不知道“字符指针变量”怎么“实现字符串的存储”

P258

……而常有人用下面的方法：
char *a;
scanf("%s",a);
目的是想输入一个字符串，虽然一般也能运行，但这种方法是危险的，决不应提倡。
评：好一个“决不应提倡”！K&R 若是看到，不知道是否会被气死

btw:如果有人想热心地让我看看这段 C 代码编译后的汇编代码，还是免了吧。我知道这的确可以编译

P265~266

例 10.24 有若干个学生的成绩（每个学生有 4 门课程），要求在用户输入学生序号以后能输出该学生的全部成绩。用指针函数来实现。
程序如下：

```
#include <stdio.h>
void    main( )
   {float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,65}};
     float * search(float (*pointer)[4],int n);
     float *p;
     int i,m;
     printf("enter the number of student:");
     scanf("%d",&m);
     printf("The scores of No.%d are:\n",m);
     p=search(score,m);
     for(i=0;i<4;i++)
        printf("%5.2f\t",*(p+i));
     printf("\n");
   }
float * search(float (*pointer)[4],int n)
     {float *pt;
      pt=*(pointer+n);
      return(pt);
     }
```

评：如果这也叫编程，那无病呻吟是什么

P281~282

```
   struct student
{int num ;
char name[20];
char s e x ;          //对不起，您填写的内容(如签名、帖子、短消息等)包含不良内容而无法提交，请修改。
```

int age;
float score ;
char addr[30];
};
……
struct student student1,student2;
……例如，student1 和 student2 在内存中各占 59 个字节（2+20+1+2+4+30=59）。
评：没想到老谭居然涉黄

（2+20+1+2+4+30=59）是臆测

P310

不能对共用体变量名赋值，也不能企图引用变量名来得到一个值，又不能在定义共用体变量时对它初始话。
评：王小丫的选择题：(a)信口开河 (b)胡说八道

P310

例如，下面这些都是不对的：
①union
{
int i;
char ch;
float f;
}a={1,'a',1.5};(不能初始化)
②a=1 （不能对共用体变量赋值）
③m=a （不能引用共用体变量名以得到一个值）

P310

不能把共用体变量作为函数参数，也不能使函数带回共用体变量，……
评：可怜的共用体，就这样被阉割了

P365

附录 B C 语言中的关键字
……
-bool -Complex -Imaginary
评：这是 C89 中所没有的关键字。若说是 C99 的关键字，又给写错了。这种东西也能弄错，I 服了 U

P377

void 指针具有一般性，它们可以指向任意类型的数据。
评：事实上“void *”类型的指针不指向任何类型的数据。

35-341 楼

本帖最后由 pmerofc 于 2011-09-20 17:03 编辑
LS：谭浩强带起了一堆写*(xx + yy)的恶劣风气。其实 C 标准写得很清楚，*(xx+yy)就等于 xx[yy]，但是很多人就天真地以为用了*才叫用了指针，写 xx[yy]就不是用的指针——真是没什么话好说了= =
starwing83 发表于 2010-04-20 22:09

您说的这个现象，我以前倒是没注意到

但老谭确实对指针的概念不清楚、一知半解甚至存在错误。这是根源！这本书的本质要害是概念不清，甚至是混乱和错误的。而概念“人们对事物本质的认识，逻辑思维的最基本单元和形式”，概念错了，一切都无从谈起。

10.1 地址和指针的概念

p219，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷
.....在 C 语言中，将地址形象化地称为“指针”

p220
指针是一个地址

p221，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

可以看到，这里存在着用“地址”的概念偷换“指针”的概念的错误

和你提到的现象类似
10.3.1 数组元素的指针

p229，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷
10.3.2 通过指针引用数组元素

p230，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

10.3.3 用数组名作函数参数

……

（1）形参和实参都用数组名

（2）实参用数组名，形参用指针变量

（3）实参形参都用指针变量

（4）实参为指针变量，形参为数组名

p235~239，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

在我看来，这些都表明老谭没有真正理解指针的概念、*与[]以及与数组名实参对应的所谓的"形参数组名"数据类型的本质

所以书中多次出现这样的失误是不奇怪的

```
void f(arr[],int n)
```

p236，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

```
void f(x[],int n)
```

p241，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷

题解

P38

6.7 一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子为 1、2、3，而 6=1+2+3，因此 6 是“完数”。遍程序找出 1000 以内所有“完数”，并按下面格式输出其因子“

6 its factors are 1,2,3

评：问题本身就不严格、有漏洞。什么叫“一个数”？分数是不是数？复数是不是数？

问题必须经过严格定义才可能继续求解。有经验的程序员都懂得问题定义的重要性。

我个人很讨厌所谓“一个数”这种大而不当、摸棱两可、含义不清的说法，我认为对于程序员职业来说，这种说法是很严重的误导和毒害。事实上，对于所有的科技职业，这种作风都非常有害。

P38

解：方法一

```
#define M 1000
#include <stdio.h>
```

```
void main()
{
   int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;
   int i,a,n,s;
   /*待续*/
}
```

评：“int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;”绝对是一种教唆，其危害比给孩子们看 A 片还严重。建议“打黄扫非办”先管管这个

许多张白纸，就这样被变成了一张张污浊不堪的草纸。多数草纸必然的命运是废纸！如果有什么比无知更可怕，那就是错误的知识。有人说开卷有益，其实有时开卷有害。

许多人原本不傻，“int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;”能使许多人变傻。

尽管定义了 10 个变量记录因子，但实际上这是错误的。没有任何理由认为小于 1000 的正整数的所有小于其本身的因子不会超过 10 个。举个例子

210 的因子有 1 2 3 5 6 7 10 14 15 21 30 35 42 70 105 一共 15 个因子

所以这个求解的代码是错误的。

P40

（6.7）

方法二

```
#include <stdio.h>
void main()
{
int m,s,i;
    for(m=2;m<1000;m++)
       {s=0;
          for(i=1;i<m;i++)
              if((m%i)==0)s=s+i;
        if(s==m)
          {printf("%d,its factors are",m);
             for(i=1;i<m;i++)
                if(m%i==0)printf("%d",i);
             printf("\n");
          }
       }
}
```

评：很心虚地没给出运行结果。事实上也根本不可能得到题目要求形式的输出“6 its factors are 1,2,3”。

本应该是“指哪打哪”的程序设计变成了“打哪指哪”

P50~51

7.2 用选择法对 10 个整数排序（从小到大）

……

```
void main()
{ int    i , j , min , temp,a[11];
……
```

评：这不是 C，是 BASI—C。

P50~52

7.2 用选择法对 10 个整数排序（从小到大）。

```
……
int i,j,min,temp,a[11];
……
```

说明：定义 a 数组有 11 个元素：a[0]~a[10]，但实际上只对 a[1]~a[10]这 10 个元素输入值并排序。这样符合人们的习惯。

评：这完全是半吊子对初学者的误导。把与 C 语言不符的日常生活习惯在代码加以迎合、迁就、姑息和纵容是绝对学不到真正的 C 语言的。其后果是产生了许多不伦不类、莫名其妙、似是而非的“C 代码”。

C 代码一向以简洁优美著称。1990 年代后，在这个国度生产的垃圾 C 代码的产量达到了空前绝后的程度。至少是 C 语言历史上垃圾代码总和的 1000 万倍

P58~59

7.8 找出一个二维数组中的鞍点，即该位置上的元素在该行上最大，在该列上最小。也可能没有鞍点。

解：一个二维数组最多只有一个鞍点，也可能没有。……

```
#include <stdio.h>
#define N 4
#define M 5

void    main()
{
   int i,j,k,a[N][M],max,maxj,flag;

   printf("please input matrix:\n");
   for(i=0;i<N;i++)
      for(j=0;j<M;j++)
         scanf("%d",&a[ i][j]);
   for(i=0;i<N;i++)
      {max=a[ i][0];
       maxj=0;
       for(j=0;j<M;j++)
          if(a[ i][j]>max)
             {max=a[ i][j];
              maxj=j;
```

```
            }
        flag=1;
        for(k=0;k<N;k++)
          if(max>a[k][maxj])
              {flag=0;
               continue;}
        if(flag)
          {printf("a[%d][%d]=%d\n",i,maxj,max);
           break;
          }
       }
   if(!flag)
        printf("It is not exist\n");
}
```

评：那个 continue 实在很幽默，在废话当中是最幽默的。

更严重的是这个代码是错误的

当输入

please input matrix:

1 2 3 5 5

2 4 6 8 10

3 6 9 12 15

4 8 12 16 20

时，输出的结果竟然是

a[0][3]=5

从这里可以看出

要么是这个矩阵有两个鞍点而程序只能找出一个

要么是程序输出了一个不是鞍点的结果

P70

8.1 求最大公约数

……

```
int hcf(int u,int v)
{
int t,r;
if(v>u)
   {t=u;u=v;v=t;}
while((r=u%v)!=0)
    { u= v ;
      v= r ; }
return (v);
}
```

评：1.if(v>u)

{t=u;u=v;v=t;}
完全是多余的，画蛇添足
2.{}的风格怪异

P71~72

8.2 求方程 a*x*x+b*x+c=0 的根，用 3 个函数分别求当 b*b-4*a*c 大于零、等于零和小于零时方程的根，并输出结果。从主函数输入 a、b、c 的值。

解：程序如下：

```
#include <stdio.h>
#include <math.h>
float x1,x2,disc,p,q;              /*全局变量*/
void main()
{void greater_than_zero(float,float);
   void equal_to_zero(float,float);
   void smaller_than_zero(float,float);
   float a,b,c;
   printf("\ninput a,b,c:");
   scanf("%f,%f,%f",&a,&b,&c);
   printf("equation : %5.2f*x*x+%5.2f*x+%5.2f=0\n",a,b,c);
   disc=b*b-4*a*c;
   printf("root:\n");
   if(disc>0)
     {
      greater_than_zero(a,b);
      printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else    if (disc==0)
     {equal_to_zero(a,b);
        printf("x1=%f\t\tx2=%f\n",x1,x2);
     }
   else
     {smaller_than_zero(a,b);
        printf("x1=%f+%fi\tx2=%f-%fi\n",p,q,p,q);
     }
}

void greater_than_zero(float a ,float b) /*定义一个函数，用来求 disc>0 时方程的根*/
{x1=(-b+sqrt(disc))/(2*a);
   x2=(-b-sqrt(disc))/(2*a);
}

void equal_to_zero(float a ,float b) /*定义一个函数，用来求 disc=0 时方程的根*/
```

```
{
   x1=x2=(-b)/(2*a);
}

void smaller_than_zero(float a ,float b) /*定义一个函数，用来求 disc<0 时方程的根*/
{
   p=-b/(2*a);
   q=sqrt(-disc)/(2*a);
}
```

运行结果：

①input a,b,c:2,4,1

equation : 2.00*x*x+ 4.00*x+ 1.00=0

root:

x1=-0.282893 x2=-1.707107

②input a,b,c:1,2,1

equation : 1.00*x*x+ 2.00*x+ 1.00=0

root:

x1=-1.000000 x2=-1.000000

③input a,b,c:2,4,3

equation : 2.00*x*x+ 4.00*x+ 3.00=0

root:

x1=-1.000000+0.701071i x2=-1.000000-0.701071i

评： 那几个外部变量其丑无比，毫无必要。更滑稽的是这是“函数”部分的习题，这就使滑稽更加滑稽。

注： 函数是实现结构化程序设计的有力工具，全局变量是破坏结构化程序设计最便捷的手段。只有不及格的初学者才会不加思索地使用全局变量。

评： 函数原型写在 main()之内是很外行的写法，在真正的开发中立刻就能发现这样写很愚昧。

注： 函数原型应该写在源文件所有函数定义的最前面。

评： scanf("%f,%f,%f",&a,&b,&c);是很蹩脚的写法

注： 因为比 scanf("%f%f%f",&a,&b,&c); 更容易使用户出错，用户输入更多

评： disc==0，只有不懂编程人才会这样写

注： “10.0 times 0.1 is hardly ever 1.0.”是一个编程常识

评： 几个函数毫无思想，没头没脑，再加上全局变量的配合，绝对是一种变态的思想怪胎。

注： 任何函数都应该能用自然语言描述出其功能。

评： 结果很奇怪，即使是 float 类型，精度也不至于这么差吧。误差已经达到了百分之几的数量级

结论：这只是一个水平非常差的初学者的练习而已

P73

8.4 编写一个函数，使给定的一个二维数组（3×3）转置，即行列互换。

解：程序如下

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

```
#define N 3
int array[N][N];
void main()
{void convert(int array[][3]);
   int i,j;
   printf("input array:\n ");
   for(i=0;i<N;i++)
      for(j=0;j<N;j++)
          scanf("%d",&array[ i ][j]);
   printf("\noriginal array:\n ");
   for(i=0;i<N;i++)
      {for(j=0;j<N;j++)
          printf("%5d",array[ i ][j]);
        printf("\n");
      }
   convert(array);
   printf("\convert array:\n ");
   for(i=0;i<N;i++)
      {for(j=0;j<N;j++)
          printf("%5d",array[ i ][j]);
        printf("\n");
      }
}
void convert(int array[][3])
{int i,j,t;
   for(i=0;i<N;i++)
      for(j=i+1;j<N;j++)
      {t=array[ i ][j];
        array[ i ][j]=array[j][ i ];
        array[j][ i ]=t;
      }
}
```

评：“#define N 3”值得称道。可惜被后面的几个“3”给出卖了。

外部变量“int array[N][N]; ”，非常丑陋，与函数思想直接对立、相互矛盾。

main()中的

```
   for(i=0;i<N;i++)
      {for(j=0;j<N;j++)
          scanf("%5d",array[ i ][j]);
        printf("\n");
      }
```

同样的代码居然一口气连写两次，对于程序员来说简直是奇耻大辱，是根本写不出手的。如果考虑到这是函数部分的习题就更加荒唐。

“void convert(int array[][3])”，既然写了外部变量，这里的参数根本就没有必要，反而使事情更加复杂（同名问题），完全属于自相矛盾不知所云。

更重要的是向函数传递数组压根就不可以这样写，这是这本题解对初学者极大的误导。向函数传递数组通常必须要有两个参数，这是 C 语言的特点。

printf("%5d",array[i][j]);中的“%5d”用的很不讲究，在输入数据较大时输出的内容将会一塌糊涂。

题外话：“convert array:”算不算英语，我很怀疑

P74

8.5 编写一个函数，使输入的一个字符串按反序存放，在主函数中输入和输出字符串。

解：

```
#include <stdio.h>
#include <string.h>
void main()
{void inverse(char str[]);
char str[100];
printf("input string:");
scanf("%s",str);
inverse(str);
printf("inverse string :%s\n",str);
}

void inverse(char str[])
{char t;
int i,j;
for(i=0,j=strlen(str);i<(strlen(str)/2);i++,j--)
   {t=str[ i];
    str[ i]=str[j-1];
    str[j-1]=t;
   }
}
```

评：拖泥带水。

(strlen(str)/2)不仅毫无必要，而且由于毫无意义地被调用多次，可能影响程序的效率（不能假设编译器一定会优化）。

既然实际用到的表达式是 j-1，应该一开始就 j=strlen(str)-1，这样至少 j 有明确的意义。

P76

8.8 编写一个函数，输入一个 4 位数字，要求输出这 4 个数字字符，但每两个数字字符间空一个空格。如输入 1990，应输出“1 9 9 0”。

解：

```
#include <stdio.h>
#include <string.h>
void main()
{char str[80];
```

```
    void insert(char []);
    printf("input four digits:");
    scnf("%s",str);
    insert(str);
}

void insert(char str[]);
{int i;
    for(i=strlen(str);i>0;i--)
        {str[2*i]=str[ i ];
            str[2*i-1]=' ';
        }
    printf("output:\n%s\n",str);
}
```

评："输入一个 4 位数字"，我的理解，应该是对函数功能的描述。然而，函数 insert 并没有实现这个功能。

"4"应该是问题的一个常数，可是在整个代码中没有这个数。所以它要么是多余的，要么程序代码是有问题的。

"要求输出这 4 个数字字符"，要求的只是输出而已，并没有要求改写 。"str[2*i-1]=' ';"不仅罗嗦而且自作多情。一个常识是，完成功能要求以外的功能也是错误。

"printf("output:\n%s\n",str);"如果能输出正确的结果就太神奇了，因为'\0'前面还有个空格。

语文问题："4 位数字"是什么，"数"和"数字"不是一回事儿吧？

P79~80

8.11 编写一个函数用"起泡法"对输入的 10 个字符按由小到大顺序排序。

解：主函数的 N-S 图如图 8-2 所示。sort 函数的作用是排序，其 N-S 图如图 8-3 所示。

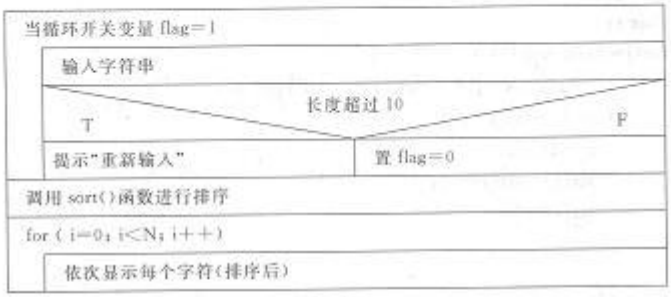


图 8-2

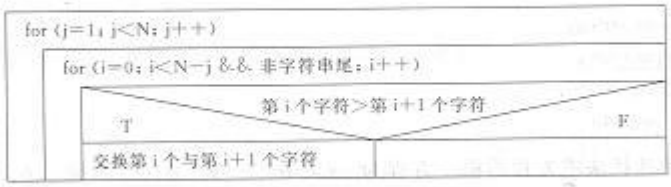


图 8-3

程序如下：

```
#include <stdio.h>
#include <string.h>
#define N 10
char str[N];
void main()
{void sort(char []);
 int i,flag;
 for (flag=1;flag==1;)
  {printf("input string:\n");
   scanf("%s",&str);
   if (strlen(str)>N)
      printf("string too long,input again!");
   else
      flag=0;
  }
 sort(str);
 printf("string sorted:\n");
 for (i=0;i<N;i++)
   printf("%c",str[i]);
 printf("\n");
}

void sort(char str[])
{int i,j;
 char t;
 for(j=1;j<N;j++)
   for (i=0;(i<N-j)&&(str[i]!='\0');i++)
     if(str[i]>str[i+1])
       {t=str[i];
        str[i]=str[i+1];
        str[i+1]=t;
       }
}
```

运行结果：

```
input string:
reputation↙
string sorted:
aeionprttu
```

评：

第一个问题是外部变量

第二个问题是 flag 用的简直变态（印象中这本题解离开了 flag 就不会编程）

第三个和第四个问题是

scanf("%s",&str);

这里至少有两个错误，&str 是严重的概念错误

还有至少 n 个错误……

P91~92

8.17 用递归法将一个整数 n 转换成字符串，例如输入 483，应输出字符串“483”。n 的位数不确定，可以是任意位数的整数。

……

```
#include <stdio.h>
void mian()
{void convert(int n);
   int number;
   printf("input an integer:");
   scanf("%d",&number);
   printf("output:");
   if(number<0)
     {putchar('-');
        number=-number;
     }
   convert(number);
   putchar('\n');
}
void convert(int n)
{
   int i;
   if ( (i=n/10)!=0)
      convert(i);
   putchar(n%10+'0');
}
```

评：这个题目完成的质量算比较高的。可惜，压根就没看到字符串。

“n 的位数不确定，可以是任意位数的整数”是妄语，没有人可以完成这样的程序。在程序设计中，根本就没有“任意”这两个字

P94

9.1 定义一个带参数的宏，使两个参数的值互换，并写出程序，输入两个数作为使用宏时的实参。输出已交换后的两个值。

解：

```
#include <stdio.h>
#define swap(a,b)t=b;b=a;a=t
void main()
{
   int a,b,t;
   printf("input two integer a,b"):
   scanf("%d,%d",&a,&b);
   swap(a,b);
   printf("Now,a=%d,b=%d\n",a,b):
}
```

评：从来没见过这么卑鄙恶劣的宏。如果有宏能展现出宏定义的全部卑鄙与邪恶，那这个宏绝对当之无愧。这个宏创造的最惊人的奇迹在于，它居然被使用了一次。

补充：写成

```
#define swap(a,b)   {t=b;b=a;a=t;}
```

应该是最起码的要求

P134~135

……，编写一个函数 input，用来输入 5 个学生的数据记录。

……

```
#define N 5

struct student
{char num[6];
     char name[8];
     int    score[4] ;
}stu[N];

input(struct student stu[])
{
   ……
   for(j=0;j<3;j++)
   {printf("score %d:",j++);
      scanf("%d",&stu[ i].score[j]);
   }
}
```

评：居然能想到编写输入 5 个学生数据记录的函数！那程序某出如果需要输入 6 个学生数据记录看来还要再编写一个函数喽！

不需要定义外部数组 stu 却偏要定义一个外部数组，理由何在？典型的拿尿盆当饭碗使用。看来有些人不用外部变量就不会写代码有是有根源的。

既然已经定义了外部数组 stu，input()这个函数居然还有参数，滑稽！

printf("score %d:",j++);中的“j++”清楚地表明这段代码的编写者根本不懂编程。哪怕是初级程序员也

不会犯这种低级错误。

P135~138

11.5 有 10 个学生……

评：使用外部数组就不提了

自相矛盾和令人感到蹊跷的是整个程序只有一个 main()

只有一个 main()的程序居然使用外部变量，滑稽！

P146

8.9 编写一个函数，由实参传来一个字符串，统计此字符串中字母、数字、空格和其他字符的个数，在主函数中输入字符串并输出结果。

解：

```
#include <stdio.h>
int letter,digit,space,others;
void main()
{void count(char []) ;
char text[80];
printf("input string:\n"    )   ;
gets(text);
printf("string:    "    )   ;
puts(text);
letter=0;
digit=0;
space=0;
others=0;
count(text);
printf("letter:%d,digit:%d,space:%d,others=%d\n",letter,digit,space,others);
}

void count(char str[])
{int i;
for(i=0;str[ i]!='\0';i++)
if((str[ i]>='a'&& str[ i]<='z')||(str[ i]>='A'&& str[ i]<='Z'))
    letter++;
else if(str[ i]>='0'&& str[ i]<='9')
    digit++;
else if(str[ i]==32)
    space++;
else
    others++;
```

}
评：外部变量就不说了。令人费解的是
letter=0;
digit=0;
space=0;
others=0;
这几句是干吗的？

printf("string:") ;
puts(text);
这两句给人感觉怪怪的，后来看到
1.puts 函数
其一般形式为
puts(字符数组)
p146，《C 程序设计》（第三版），谭浩强 著，清华大学出版社，2005 年 7 月第 3 版，2009 年 10 月第 26 次印刷
原来 puts 函数是输出字符数组的，所以才用 printf 函数输出"string:"

str[i]==32 看着很扎眼

P156

12．1 编写一个函数 getbits，从一个 16 位的单元中取出某几位（即该几位保留原值，其余位为 0）。函数调用形式为
getbits(value,n1,n2)
value 为该 16 位（2 个字节）中的数据值，n1 为欲取出的起始位，n2 为欲取出的结束位。例如：
getbits(0101675,5,8)
表示对八进制 101675 这个数，取出它的从左面起第 5 位到第 8 位。——谭浩强 ，《C 程序设计题解与上机指导（第三版）》，清华大学出版社，2005 年 7 月，p156

```
#include <stdio.h>
void main()
{unsingned short int getbits(unsigned value,int n1, int n2);
unsingned short int a;
int n1,n2;
printf("input an octal number:");
scanf("%o",&a);
printf("input n1,n2:");
scanf("%d,%d",&n1,&n2);
printf("result:%o\n",getbits(a,n1-1,n2));
}

unsingned short int getbits(unsigned value,int n1, int n2)
{unsigned short int z;
z=~0;
```

```
z=(z>>n1)&(z<<(16-n2);
z=value&a;
z=z>>(16-n2);
return(z);
}
```
——谭浩强 ，《C 程序设计题解与上机指导（第三版）》，清华大学出版社，2005 年 7 月，p156

评：

```
07.scanf("%o",&a);
```

这是一个简单的低级错误，且是 UB

```
10.printf("result:%o\n",getbits(a,n1-1,n2));
```

驴唇不对马嘴。n1 减 1，n2 也应该减 1；反过来说，既然 n2 不减 1，n1 也不应该减 1

```
18.z=z>>(16-n2);
19.return(z);
```

愚蠢地违反了题目所要求的“即该几位保留原值，其余位为 0”

```
10.printf("result:%o\n",getbits(a,n1-1,n2));
```

这个也是错误的

P157

写一个函数，对一个 16 位的二进制数取出它的奇数位(即从左边起第 1、3、5……15 位)。

——谭浩强 ， 《C 程序设计题解与上机指导（第三版）》，清华大学出版社，2005 年 7 月，p157

题目要求很奇怪

完全弄不清什么意思

解：

```
#include <stdio.h>
void main()
{unsigned short getbits(unsigned short);
unsigned short int a;
printf("\ninput an octal number:");
scanf("%o",&a);
printf("result:%o\n",getbits(a));
}

unsigned short    getbits(unsigned short value)
{int i,j;
unsigned short int z,a,q;
z=0;
for(i=1;i<=15;i+=2)
   {q=1;
```

```
    for(j=1;j<=(16-i-1)/2;j++)
       q=q*2;
    a=value>>(16-i);
    a=a<<15;
    a=a>15;
    z=z+a*q;
  }
  return(z);
}
```
——谭浩强 ，《C 程序设计题解与上机指导（第三版）》，清华大学出版社，2005 年 7 月，p157

评：

```
02.void main()
03.{unsigned short getbits(unsigned short);
04.unsigned short int a;
05.printf("\ninput an octal number:");
06.scanf( "%o",&a);
07.printf("result:%o\n",getbits(a));
08.}
```

main()至少二处错误

```
10.unsigned short    getbits(unsigned short value)
11.{int i,j;
12.unsigned short int z,a,q;
13.z=0;
14.for(i=1;i<=15;i+=2)
15.    {q=1;
16.     for(j=1;j<=(16-i-1)/2;j++)
17.        q=q*2;
18.     a=value>>(16-i);
19.     a=a<<15;
20.     a=a>15;
21.     z=z+a*q;
22.    }
23.   return(z);
24.}
```

一堆 Magic Number

根本不具备可读性

尤其是

```
20.     a=a>15;
```

简直是天外飞仙

算法笨拙，极其别扭

P158

编一程序，检查一下你所用的计算机系统的 C 编译系统在执行右移时是按照逻辑位移的原则还是按算术右移的原则？如果是逻辑右移，请编写一函数实现算术右移？如果是算术右移，请编写一函数实现逻辑右移。——谭浩强 ，《C 程序设计题解与上机指导（第三版）》，清华大学出版社，2005 年 7 月，p158

居然出现了“位移”，这是力学概念
“原则”两字也不伦不类，应该是运算规则
“请编写一函数实现算术右移？”，标点错误，小学语文没念好
“你所用的计算机系统的 C 编译系统”，啰嗦，不过这回总算分清什么系统了，但说错了。编译器不一定为本机编译，所以“你所用的计算机系统”不但多余而且错误

解：

```
1.  #include <stdio.h>
2.  void main()
3.  {short getbits1(unsigned value,int n);
4.  short getbits2(unsigned short value,int n);
5.  short int a,n,m;
6.  a=~0;
7.  if((a>>5)!=a)
8.  {printf("Logical move!\n");
9.  m=0;
10. }
11. else
12. {printf("Arithmetic move!\n");
13. m=1;
14. }
15. printf("input an octal number:");
16. scanf("%o",&a);
17. printf("how many digit move towards the right:");
18. scanf("%d",&n);
19. if(m==0)
20. printf("Arithmetic right move,result:%o",getbits1(a,n));
21. else
22. printf("Logical right move,result:%o",getbits2(a,n));
23. }

24. short getbits1(unsigned value,int n)
25. {unsigned short z;
26. z=~0;
27. z=z>>n;
28. z=~z;
29. z=z|(value>>n);
30. return(z);
```

```
31.  }

32.  short getbits2(unsigned short value,int n)
33.  {unsigned short z;
34.  z=(~(1>>n))&(value>>n);
35.  return(z);
36.  }
```

——谭浩强 ，《C 程序设计题解与上机指导（第三版）》，清华大学出版社，2005 年 7 月，p158

评：

```
02.void main()
```

多少人被这种错误写法误过

```
03.{short getbits1(unsigned value,int n);
04.   short getbits2(unsigned short value,int n);
```

奇也怪哉
两个函数居然类型不一致

如此低级的错误竟然看不出来？

```
06.   a=~0;
07.   if((a>>5)!=a)
```

无非是
if((-1>>5)!=-1)
而已
弄个 a 做马甲毫无意义

```
15.   printf("input an octal number:");
16.   scanf("%o",&a);
```

这个是错的

```
18.   scanf("%d",&n);
```

又是错的

```
20.       printf("Arithmetic right move,result:%o",getbits1(a,n));
```

实参类型错
导致 getbits()不清楚究竟在操作什么

```
25.short getbits1(unsigned value,int n)
26.    {unsigned short z;
27.      z=~0;
28.      z=z>>n;
29.      z=~z;
30.      z=z|(value>>n);
31.      return(z);
32.    }
33.
34.short getbits2(unsigned short value,int n)
35.    {unsigned short z;
36.      z=(~(1>>n))&(value>>n);
37.      return(z);
38.    }
```

两个函数不用多看就可以知道都是错的
因为就 C 语言的定义来讲
short 类型的数据右移得到的不可能是 short 类型的值