Programming Challenges

The Programming Contest Training Manual

挑 战 编 程

程序设计竞赛训练手册

Steven S. Skiena
Miguel A. Revilla 著
刘汝佳 译

清华大学出版社
北 京

图书在版编目（CIP）数据

挑战编程：程序设计竞赛训练手册/（美）斯基纳（Skiena, S. S.），（西）雷维拉（Revilla, M. A.）著；刘汝佳译. —北京：清华大学出版社, 2009.7
书名原文：Programming Challenges：The Programming Contest Training Manual
ISBN 978–7–302–19797–3

Ⅰ. 挑…　Ⅱ. ①斯…　②雷…　③刘…　Ⅲ. 程序设计　Ⅳ. TP311.1

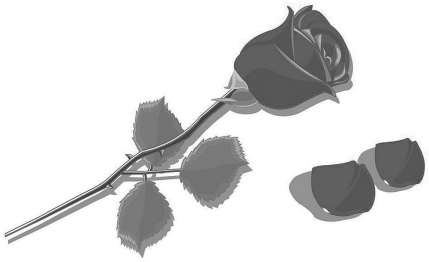
中国版本图书馆 CIP 数据核字（2009）第 045370 号

责任编辑：龙啟铭
责任校对：徐俊伟
责任印制：

出版发行：清华大学出版社　　**地　　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：**100084
社 总 机：010–62770175　　**邮　　购：**010–62786544
投稿与读者服务：010–62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010–62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185×230　　**印　张：**20.25　　**字　数：**450 千字
版　　次：2009年7月第1版　　**印　次：**2009 年 7 月第 1 次印刷
印　　数：
定　　价：0.00元

仅以此书献给我们的妻子们 Renee 和 Carmela，以及孩子们 Bonnie，Emilio 还有 Miguel。和书中的难题相比，抽出时间来陪伴这些我们所爱的人更具挑战。



译 者 序

不管对初出茅庐的新人还是身经百战的老手，用“挑战”一词形容程序设计竞赛是再合适不过的了。酷爱编程的人们往往喜欢挑战，但大多数程序员对各种程序设计竞赛却是“敬而远之”，为什么会这样呢？原因在于，学习编程语言和软件开发的知识只是接受这些挑战的必要而非充分条件。要想在程序设计竞赛中脱颖而出，还需要更多的知识和技能。而这些知识和技能，却是很难在传统的课堂和教科书中学到的。

本书的目标读者便是那些已经具备初步的编程技能，对程序设计竞赛充满好奇，希望有机会武装自己、接受编程挑战的人，以及他们的老师和教练 (甚至父母)。即使不参加任何竞赛，从本书的编程挑战中学到的东西，也会对程序员的职业生涯产生重要影响，更不用说这些挑战本身就是充满乐趣、引人入胜的。

本书文字精练、通俗易懂。尽管每一章都涉及一个不同的领域，但篇幅却短得甚至可以一口气读完。另外，所有题目均附有难度、流行度等客观评价系数，并可以在线提交。写出程序并不意味着完善的解决了难题，只有通过了评测系统的严格把关才能让人信服。

全书由刘汝佳主译，并得到王希、杨锐、尹淳兴、邱前皓等的大力协助。感谢两位作者算法大师 Steven S. Skiena 教授和在线评测系统 UVaOJ 的创立者 Miguel A. Revilla 教授邀请译者完成本书的翻译工作，提供了书的源程序和插图，并讨论书中的一些细节；感谢清华大学出版社的龙啟铭编辑，他对工作认真负责的态度和严谨的科学作风令人钦佩。尽管我们付出了许多努力，但译文中难免有翻译不当之处，敬请批评指正。

前　言

计算机编程能给人带来很多特殊的快乐。付出总会有回报，亲手做出一个有用的东西并看着它成功运转时，你为之满足；灵光一现，于是轻松解决一个困扰你多年的问题时，你为之兴奋；对美的执着追求能让一名普通黑客成为艺术家，而吝啬已然成为了一种美德，尤其是从那些经过锤炼后的精巧算法和简洁代码中榨取最后一滴 “性能之油”。

国际编程竞赛题目中的游戏、谜题和挑战是体验这些快乐的绝佳途径，同时还能提高你的算法能力和编程技巧。本书包含了百余个历届比赛中出现的题目，并讨论了解决这些题目所需的理论和思维方式。读者可以在两个在线评测系统中提交其中任何一道题目的程序，并获得即时的自动评分。若能将评测系统和本书有机结合，你会亲身体会到：迎接挑战和提升编程水平是一件多么新奇、刺激的事啊!

本书可用于自学、讲授算法、编程类创新课程以及比赛的训练。

致读者

本书中的题目选自 Valladolid 大学在线评测系统 (*http://uva.onlinejudge.org/*) 中的上千道编程题目。到目前为止，该系统已经评测了来自 27 000 个注册用户的超过一百万份提交。我们只选择了精品中的精品，即那些最好玩、最刺激和最有意思的题目。

我们把这些题目分成若干个类别，然后提供了足够的教学材料 (主要是数学和算法) 来帮助你解决它们。书中配备了很多参考程序来更好地讲解重要的概念。阅读本书并尝试解决这些题目，你会对回溯法、动态规划等算法技巧以及数论、计算几何等高级话题有更加具体和深刻的理解。即使不参加编程竞赛，关注这些东西也是十分有益的。

多数题目都很生动。它们展示了计算机科学和数学中令人着迷的话题，有时还会以搞笑故事的面目出现。它们往往涉及一些有意思的领域来深入学习研究，因此在合适的时候，我们总是给出注解和进一步的阅读材料。

我们发现，主要接受项目开发和软件工程方面训练的人们通常忽视了算法的重要性。类似地，理论派算法研究者往往低估了把算法转化为程序的难度，也不清楚编程智慧如何化繁为简。

基于上述原因，本书的第一个部分主要关注编程技巧，包括数据类型和库函数的合理运用。这些是书中第二部分那些以算法为主线，难度更大章节的基础。要想成为一名全能的问题求解大师，必须同时精通这两个部分。

致教师

本书被设计为以下三种类型的课程教材：

- 偏重编程的算法课程。
- 偏重算法的编程课程。
- 训练学生参加 ACM/ICPC(ACM 国际大学生程序设计竞赛) 或 IOI(国际信息学奥林匹克) 的选修课。

这样的课程对所有相关人士来说都是愉快的。 学生的潜能很容易被比赛本身的刺激所激发，并且每次提交通过一道题目时都将得到增强。最直白的程序可能会被测试系统返回 "超时" 信息，于是学生会很自然地寻找更有效的解法。正确的洞察力会让一个十余行的小程序取代一大片杂乱无章的代码。最优秀的学生甚至自愿尝试更多的题目来获得快感。

讲授这样的课程也是愉快的。很多题目十分精巧，尽管在算法和编程本质上并无特殊之处，但题目却令人耳目一新。对于这样的问题，要找到最佳解法需要洞察力和灵感。寻找这些题目的解法是令人兴奋的，而当我们所教的学生独立找到解法时，我们会更加兴奋。

从教育学的角度讲，本书的特点包括：

- *作为标准算法教材的补充*—— 尽管本书体系完整，写作时仍然考虑到大多数学生已有了一些算法基础。本书是作为传统算法课程的补充教材来设计 (与定价) 的，它用具体实现补充抽象描述，用实践经验补充理论体系。另外，它还涉及一些多数标准教材中都没有提到的有趣主题。
- *提供经典算法的完整实现*—— 很多学生在把抽象的算法描述转化成能正确运行的代码时困难重重。为了帮助他们，我们精心编写了书中所讨论的所有重要算法的程序。这些程序只用到 C 语言的一个子集，因此 C++ 和 Java 程序员也能轻易看懂。书中的不少习题都只需适当修改这些例程即可解决，从而为初学者提供了一条宽敞的入门之路。
- *内置课程管理系统*—— 我们已经打造了一个特别的课程管理系统，使得管理一门这样的课程易如反掌。所有测试和评分都是自动进行的，你完全不必操心！用我们的网站 *http://www.programming-challenges.com* 可以轻松管理花名册、布置作业、查看学生的程序和分数，甚至还能检测到疑似作业!
- *帮助程度参差不齐的学生*—— 本书在题目选择时有意覆盖了一个较宽的难度范围。多数题目适合初学者，而其他题目即使对于备战国际竞赛的选手来说也是富有挑战的。我们为多数题目提供了解题提示。

  为了帮助学生挑选适合自己的题目，我们给每道题目标注了三种类型的难度值。题目的*流行度* (A, B, 或 C) 是指有多少人提交过此题，而*成功率*(low 表示低，high 表示高) 是指这些人中正确解决此题的比率。最后，题目的*等级* (从 1 到 4，大致表示从新手到高水平读者) 表示解题者应具有的水平等级。

致竞赛选手与教练

本书尤其适合作为高中和大学的编程竞赛培训材料。我们为数学和算法中的重要主题提供方便的总结和参考，还配备了合适的挑战题目来帮助你掌握这些内容。

自动评测系统像 ACM/ICPC 中的真人裁判一样检查所提交程序的正确性。只要你拥有该系统的个人账号，你就可以提交 C、C++、Pascal 或 Java 程序，然后等待系统告诉你这个程序是对还是错。系统还会为你保存一份统计信息，让你有机会与成千上万的其他用户进行对比。

为了帮助竞赛选手，我们请到了三个重要编程比赛——ACM 国际大学生程序设计竞赛 (ICPC)、国际信息学奥林匹克 (IOI)、TopCoder 编程挑战赛 —— 的决赛选手分享经验，并把这些成功的秘密写在附录中。我们还介绍了这三项赛事的历史和参赛方法，最大限度地帮助你施展才华。

大约有 80% 的 ACM/ICPC 决赛选手用 Valladolid 在线评测系统进行训练。ICPC 全球总决赛常常在夏威夷这样令人神往的异国他乡举行，这无疑为选手们提供了额外的精神动力。加油吧!

相关网站

本书有两个配套网站。书中的所有题目均可在 *http://www.programming-challenges.com* 提交，该网站还提供了一些辅助材料，包括书中所有程序的电子版以及帮助教师们把书中的材料融入课堂的课程笔记。

本书中的所有题目 (以及很多其他题目) 也可以在 Valladolid 大学在线评测系统*http://uva.onlinejudge.org/* 中提交。书中的所有习题均同时标注两个评测系统的 ID 号，供读者选择。

致谢

本书的成书在很大程度上归功于那些欣然授权把他们设计的竞赛题目用在本书和在线评测系统中的人们。本书中的题目由来自四大洲的至少 17 位命题者提供，尤其是 Gordon Cormack 和 Shahriar Manzoor，就好比 Sam Loyd 和 H. E. Dudeney[1]一样!

作者和题目的完整列表见附录，但我们要在这里特别感谢如下的竞赛组织者：Gordon Cormack (38 题), Shahriar Manzoor (28), Miguel Revilla (10), Pedro Demasi (8), Manuel Carro (4), Rujia Liu(刘汝佳) (4), Petko Minkov (4), Owen Astrakan (3), Alexander Denisjuk (3), Long Chong(龙翀) (2), Ralf Engels (2), Alex Gevak (1), Walter Guttmann (1), Arun Kishore (1), Erick Moreno (1), Udvranto Patik (1) 以及 Marcin Wojciechowski (1)。其中的部分题目由第三方设计，我们已在附录中向他们致谢。

[1] 译者注：两位都以提出了大量有趣但富有挑战性的数学谜题而著称。

其中一些题目的原作者已经难以考证。我们想尽办法找到每道题目的原作者，并得到了能代表作者发言的人授权。如有疏漏，在敬请原谅的同时还希望能告知我们，以便在致谢中补充。

在线评测系统项目是许多人辛勤劳动的结晶。Ciriaco García 是评测系统软件的主要开发人员，也是项目的核心支持人员。Fernando P. Nájera 开发了一系列辅助工具，使得评测系统更加友好。Carlos M. Casas 维护测试数据，确保它们在正确的前提下既公平又严格。José A. Caminero 和 Jesús Paúl 负责检查和完善题目描述，并测试参考程序。我们还要特别感谢 Miguel Revilla, Jr，他建立并维护着 *http://www.programming-challenges.com*。

本书曾作为 Stony Brook 2002 年春季学期的一门课程的教材。该课程由 Vinhthuy Phan 和 Pavel Sumazin 任教，并修改了诸多错误。我们今年很棒的两支参赛队伍的成员 (Larry Mak, Dan Ports, Tom Rothamel, Alexey Smirnov, Jeffrey Versoza 和 Charles Wright) 帮忙评阅了手稿，感谢他们的兴趣和反馈。Haowen Zhang 仔细阅读了手稿，测试了书中的程序，并简化了代码，为本书的修改做出了重要贡献。

感谢 Springer-Verlag 的 Wayne Yuhasz, Wayne Wheeler, Frank Ganz, Lesley Poliner 和 Rich Putter。没有他们的帮助，本书无法从手稿最终变成出版物。我们还要感谢 Gordon Cormack, Lauren Cowles, David Gries, Joe O'Rourke, Saurabh Sethia, Tom Verhoeff, Daniel Wright 和 Stan Wagon。他们认真仔细的评阅极大地提高了最终成品的质量。Fulbright Foundation 和 Valladolid 大学应用数学与计算系为两位作者面对面交流和一起工作提供了必要的支持。Citigroup CIB，通过 Peter Remch 和 Debby Z. Beckman 所做出的努力，为 Stony Brook 大学的 ACM ICPC 产生了深远的影响。它的参与促成了本书的写作。

Steven S. Skiena
Stony Brook, NY

Miguel A. Revilla
Valladolid, Spain

目　　录

第 1 章　入　　门

在本书的开始，我们将看到一些只用数组和迭代即可解决的基础编程题目。但是，基础并不意味着简单！通过这些题目，读者可以了解到自动评测系统对程序的严格要求，并认识到认真阅读和理解规范的重要性。另外，这些题目也为我们讨论最合适的编程风格提供了一个良好的平台。

为了帮助读者入门，我们首先介绍自动评测系统和它的特性。接下来，在介绍本书的第一组习题之前将讨论基本的编程风格和数据结构。和本书的其余章节一样，我们在习题之后提供了提示和评注，以帮助读者进一步学习。

1.1　初识自动评测系统

本书最好与两个由 Miguel Revilla 管理的自动评测系统结合使用。挑战编程自动评测系统 (The Programming Challenges judge)*http://www.programming-challenges.com*专门为本书而运行，而 Valladolid 大学自动评测系统 (The Universidad de Valladolid judge)*http://uva.onlinejudge.org/* 虽然界面略有不同，却拥有成百上千道本书没有的题目。

本书中的所有题目均可以在两个评测系统中提交，但出于清晰性的考虑，本书对这些题目的文字做了少量修订。尽管两个系统的题目描述存在差异，但本质是相同的。理论上，能在其中一个系统中通过测试的程序也能在另外一个系统中通过。两个系统都在不断发展，因此使用方法可能会有变化。请在使用前阅读两个网站各自的说明[1]。

不管使用哪个系统，第一步都是注册帐号。请注意，两个系统的用户数据目前是独立的，这意味着你需要多花一点时间在两个网站分别注册，才能享受它们各自的特色服务。

挑战编程自动评测系统 (*http://www.programming-challenges.com*) 提供了和本书习题风格一致的环境，还包含了一个特别的课程管理系统，而 Valladolid 大学自动评测系统 (*http://uva.onlinejudge.org/*) 则拥有更多的题目，为读者提供了更大的挑战。二者都可以方便地通过 Web 提交，并能很快看到评测结果。

1.1.1　评测系统反馈

两个系统对于 “正确程序” 的定义都是相当苛刻的，因此正确地理解题目说明 (problem specification) 十分重要。不要对题目说明中没有明确规定的东西做出任何假设。例如，不要

[1] 在本书英文版出版后，Valladolid 自动评测系统已经发生了很大变化。根据原作者的建议，译者根据网站现在的情况重写了本节。

假定输入已排好序、图是连通的或者题目中的整数都是不太大的正数，除非题目说明中作了明确规定。

正如 ACM 国际大学生程序设计竞赛 (ACM/ICPC) 中的裁判一样，自动评测系统只给你极少的反馈信息，包括：

- *Accepted* (AC) — 正确。祝贺你! 你的程序是正确的，并且运行时间和内存开销均不超过相应的限制。
- *Presentation Error* (PE) — 格式错。你的程序输出了正确的结果，但是输出格式并非严格满足题目说明。请检查空格或换行、左右对齐等。
- *Accepted* (PE) — 几乎正确。你的程序有格式错，但是评测系统在给出警告信息的同时仍然认为该程序正确。不要介意，因为很多题目说明中的输出格式规定本身就有着或多或少的歧义。你的程序很可能只是在行末多加了一个空格，因此大可就此打住，宣告胜利。
- *Wrong Answer* (WA) — 答案错。你的程序对一组或者多组评测系统内部的非公开测试数据给出了错误的结果，因此还需要更进一步的调试。
- *Compile Error* (CE) — 编译错。编译器无法成功地编译你的程序，错误信息将返回给用户。编译过程中产生的警告信息并不会导致此错误。
- *Runtime Error* (RE) — 运行错。你的程序在运行结束之前由于段错误 (segmentation fault)、浮点异常 (floating point exception) 或者其他类似的问题异常终止。程序终止时的信息将返回给用户。建议检查无效指针引用 (invalid pointer reference) 或者除零错误 (division by zero)。
- *Time Limit Exceeded* (TL) — 超时。你的程序在某一组或多组数据上运行的时间过长，因此该程序可能存在效率问题。注意，在一组数据上超时并不意味着在其他数据上正确!
- *Memory Limit Exceeded* (ML) — 超内存。你的程序试图使用超过评测系统规定大小的内存。
- *Output Limit Exceeded* (OL) — 超输出。你的程序输出了过多的信息。这通常意味着你的程序陷入了一个带输出的无限循环中。
- *Restricted Function* (RF) — 你的程序试图使用一些被测试系统禁止的非法的函数，例如`fork()`或`fopen()`。请不要使用这些函数。
- *Submission Error* (SE) — 你错误地指定了一个或多个提交信息域，例如错误的 ID 或者题目编号。

再次强调：如果你的程序被认为是错误的，评测系统并不会告诉你错在哪个数据，也不会告诉你为什么是错的。即使你对你的程序*很有把握*，评测系统仍然可能始终对它说不。也许这只是因为你忽略了一个边界情况或者做了一个不成立的假定，但是不对代码进行修改就直接重新提交在绝大多数情况下都是不可取的。再次仔细读题，进一步确认题目的意思和

你先前所理解的是否完全一致。

系统有时也会返回一些上面没有列举出来的特殊反馈信息。这些信息往往和你的程序无关，请阅读网站上的说明。

1.2 挑选你的武器

一般来说，你应当选择你最熟悉的语言来写程序。评测系统目前接受 C、C++、Pascal 和 Java 语言的程序。对于大多数人来说，最熟悉的编程语言已经包含在其中了。这些语言各有自身擅长的领域，但是就本书所讨论的问题求解 (problem-solving) 而言，所有四种语言都可以很好地完成任务。问题求解的关键在于算法，而语言之间的可移植性、模块性和效率等方面的差异已经变得不那么重要了。

1.2.1 程序设计语言

评测系统所支持的四种语言是在四个不同的时期，以不同的目标为导向所设计出来的：

- *Pascal* — 作为 20 世纪 80 年代教育领域最流行的编程语言，Pascal 语言鼓励结构化的编程习惯。尽管它已经不再流行，甚至几乎处于消亡的边缘，但在东欧和中学教育中仍然处于重要地位。
- C — 作为 UNIX 操作系统的原始语言，C 语言为有经验的程序员提供了足够强大的力量来做任何他们想做的事，包括用无效的指针引用和强制类型转换来让程序崩溃。20 世纪 90 年代，面向对象编程技术的进步使得 C 语言逐步发展成了一个崭新的，更先进的 ···
- C++ — 第一个成功的、商业化的、面向对象的程序设计语言在成功实现对 C 语言向后兼容的同时引入了新的数据抽象和继承机制。C++ 成为了 20 世纪 90 年代中后期教育和工业领域的主流语言，但它的地位慢慢地被另一个语言所动摇 ···
- *Java* — 作为一门功能强大的通用程序设计语言，它拥有一个特殊的安全机制来避免一些常见的编程错误，如数组下标越界或者非法指针访问。

注意上面所提到的每种编程语言都有一些与编译器和系统相关的特性，因此在你的本地机器上可以正确运行的程序不一定能在评测系统中运行。请阅读评测系统中关于编程语言的说明，尤其是当你使用 Java 的时候。

到 2002 年 12 月为止，评测系统收到了超过 1 250 000 份提交。其中接近一半的程序用 C++ 语言写成，另外还有三分之一是 C 语言。只有很少部分的提交使用 Java 语言，但请注意系统从 2001 年 11 月才开始支持 Java 语言。

图 1.1 把这些提交更加细致地按月统计。早期 C 一直是最受欢迎的语言，但在 1999 年底被 C++ 超过。从整体上看，系统的繁忙程度逐年增加。

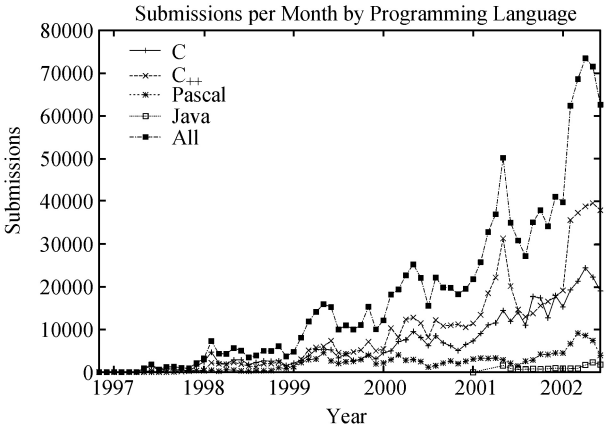


图 1.1　2002 年 12 月之前自动评测系统所接收到的提交

表 1-1　2002 年 12 月为止评测系统给出的结果

语言	总数	AC	PE	WA	CE	RE	TL	ML	OL	RF
C	451447	31.9%	6.7%	35.4%	8.6%	9.1%	6.2%	0.4%	1.1%	0.6%
C++	639565	28.9%	6.3%	36.8%	9.6%	9.0%	7.1%	0.6%	1.0%	0.7%
Java	16373	17.9%	3.6%	36.2%	29.8%	0.5%	8.5%	1.0%	0.5%	2.0%
Pascal	149408	27.8%	5.5%	41.8%	10.1%	6.2%	7.2%	0.4%	0.4%	0.5%
总计	1256793	29.7%	6.3%	36.9%	9.6%	8.6%	6.8%	0.5%	1.0%	0.6%

1.2.2　如何阅读本书的程序

本书中包含若干编程实例，用于展示一些编程技巧和基础算法的完整实现。所有的代码均可以在*http://www.programming-challenges.com*上下载，供读者使用和实验。这些程序被成千上万的学生使用和阅读，因此建议读者到该网站查阅最新的勘误表和代码的修订版本。

本书的所有代码均用 C 语言的一个子集实现。我们希望这样的代码可以比较容易地被所有读者理解。C 本身是 C++ 的一个子集，语法和 Java 有不少的相似之处。我们在本书的所有代码中都刻意回避了 C 语言所特有的语法以及指针和动态内存分配，而剩下来的部分对于所有四种语言的用户来说，都应当是熟悉的。

为了帮助读者更好地阅读本书的程序，下面是一些提示：

- *参数传递* — C 语言统一采用传值 (call-by-value) 方式来进行参数传递。这*看上去*似乎意味着函数不能有任何副作用，但实际并非如此。如果你想修改某一个参数，可以把指向它的指针传递给一个函数。尽管函数无法修改这个指针，但它可以修改该指针所指向的内容，也就是你想修改的那个参数。

  在本书中，指针唯一的作用就是参数传递。指向 x 的指针写作 &x，而指针 p 所指向的内容写作 *p。这里的星号代表 “取内容” 运算，而不是算术乘法，请读者注意不要

混淆。

- *数据类型* — C 语言支持一些基本数据类型，包括 `int`、`float` 和 `char`。更高精度的整数和浮点数分别为 `long` 和 `double` 类型。未说明返回类型的函数均返回 `int` 类型。
- *数组* — C 语言的数组下标范围总是 0 到 $n-1$，其中 n 是数组中的元素个数。因此如果为了使用方便而用 1 作为起始下标，最好是定义一个 $n+1$ 个元素的数组。C 语言不提供运行时刻的数组下标范围检查，因下标越界而引起的程序崩溃非常常见。

  在实际应用中，第一个元素的位置并不总是一致的。尽管从 0 开始是 C 语言的传统风格，但有时候把 1 作为开始会让代码更加清晰和简单。在这种情况下，我们愿意为此浪费一个内存单元，来获取这样的简洁性。请读者在阅读本书代码时注意这一点，不要被这种处理方法所迷惑。
- *运算符* — C 语言有一些看上去很奇怪的运算符。例如，整数取余 (即取模) 运算符为%，在条件语句中常见的逻辑与 (logical-and) 和逻辑或 (logical-or) 运算分别为 `&&` 和 ||。

1.2.3 标准输入输出

UNIX 程序员大都对流式输入输出的管道程序十分熟悉，它们的输出经常会作为另外一些程序的输入。人们习惯于把很多这样的小程序串在一起来完成任务，而不是制造一个复杂的、功能完善的软件系统并试图覆盖所有的应用。

这样的 “小工具哲学” 近来受到了图形用户界面 (graphical user interface) 的冲击。很多程序员习惯于为每个程序包装上一个 “指向 - 单击” 的图形界面，但这样将使程序之间的数据传输变得非常困难。处理程序的文本输出是很容易的，但面对一张图片的时候，除了 “用眼睛看” 之外，你还能做什么呢？

和很多正规的 ACM/ICPC 比赛一样，评测系统要求程序从标准输入 (standard input) 读取数据，并把结果打印到标准输出 (standard output) 中。程序不允许打开任何文件，也不能执行一些特定的系统调用，如进程和网络相关的函数[2]。

标准输入输出 (standard input/output) 的使用在 C、C++ 与 Pascal 语言中都是很简单的。图 1.2 展示了如何用这三种语言分别写一个程序，每行读取两个整数，打印它们之差的绝对值，直到文件结束。请读者注意不同语言是如何判断文件结束的。大多数题目通过指定测试数据个数或者规定数据结束标记来避免程序显式的判断文件结束。

大多数语言都提供了强大的格式化 I/O 函数。如果善加利用，可能只需要一行代码就能完成某些看上去很复杂的输入输出任务。这些任务对于既不熟悉这些函数、又不看手册 (manual) 的程序员来说通常意味着痛苦的解析 (parsing) 和格式化 (formatting)。

可惜的是，Java 的标准输入输出并不简单。*http://www.programming-challenges.com* 包含了一个 35 行长的 Java 输入输出模板以供参考，而其他注意事项可以在*http://uva.*

[2] 如果你希望在本地采用文件输入输出来测试程序，而在评测机器上自动改为标准输入输出，只需测试 `ONLINE_JUDGE` 符号是否已定义。

*onlinejudge.org/*找到。

```c
#include<stdio.h>

int main() {
   long p,q,r;

   while (scanf("%ld %ld",&p,&q)
               !=EOF) {
     if (q>p) r=q-p;
     else r=p-q;

     printf("%ld\n",r);
   }
}
```

```cpp
#include<iostream.h>

void main()
{
long long a,b,c;

while (cin>>a>>b) {
   if (b>a)
      c=b-a;
   else
      c=a-b;
   cout << c << endl;
}
}
```

```
{$N+}
program acm;
var
  a, b, c : integer;
begin
  while not eof do
  begin
    readln(a, b);
    if b > a then
    begin
      c := b;
      b := a;
      a := c
    end;
    writeln(a - b);
  end
end.
```

图 1.2 C (左)、C++ (中)、Pascal (右) 中的标准输入输出

1.3 编 程 提 示

本书不是要教你如何编程，而是要教你如何更好地编程。我们假定你已经熟悉了一些基本概念，如变量 (variable)，条件语句 (例如 `if-then-else`, `case`)，循环语句 (例如 `for-do`, `while-do`, `repeat-until`)，子程序 (subroutine) 和函数 (function)。如果你对这些概念并不熟悉，也许你挑错了书。但我们仍然建议你把本书买下来备用。

你应当弄清楚你已经学会的东西到底有多大威力。在理论上，所有有趣的程序和算法都可以由你在第一门程序设计课程中学到的东西构建起来。现代编程语言的强大特性对于构建这些 “有趣的东西” 并不是必须的，它们只是让这个构建过程更清晰、更合理。

打个比方，成为优秀作家的关键不是持续地学习新词汇，而是努力寻找自己想说的。同样的，学习了一两门程序设计课程之后，你好比是已经学会了让别人能听懂自己所需的所有单词。剩下的问题正是本书所要努力解决的：找一些有趣的东西来说。

我们提供一些底层的编程技巧来帮助读者编写高质量的程序。下面将看到的例子均来自于评测系统接收到的实际提交代码。

- *先写注释* — 在写程序或者子程序之前首先用几句话描述你将要写出的代码应完成怎样的任务。这一点很重要，因为如果你不能轻易的写出这些描述性注释，你很可能并不十分清楚你到底想做什么。一般来讲，调试注释比调试程序要容易得多，而且这部分的时间花费是值得的。当然，在比赛现场的紧张环境和压力下，选手们往往不肯为写注释

而花时间，但在节省这一点微不足道的时间的同时，风险也随之而来。

- *给每个变量编写注释* — 在声明每个变量的时候写一个单行注释来表明它的用途。同样的道理，如果你无法轻易地描述这个变量，你甚至可能会怀疑这个变量存在的必要性。由于你很可能会花一点调试时间来与这个程序作进一步的近距离接触，用这个方法适度提高程序的可读性以减少调试时间是很划算的。
- *使用符号来表示常量* — 无论何时，只要你需要使用常量 (输入规模、数学常量、数据结构大小等) 时，就应在程序的开头部分以符号的形式定义它们。在程序中使用不一致的常量可能会引入极其隐蔽的错误。当然，你应当记得在程序中每个需要使用常量的地方引用相应的符号而不是这些常量本身，否则符号常量就失去了它们的意义 ···
- *不要滥用枚举类型* — 枚举类型 (即符号变量，例如布尔型 (`true`, `false`)) 能提高复杂程序的可读性，但在短程序中往往是不必要的。下面的代码表示了一堆牌的花色，即草花 (club)、方块 (diamond)、红心 (heart) 和黑桃 (spade)：

```
switch(cursuit) {
    case 'C':
      newcard.suit = C;
      break;
    case 'D':
      newcard.suit = D;
      break;
    case 'H':
      newcard.suit = H;
      break;
    case 'S':
      newcard.suit = S;
...
```

  与原始的字符表示法 (`'C','D','H','S'`) 相比，枚举类型 (`C,D,H,S`) 的引入不仅没有提高可读性，反而带来了更多的出错机会。

- *用子程序来避免冗余代码* — 请读者阅读下面一段维护矩形棋盘状态的代码，思考你将如何缩短并简化它：

```
...

     while (c != '0') {
        scanf("%c", &c);
        if (c == 'A') {
            if (row-1 >= 0) {
```

```
                temp = b[row-1][col];
                b[row-1][col] = ' ';
                b[row][col] = temp;
                row = row-1;
            }
        }
        else if (c == 'B') {
            if (row+1 <= BOARDSIZE-1) {
                temp = b[row+1][col];
                b[row+1][col] = ' ';
                b[row][col] = temp;
                row = row+1;
            }
        }
...
```

在完整的程序中，一共有四个这样的代码段，每个代码段有三行，但作用仅仅是把一个值移动到某个相邻的单元格中。这段代码不仅长，而且把其中任何一个 + 或者 − 敲错都可能是致命的。相反，如果写一个 "移动并交换" 的子过程，只需要用正确的参数调用四次即可。

- *输出有意义的调试信息* — 花一点时间学习如何在你的系统中调试程序，例如在特定的语句或条件下中断程序运行，然后检查所有相关变量的值。一般来说这比加入大量输出语句的方法更快更简单，但如果你*确实*需要输出调试信息，应尽量让这些信息有意义。输出所有相关变量，并在打印数值之前*标注*相应的变量名，否则你将会很容易迷失在大量无意义的调试信息中。

大多数学习计算机科学的学生习惯于使用面向*对象程序设计* (*object-oriented programming*)。尽管它在构建大型的、可重用的程序方面非常有用，但本书中的绝大多数程序只需要短小但精巧的程序即可解决。面向对象程序设计中的一些基本假定在本领域中并不成立，因此定义一些复杂的新对象 (*使用*已有对象不算) 一般来说是浪费时间。

请注意，我们并不是要大家摒弃良好的编程风格，而是提倡根据任务规模选择合适的编程方式。这才是成功的关键。

1.4　基本数据类型

数组与链表相比有一个重要的优势：简单。很多在基于指针的数据结构中经常出现的错误根本不可能在静态数组中出现。

职业选手的成熟标志是能保持问题的简单性。这对于刚刚接触一个新领域的人来说相当具有挑战性。有一个很经典的例子：在学习完一系列的热带疑难杂症之后，一个年轻的医学博士担心患有鼻塞和疹子的病人都有可能感染依波拉病毒 (Ebola virus) 或者淋巴腺鼠疫 (bubonic plague)，但有经验的医生只会给这样的病人开一瓶阿司匹林 (aspiron)。

同样地，你可能最近刚刚学会平衡排序二叉树 (balanced binary search tree)、异常处理 (exception handling)、并行处理 (parallel processing) 以及对象继承 (object inheritance) 的若干模型。这些都很有用、很重要，但并不见得是解决一个简单问题的最佳途径。

因此，尽管基于指针的数据结构在你无法预知最大输入规模，或者需要支持快速查找与更新的时候十分有用，但本书中的很多题目已经指明了最大规模，而且时限相当宽松。你的程序并不会因为特别快速而获得额外的奖励。

那么，什么才是简单、成熟的通往数据结构之路呢？首先，要熟悉程序设计语言中内置的原始数据**类型**。理论上，你可以用下述类型构造出任意的数据结构：

- **数组** — 数组允许我们按位置而非按内容访问数据，正如门牌号码让我们按照地址而非户主名字来访问。数组用来存放单一类型的元素序列，如整数，实数或者记录 (record)[3] 等复合对象。字符数组可以用来表示文本串，而文本串数组几乎可以用来表示任何东西。

  哨兵 (*sentinel*) 是一个用来简化数组访问的常见技巧。哨兵是一个特殊元素，在不用显式测试下标范围的情况下保证程序不会访问到非法地址。考虑在一个包含 n 个元素的有序数组 `a` 中插入元素 x 的情形。我们可以像左边的代码那样显式的测试下标是否到达最大允许值：

```
i = n;
while ((a[i]>=x) && (i>=1)) {
        a[i+1] = a[i];
        i=i-1;
}
a[i+1] = x;
```

```
i = n;
a[0] = SMALLINT
while (a[i] >= x) {
        a[i+1] = a[i];
        i=i-1;
}
a[i+1] = x;
```

  也可以像右边的代码那样在 `a[0]` 处设置一个保证比所有元素小的整数 `SMALLINT` 作为哨兵。合理的使用哨兵并确保数组的大小略大于它实际被使用的部分往往可以避免很多边界错误。

- **多维数组** — 尽管在提到二维数组时，人们总是会不自觉地想到棋盘、图像这样的矩形网格结构，但更一般地，多维数组可以用来聚集**同构** (*homogeneous*) 数据。例如，$x-y$ 平面上的 n 个点所组成的数组可以被看作一个 $n \times 2$ 数组，其中 `A[i][j]` 中的第二维 (0 或 1) 表示该元素是相应点的 x 坐标还是 y 坐标。

[3] 译者注：这是 Pascal 语言的概念，在 C/C++ 语言中对应于结构体。

- **记录** — 记录用来聚集**异构** (*heterogeneous*) 数据。例如人事记录把名字、身份证号码、身高和体重捆绑到一起。在大程序中，记录往往对整个程序的概念清晰性起到了重要作用，但在中等规模的程序中也许把各个域分解成多个独立的数组会更加方便。

  用记录还是用多维数组，哪个更好呢？这个问题的答案并不总是那么清晰。还是回到刚才所讨论过的 $x-y$ 平面上点的表示。最显然的表示莫过于下面的记录 (结构体)，而不是一个双元素数组：

```
struct point {
    int x, y;
};
```

  记录表示法的主要优势在于 `p.x` 和 `p.y` 看上去更加接近于我们的日常用法，但问题在于你无法方便地遍历一个记录中的各个变量。而在数组中，遍历元素是非常方便的。

  假设你准备把程序扩展到支持三维、甚至任意维空间中。当然，你可以简单地在记录中增加一个域，但接下来你还需要修改几乎所有的函数，把 x 和 y 分量所做的事情对于 z 分量重复一遍。另一方面，如果你采用的是数组表示法，则把距离函数从二维改到三维仅需修改一个常量：

```
typedef int point[DIMENSION];

double distance(point a, point b)
{
    int i;
    double d=0.0;

    for (i=0; i<DIMENSION; i++)
        d = d + (a[i]-b[i]) * (a[i]-b[i]);

    return( sqrt(d) );
}
```

在第 2 章中，我们将会看到一些相对高级的数据结构，它们都可以由这些原始数据类型构造出来。这些数据结构将让我们工作在一个更高的抽象层次中，但当简单数据类型完全可以胜任时，不要害怕使用它们。

1.5 关 于 习 题

本书每章的末尾均有一个富有挑战性的编程题集。这些题目都是从 Valladolid 评测系统的上千道题目中精选出来的清晰而精巧的、难度各异的问题。我们尤其关注这些题目中闪现