计算机科学丛书

Pearson

原书第4版

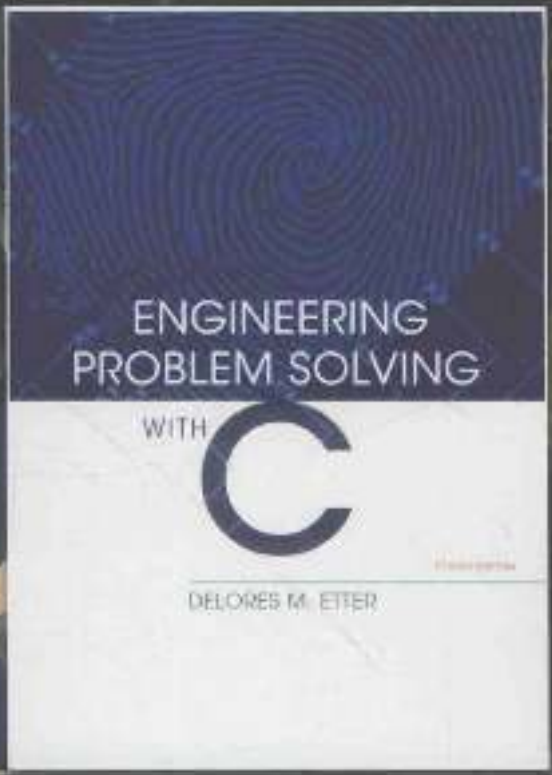
工程问题C语言求解

[美] 德洛莉丝 M. 埃特尔（Delores M. Etter）著

宫晓利 周阳 张金 译

Engineering Problem Solving with C

Fourth Edition



机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

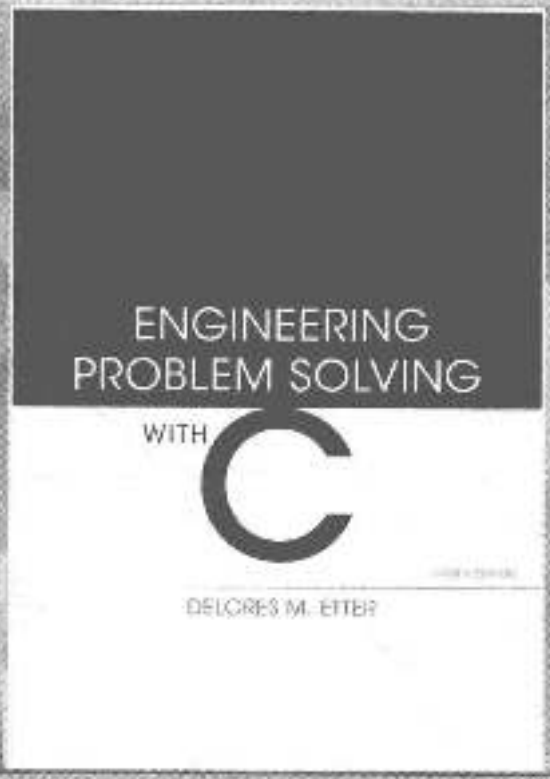
原书第4版

工程问题C语言求解

[美] 德洛莉丝 M. 埃特尔（Delores M. Etter）著
宫晓利 周阳 张金 译

Engineering Problem Solving with C

Fourth Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

工程问题C语言求解（原书第4版）/（美）德洛丽丝 M. 埃特尔（Delores M. Etter）著；[illegible]，周阳，张命译．—北京：机械工业出版社，2016.12
（计算机科学丛书）
书名原文：Engineering Problem Solving with C, Fourth Edition

ISBN 978-7-111-55441-7

Ⅰ. 工… Ⅱ. ①德… ②[illegible]… ③周… ④张… Ⅲ. C语言－程序设计 Ⅳ. TP312.8

中国版本图书馆 CIP 数据核字（2016）第 281000 号

本书版权登记号：图字：01-2012-4855

Authorized translation from the English language edition, entitled Engineering Problem Solving with C, 4E, 9780136085317 by Delores M. Etter, published by Pearson Education, Inc., Copyright © 2015, 2007.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Chinese simplified language edition published by Pearson Education Asia Ltd., and China Machine Press Copyright © 2017.

本书中文简体字版由 Pearson Education（培生教育出版集团）授权机械工业出版社在中华人民共和国境内（不包括香港、澳门特别行政区及台湾地区）独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

本书封底贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

本书介绍如何使用C编程语言来解决工程问题。书中从适用5步方法论入手，以犯罪现场调查、航班导航、飓风等级测量等众多热点技术领域的工程问题为应用对象，生动、有趣地讲解了C语言中的基本操作符、标准输入/输出、基本函数、控制结构、数据文件、模块化编程、数组以及指针等编程概念。

本书内容翔实，具有很强的操作性和实践性，可作为高等院校工程和科学计算相关专业的教材，也可作为初学者在C语言编程组织与实际工程应用之间搭建桥梁的参考书。

出版发行：机械工业出版社（北京市西城区百万庄大街22号 邮政编码：100037）
责任编辑：朱劲英　　责任校对：殷　虹
印　　刷：北京市密云县彩色印刷有限公司　　版　　次：2017年1月第1版第1次印刷
开　　本：185mm×260mm　1/16　　印　　张：25（含0.25印张彩插）
书　　号：ISBN 978-7-111-55441-7　　定　　价：79.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379891　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

版权所有·侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所　韩光/邹晓东

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还展示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brian W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

华章科技图书出版中心

译者序

Engineering Problem Solving with C, Fourth Edition

第一次技术革命让我们摆脱了终日与自然抗争的求生之路，创造了农耕文明。

第二次技术革命让我们超越了身体的物理极限，开创了蒸汽工业社会。

第三次技术革命让我们释放了思想，开启了信息与数据的时代。

未来是一个深度融合的时代，人与人之间的思想通过互联网相互汇聚形成前所未有的逻辑虚拟空间，物与物之间通过物联网罗织成物理世界在虚拟空间中的完整映射，最终虚拟空间将与物理世界相重合，从而使得人的智慧与机器的能力相互交叉融合，进入人类社会的全新纪元。

因此，对于计算机尤其是"计算思想"的认知和运用就变得极其重要。而编程语言正是运用计算思想与机器沟通的重要工具，或者说是与计算机这种"另类生物"的沟通方法。那么如何学习编程语言呢？其实只是很简单的一句话——"无他，唯手熟尔。"我们常常困惑，即使是看过一本本厚厚的编程书籍，但仍然不能掌握编程方法，甚至对于编程概念都感到模糊。究其原因在于没有建立计算思想，即没有形成利用编程思维去解决实际问题的习惯。在编程的学习过程中，通过反复训练形成编程思维，建立运用计算思想去解决问题的习惯尤为重要。这也是我们翻译并推介美国著名学者 Delores M. Etter 所著的这本书的重要原因。

与传统的编程语言教材不同的是，本书从一个个技术领域的趣味性问题入手，利用计算思想提炼和抽象问题之后，再运用 C 语言的案例作答；从而在形成计算思想和编程思维训练的同时，帮助读者熟悉编程语言的运用。这些趣味性的问题涵盖了当今大多数的热点技术领域，例如犯罪现场调查、海水冰点、速度计算、波互作用、臭氧测量、冰山追踪、仪器可靠性、系统稳定性、飓风等级测量、分子量、地形导航、电路分析、地震监测和地表风向等。这种从趣味性的真实问题入手，引入相关的语法知识，通过设计有效的算法来求解问题，最后对求解方法加以验证的科学讲授方法，令初学者能够快速建立起 C 语言的知识体系，同时还能通过求解实际问题对计算思想加深理解，使得原本单调的语法和代码跃然纸上，这是很多教材中刻板的教学方法和玩具式的示例程序难以望其项背的。

计算机编程虽属工科范畴，但从本质上讲，它跟世界上任何一门人类语言一样，在漫长的学习过程中，唯有勤思多练，才能逐渐领悟精髓。就好比研究汉语言的学者，他们对汉语的钻研早已融入长年累月的工作、学习和交流之中，因此自然口吐莲花、谈吐不俗。所以，经过长期而专注的阅读、思考、训练和总结，终有一天你会由"渐悟"走向"顿悟"，从"不得其法"走向"触类旁通"。

在这里要衷心感谢南开大学计算机与控制工程学院的刘晓光、王刚、李忠伟、张海威、任明明以及其他老师在本书翻译过程中提供的编程经验和技术指导。同时还要感谢谢彦茜、李欣、张瑞和杨皓翔四位同学对本书文字及内容的认真审阅。此外，还要感谢南开大学区人式与信息安全实验室的全体同学在全书的翻译过程中提供的支持与帮助。

限于译者的水平和经验，译文中难免存在不当之处，恳请读者提出宝贵意见。

译者

2016 年 11 月于马鞍湖畔

从简单函数估计到非线性方程组求解，工程师需要利用计算机解决各种各样的问题。为完成这些工作，C 语言已经成为许多工程师和科学家的选择，不仅是因为它强大的指令和数据结构，而且还因为它很容易被用于实现系统级操作。既然 C 语言是许多新入行的工程师们在工作中不得不面对的编程语言，那么我们就在此对 C 语言做一个详细介绍。本书将包括以下内容：

- 展示一种用于求解工程问题的通用方法。
- 对 C 语言基础进行详细介绍，因为 C 语言已经成为众多工程师和科学家的重要工具。
- 通过提供多种多样的有趣的工程实例和应用，说明使用 C 语言求解问题的过程。

为了清晰地表述以上内容，本书第 1 章介绍了解决工程问题的 5 步过程，这在本书后面的内容里会一直用到。第 2 – 7 章对使用 C 语言来求解工程问题进行了基本介绍。第 8 章简单介绍了使用 C++ 进行面向对象的程序设计，因为面向对象程序设计在工程和科学的诸多领域中日渐普及，并且很可能会在以后的工作中遇到。我们用大量工程和科学学科中的例子来贯穿这些章节。对于这些例子的求解方法，则主要是通过使用前述的 5 步过程和 ANSI C（关于 ANSI C++ 会在第 8 章介绍）来实现。其中 ANSI C 是由美国国家标准协会设计的标准。

第 4 版中的变化

- 新版的主题是犯罪现场调查（CSI）。学习犯罪现场调查背后的技术不仅非常有趣，而且还为本书提供了一些很好的编程问题。在本书中，我们将用 C 语言程序方法解决这些问题。
- 修改了 1.2 节，加入了对诸如云计算和内核等当前热点话题的讨论。
- 增加了彩色插图来定义犯罪现场调查的重要领域——生物特征识别。生物特征识别是指通过物理特征或行为特征来进行身份识别。插图中讨论了指纹、人脸识别、虹膜识别、DNA 以及语音识别的技术。
- 每章都以犯罪现场调查技术的一张配图和相关讨论开始。除第 1 章以外，后面的每一章里都增加了相关的章以讲解技术的应用。除了讲解 C 语言的主要功能以外，还将介绍法医人类学、人脸识别与监控视频、虹膜识别、语音分析和语音识别、DNA 分析、指纹识别以及手势识别等相关技术。在这些应用部分，我们会设计对应的 C 程序来解决犯罪现场调查中遇到的技术问题。
- 基于每个工程应用提出的问题，后面都增加了“修改”练习题以对原问题进行扩展。
- 根据最新的 C++ 标准，更新了第 8 章中关于 C++ 的材料。

预备知识

本书假设读者先前没有任何计算机编程经验。对于数学的预备知识是高等代数和三角函

数。当然，如果读者使用过其他的计算机语言和软件工具，则可以跳过开头的预备材料以便更快地阅读内容。

课程结构

本书可以作为理工类相关专业本科生一个学期的课程教材。其中涉及的基本内容包括数学计算、字符数据、控制结构、函数、数组、指针和结构体等。学习过其他计算机编程语言的学生应该可以在一学期内完成这些内容。如果是在短学时课程中对C语言进行初步学习，可以仅学习书中的必修章节（可选章节在目录中已用“*”标出）。下面介绍使用本书的三种方式及对应的推荐章节：

- C语言基础。许多基础入门类课程除了向学生介绍编程语言外，还会介绍一些计算机工具。对于这些课程，建议涵盖必修部分的第1～5章。这些内容向学生介绍了C语言的基本功能，通过一定程度的学习之后，学生能够使用数学计算、字符数据、控制结构、函数和数组编写大规模的程序。
- 使用C语言解决实际问题。如果要通过一学期的课程教会学生掌握C语言，那么建议讲授第1～7章的全部必修章节。这些章节囊括了C语言的所有基本概念，包括数学计算、字符数据、控制结构、函数、数组、指针和结构体。
- 使用C语言和数值分析方法求解工程问题。书中许多章节都包含了常见的数值分析方法，比如线性插值、线性模型、求多项式的根、解联立方程组等。这些都为需要使用数值分析来完成课程作业的学生提供了强有力的工具。为了达到这样的课程目的，需要学习第1～7章的所有内容。

许多学生在读到有关C++中面向对象特性的附加内容时可能会很感兴趣。这里还是建议首先将第1～7章的所有必修内容学习完毕，最后再来了解第8章的内容。

解决问题的方法论

对于工程和科学问题的求解是本书不可或缺的重要部分。第1章介绍了利用计算机解决工程问题的5步处理过程。这5步处理过程是本书作者在她学术生涯早期提出的，并且由她班里或使用本书的数以千计的学生成功使用。不仅如此，这个成功的问题求解过程同时也被很多其他作者采纳。这5步分别为：

1）清楚地描述问题。

2）描述输入/输出信息。

3）手动计算一个简单例子。

4）设计算法并将它转换为计算机程序。

5）使用多种数据测试解决方案。

为了不断强化求解问题的能力，每次解决工程问题的过程中，都要清晰地标识出这5步中的每一步。除了经典的5步法之外，书中还使用了分解提纲、伪代码和流程图来完成自顶向下的程序设计并且将算法逐步求精。

工程和科学应用

本书的重点是将现实生活中的工程与科学的实例和问题相结合。其中涉及的工程应用包罗万象、种类繁多，下面是书中给出的例子：

- 海水盐度
- 速度计算
- 氨基酸分子量
- 风洞
- 波互作用
- 臭氧测量
- 探测火箭轨迹
- 缝合线封装
- 木材再生
- 关键路径分析
- 探空气球
- 冰山追踪
- 仪器可靠性
- 系统稳定性
- 元件可靠性
- 飞行模拟器的风速
- 飓风等级
- 分子量
- 语音信号分析
- 地形导航
- 电路分析
- 电厂数据
- 密码学
- 温度分布
- 厄尔尼诺－南方涛动现象
- 地震监测
- 海啸分析
- 地表风向

此外，每章开头都是以某方面的主题讨论开始，后续内容里，都会解决一些与犯罪现场调查技术相关的问题。这些问题涉及以下应用：

- 法医人类学
- 人脸识别与视频监控
- 虹膜识别
- 语音分析
- DNA 分析
- 指纹识别
- 手部识别

ANSI C

书中的所有语句和程序都是根据美国国家标准协会制定的 C 语言标准编写的。通过使用 ANSI C，学生可以学习编写适用于不同计算机系统的可移植程序。

软件工程观点

工程师和科学家们一直都希望设计并实现用户友好和可复用的计算机解决方案，因此学习软件工程技术就显得至关重要。在程序设计中，我们重点强调代码的可读性和文档完整性，有关软件工程的其他问题在本书中也都有讨论，比如软件生命周期、移植性、维护、模块化、递归、抽象化、复用性、结构化程序设计、验证和确认。

4 种类型的练习题

学习任何新技术都需要不同难度等级的练习。本书设计了 4 种类型的练习题来提高学生解决问题的能力。第一类题型标注为练习，这些都是与某节内容相关的客观题。大多数小节最后都带有一组“练习”题目，帮助学生判断自己对该节内容的掌握程度，以确认是否为后续学习做好准备。

除了“练习”题目，本书还设计了标注为修改的题型以给学生提供动手实践练习，这些习题都与“解决应用问题”一节中开发的程序有关。在“解决应用问题”中，我们使用 5 步处理过程设计了一个完整的 C 程序，而“修改”题要求学生使用不同的数据运行程序，以检验他们对程序的设计原理以及工程变量间关系的理解是否正确。此外，还要求学生对程序进

行简单的修改，然后重新运行程序来测试这些修改。本书结尾给出了一些“修改”题的参考答案。

每章都以两组习题结尾。其中，简述题包括判断题、多选题、匹配题、语法题、填空题、内存快照题、程序输出题和程序分析题。本书结尾给出了全部简述题的完整答案。

每章（除第1章外）最后一类题型是编程题。这些编程题都是关于各种工程应用的新问题，题目从易到难。每道习题都要求学生开发一个完整的C程序或函数。本书结尾给出了一些编程题的参考答案，教师参考书中包含了编程题的完整答案。

学习和编程辅助

每章的小结部分都包含对编程风格和调试说明的总结，再加上关键术语列表和C语句总结，这些都使本书具有很高的学习和参考价值。在全书末尾的术语表中包含了完整的关键术语表及其含义。此外，书中还包括常见函数和优先级列表，以及大部分C语句的例子。

可选的数值方法

本书讨论了经常用于求解工程问题的常见数值方法，包括插值、线性模型（回归）、求根和求解联立方程组。书中还介绍了矩阵的概念，并使用大量的例子进行说明。所有这些主题都假定读者仅具有三角函数和高等代数的知识储备。

MATLAB和可视化

具有创造力的工程师往往需要拥有分析问题的直观能力，而将与问题和求解方法相关的信息进行可视化处理是理解问题和提高直观能力的重要部分。因此，本书包含了大量的数据分布图来说明解决指定问题所需的信息之间的关系。所有图像都由MATLAB生成（MATLAB是一个可以进行数值计算、数据分析和可视化的功能强大的工具软件）。附录中也介绍了如何由存储在文本文件中的数据生成简单的分布图像，其中文本文件由文字处理软件或C程序生成。

附录

为了进一步方便读者参考，附录包含了许多重要内容。附录A对ANSI C标准库进行了详细讨论。附录B给出了ASCII字符编码表。附录C介绍了如何使用MATLAB绘制文本文件中的数据点。这使得学生可以用C程序生成ASCII文件，并使用MATLAB绘制文件中的数据分布图像。

非技术技能

除了在工程项目中学到的技能，21世纪的工程师还需要具备更多的能力。第1章对工程师需要具备的非技术技能进行了简短介绍，其中特别讨论了以下内容：提高口语和书面交流能力，了解将想法变为产品的设计/加工/制造过程，在跨学科团队中工作，了解全球化市场，综合与分析的重要性以及在解决工程问题时伦理和其他社会问题的重要性。本书讲授的重点是利用C语言来解决工程问题，与此同时，还尝试将这些非技术内容穿插在书中，结合具体问题一并介绍和讨论。

其他资源[1]

所有老师和学生都可以访问 www.pearsonhighered.com/etter。在这里，学生可以访问书中的学生数据文件，老师可以注册教师资源中心（Instructor's Resource Center，IRC）。IRC 中包含了本书中出现的所有编程项目的完整答案，以及一套完整的 PPT 讲稿。

致谢

许多人都为本书做出过巨大贡献。学生是对教学内容"好"与"不好"的最佳评判者。非常感激那些在写这本书之前从未使用过计算机的学生们，以及学习过其他语言的本科生和希望使用 C 语言进行科研分析的研究生的反馈。这些学生的评价和建议对本书的改进提供了很大帮助。

尤其重要的是，本书收到了很多相当具有建设性的评审意见。许多评审员都对本书做出了宝贵指导，包括：Murali Narayanan（堪萨斯州立大学），Kyle Squires（亚利桑那州立大学），Amelia Regan（加州大学欧文分校），Hyeong-Ah Choi（乔治华盛顿大学），George Friedman（伊利诺伊大学香槟分校），D. Dandapani（科罗拉多大学斯普林斯校区），Karl Mathias（奥尔大学），William Koffke（维拉诺瓦大学），Paul Heinemann（宾夕法尼亚州立大学），A. S. Hodel（奥本大学），Armando Barreto（佛罗里达国际大学），Arnold Robbins（佐治亚理工学院），Avelino Gonzalez（中佛罗里达大学），Thomas Walker（弗吉尼亚理工学院暨州立大学），Christopher Skelly（洞察资源公司），Betty Barr（休斯顿大学），John Cordero（南加利福尼亚大学），A. R. Marundarajan（加州州立理工大学波莫那校区），Lawrence Genalo（艾奥瓦州立大学），Karen Davis（辛辛那提大学），PetrosGheresus（通用汽车研究所），Leon Levine（加州大学洛杉矶分校），Harry Tyrer（密苏里大学哥伦比亚分校），Caleb Drake（伊利诺伊大学芝加哥分校），John Miller（密歇根大学迪尔伯恩分校），Elden Heiden（新墨西哥州立大学），Joe Hootman（北达科他大学），Nazeih Botros（南伊利诺伊大学），Mark C. Petzold（圣克劳德州立大学），Ali Saman Tosun（得克萨斯大学圣安东尼奥分校），Turgay Korkmaz（得克萨斯大学圣安东尼奥分校），Billie Goldstein（天普大学），Mark S. Hutchenreuther（加州州立理工大学），Frank Friedman（天普大学），Harold Mitchell Jr.（休斯顿大学）。

很高兴继续与培生教育出版集团的优秀团队一起出版这本书，他们是 Marcia Horton、Tracy（Dunkelberger）Johnson、Emma Snider、Kayla Smith-Tarbox 和 Eric Arima。我要感谢 Jeanine Ingber（新墨西哥大学）在第 2 版中作为合著者做出的贡献，她的许多贡献仍然体现在第 4 版中。

Delores M. Etter
南卫理公会大学电气工程系
得克萨斯，达拉斯

---

[1] 关于本书教辅资源，有需要的读者可向培生教育出版集团北京代表处申请，电话 010-57355160/5735 5171，电子邮件：service.cn@pearson.com。——编辑注

航空航天工程

风洞数据分析（第 2 章习题；第 5 章习题）
探空火箭（第 3 章习题）
飞行模拟器的风速控制（第 4 章习题）

生物医学工程

缝合线封装（第 3 章习题）

化学工程

温度转换（第 3 章习题）
分子量计算（5.3 节）
温度分布计算（第 5 章习题）

计算机工程

仿真方法（第 4 章习题）
密码学（第 5 章习题）
模式识别（第 6 章习题）

犯罪现场调查

法医人类学（2.5 节）
人脸识别（3.4 节）
虹膜识别（4.3 节）
语音识别（5.5 节）
DNA 序列（6.7 节）
指纹分析（7.3 节）
手部识别（8.5 节）

电气工程

电路分析（5.12 节）
噪声仿真（第 5 章习题）
电厂电力分布（第 5 章习题）

环境工程

臭氧测量（3.9 节）
木材再生（第 3 章习题）
探空气球（第 3 章习题）
地震监测（6.5 节）

基因工程

氨基酸分子量（第 2 章习题）

制造工程

关键路径分析（第 3 章习题）
仪器可靠性（4.6 节）
元件可靠性（第 4 章习题）

机械工程

先进的涡轮喷气发动机（2.10 节）
系统稳定性（4.8 节）
地形导航（5.9 节）

海洋工程

海水的冰点（2.7 节）
波互作用（3.6 节）
冰山追踪（4.4 节）
飓风等级（5.2 节；第 7 章习题）
厄尔尼诺－南方涛动现象（6.3 节）
海啸分析（7.5 节；第 7 章习题）
地表风向（8.6 节）

出版者的话
译者序
前言
工程应用项目

第 1 章 工程问题求解 …… 1
犯罪现场调查 …… 1
1.1 21 世纪的工程学 …… 1
1.1.1 现代工程学取得的成就 …… 1
1.1.2 不断变化的工程环境 …… 5
1.2 计算机系统：硬件与软件 …… 6
1.2.1 计算机硬件 …… 7
1.2.2 计算机软件 …… 7
1.3 工程问题求解方法论 …… 11
本章小结 …… 13
习题 …… 14

第 2 章 简单的 C 程序 …… 18
犯罪现场调查：法医人类学 …… 18
2.1 程序结构 …… 18
2.2 常量和变量 …… 21
2.2.1 科学计数法 …… 23
2.2.2 数值数据类型 …… 23
2.2.3 字符型数据 …… 24
2.2.4 符号常量 …… 26
2.3 赋值语句 …… 26
2.3.1 算术运算符 …… 28
2.3.2 运算符优先级 …… 29
2.3.3 上溢和下溢 …… 31
2.3.4 自增运算符和自减运算符 …… 32
2.3.5 缩写赋值运算符 …… 32
2.4 标准输入和输出 …… 33
2.4.1 输出函数 printf …… 34
2.4.2 输入函数 scanf …… 37
2.5 解决应用问题：根据骨骼长度估算身高 …… 38
2.6 数值方法：线性插值 …… 41
2.7 解决应用问题：海水的冰点 …… 44
2.8 数学函数 …… 47
2.8.1 基本数学函数 …… 47
2.8.2 三角函数 …… 48
*2.8.3 双曲函数 …… 49
2.9 字符函数 …… 50
2.9.1 字符输入/输出 …… 50
2.9.2 字符比较 …… 51
2.10 解决应用问题：速度计算 …… 52
2.11 系统边界 …… 55
本章小结 …… 56
习题 …… 58

第 3 章 控制结构和数据文件 …… 63
犯罪现场调查：人脸识别与监控视频 …… 63
3.1 算法开发 …… 63
3.1.1 自顶向下设计 …… 64
3.1.2 结构化编程 …… 65
3.1.3 多种解决方案评估 …… 67
3.1.4 条件错误 …… 67
3.1.5 测试数据的生成 …… 67
3.2 条件表达式 …… 68
3.2.1 关系运算符 …… 68
3.2.2 逻辑运算符 …… 69
3.2.3 优先级和结合性 …… 70
3.3 选择语句 …… 71
3.3.1 简单 if 语句 …… 71
3.3.2 if/else 语句 …… 72
3.3.3 switch 语句 …… 74
3.4 解决应用问题：人脸识别 …… 76
3.5 循环结构 …… 79
3.5.1 while 循环 …… 79
3.5.2 do/while 循环 …… 80

3.5.3 for 循环 …… 81
3.5.4 break 语句和 continue 语句 …… 84
3.6 解决应用问题：波互作用 …… 85
3.7 数据文件 …… 91
3.7.1 输入/输出语句 …… 92
3.7.2 读取数据文件 …… 94
3.7.3 生成数据文件 …… 100
*3.8 数值方法：线性建模 …… 102
*3.9 解决应用问题：臭氧测量 …… 105
本章小结 …… 108
习题 …… 111

第 4 章 用函数实现模块化程序设计 …… 117

犯罪现场调查：虹膜识别 …… 117
4.1 模块化 …… 117
4.2 自定义函数 …… 119
4.2.1 函数示例 …… 120
4.2.2 函数定义 …… 122
4.2.3 函数原型 …… 124
4.2.4 参数列表 …… 125
4.2.5 存储类型和作用域 …… 127
4.3 解决应用问题：计算虹膜边界 …… 128
4.4 解决应用问题：冰山追踪 …… 133
4.5 随机数 …… 137
4.5.1 整数序列 …… 138
4.5.2 浮点数序列 …… 141
4.6 解决应用问题：仪器可靠性 …… 142
*4.7 数值方法：求多项式的根 …… 147
4.7.1 多项式的根 …… 147
4.7.2 增量搜索技术 …… 149
*4.8 解决应用问题：系统稳定性 …… 150
*4.9 宏 …… 155
*4.10 递归 …… 158
4.10.1 阶乘运算 …… 159
4.10.2 斐波那契数列 …… 160
本章小结 …… 162
习题 …… 163

第 5 章 数组和矩阵 …… 169

犯罪现场调查：语音分析和语音识别 …… 169
5.1 一维数组 …… 170
5.1.1 定义和初始化 …… 170
5.1.2 计算和输出 …… 172
5.1.3 函数参数 …… 174
5.2 解决应用问题：飓风等级 …… 176
5.3 解决应用问题：分子量 …… 180
5.4 统计测量 …… 184
5.4.1 简单统计分析 …… 184
5.4.2 方差和标准差 …… 186
5.4.3 自定义头文件 …… 188
5.5 解决应用问题：语音信号分析 …… 188
5.6 排序算法 …… 193
5.7 搜索算法 …… 195
5.7.1 无序数列 …… 196
5.7.2 有序数列 …… 196
5.8 二维数组 …… 199
5.8.1 定义和初始化 …… 199
5.8.2 计算和输出 …… 201
5.8.3 函数参数 …… 203
5.9 解决应用问题：地形导航 …… 205
*5.10 矩阵和向量 …… 208
5.10.1 点积 …… 208
5.10.2 行列式 …… 209
5.10.3 转置 …… 210
5.10.4 矩阵加减法 …… 210
5.10.5 矩阵乘法 …… 211
*5.11 数值方法：联立方程组求解 …… 212
5.11.1 图像圆滑 …… 213
5.11.2 高斯消元法 …… 215
*5.12 解决应用问题：电路分析 …… 217
*5.13 多维数组 …… 221
本章小结 …… 222
习题 …… 224

第 6 章 指针编程 …… 230

犯罪现场调查：DNA 分析 …… 230
6.1 地址和指针 …… 230
6.1.1 地址运算符 …… 231
6.1.2 指针赋值 …… 232
6.1.3 地址运算 …… 235

6.2 指向数组元素的指针 …… 237
6.2.1 一维数组 …… 238
6.2.2 二维数组 …… 239
6.3 解决应用问题：厄尔尼诺－南方涛动现象 …… 241
6.4 函数调用中的指针 …… 243
6.5 解决应用问题：地震监测 …… 246
6.6 字符串 …… 250
6.6.1 字符串定义与输入/输出 …… 251
6.6.2 字符串函数 …… 251
6.7 解决应用问题：DNA 测序 …… 254
*6.8 动态内存分配 …… 256
*6.9 快速排序算法 …… 259
本章小结 …… 262
习题 …… 263

第 7 章 利用结构体编程 …… 267

犯罪现场调查：指纹识别 …… 267
7.1 结构体 …… 267
7.1.1 定义和初始化 …… 268
7.1.2 输入和输出 …… 269
7.1.3 结构体的运算 …… 270
7.2 使用结构体的函数 …… 271
7.2.1 结构体作为函数参数 …… 272
7.2.2 返回结构体的函数 …… 273
7.3 解决应用问题：指纹分析 …… 273
7.4 结构数组 …… 277
7.5 解决应用问题：海啸分析 …… 278
*7.6 动态数据结构 …… 281
本章小结 …… 291
习题 …… 293

第 8 章 C++ 编程语言简介 …… 297

犯罪现场调查：手部识别 …… 297
8.1 面向对象编程 …… 297
8.2 C++ 程序结构 …… 298
8.3 输入和输出 …… 299
8.3.1 cout 对象 …… 299
8.3.2 流函数 …… 300
8.3.3 cin 对象 …… 301
8.3.4 定义文件流 …… 302
8.4 C++ 编程范例 …… 302
8.4.1 简单计算 …… 303
8.4.2 循环 …… 303
8.4.3 函数、一维数组和数据文件 …… 304
8.5 解决应用问题：手部识别 …… 305
8.6 解决应用问题：地表风向 …… 307
8.7 类 …… 310
8.7.1 定义类数据类型 …… 310
8.7.2 构造函数 …… 313
8.7.3 类运算符 …… 314
8.8 数值方法：复根 …… 314
8.8.1 复数类定义 …… 315
8.8.2 二次方程的复根 …… 318
本章小结 …… 320
习题 …… 321

附录 A ANSI C 语言标准库 …… 323

附录 B ASCII 字符编码表 …… 335

附录 C 使用 MATLAB 绘制文本文件中的数据点 …… 339

“练习”的完整答案 …… 342

“修改”的部分答案 …… 353

章末简述题的完整答案 …… 355

章末编程题的部分答案 …… 359

术语表 …… 362

索引 …… 367

工程问题求解

犯罪现场调查

想必大家都在电影、书籍和电视节目等场景中见到过犯罪现场调查，但是你可能不了解它们背后用到的科技。这些技术很有趣，而且可以是C语言学习过程中的一个很好的主题。从第2章开始，我们将会在每一章介绍犯罪现场调查的一个领域，并详细介绍该领域使用的相关技术。我们会从该领域中选取一个典型问题进行讨论，之后再使用C语言程序解决这个问题。书中彩页里呈现了关于犯罪现场调查的基础信息，介绍了生物特征识别的定义并给出了一些身份识别技术的实例。

学习目标

本章将会介绍：

- 近年来引人注目的工程成果。
- 工程环境的变化和适应环境所需的非技术性技能。
- 计算机系统，包括软件和硬件。
- 贯穿本书的解决问题的5步法。

1.1 21世纪的工程学

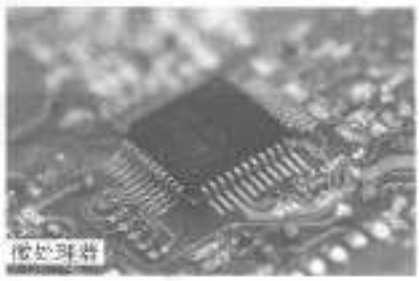
工程师是运用计算机科学、数学、物理、生物和化学等多个学科中的科学定律来解决现实世界的问题的。也正因为学科科目的多样性和现实问题的挑战性，工程学才能如此吸引人不断深入探索。在本节中，我们将会介绍近几年来一些引人注目的工程成就，还有一些作为21世纪的工程师需要具备的非技术性的技能。

1.1.1 现代工程学取得的成就

自从20世纪50年代后期计算机被发明以来，大量的工程成果如雨后春笋般出现。1989年，美国国家工程院（National Academy of Engineering，NAE）评选出了在此前25年间最为重要的10大工程成就。这些成果充分体现了工程学跨学科的特性，展示了科技是如何改变生活的，同时也展示了工程学广阔的发展前景。工程学能够给我们提供多样化的，充满吸引力和挑战性的职业生涯。后文中，我们将简单分析这10个成果。

微处理器（microprocessor）的发明是此前25年间最顶尖的工程成果。微处理技术使得

传统计算机可以缩小到邮票大小。这样的微处理器功能强大且造价不高。因此被用在电子设备、家用电器、玩具和游戏机中，在汽车、飞行器和航天飞机等高新技术领域中也很常见。微处理器同时也促进了计算器和智能手机等便携式计算设备的发展。


微处理器

这10大工程成就中有好几个都与太空探索相关。其中，登月计划（moon landing）可能是目前已知的最复杂的工程。在阿波罗宇宙飞船、登月飞行器和土星五号三级运载火箭的设计中，有很多需要突破的关键问题。宇航服的设计同样是一个非常复杂的工程项目，产生了一个包括三件套和双肩包在内的重约190磅[⊖]的系统。计算机在这一项目的完成中起到了非常重要的作用，包括各类系统的设计和登月独立飞行时的数据通信。整个登月过程涉及发射控制中心的450多名工作人员和7000多名其他相关人员的协同工作。这些人员分布在9艘船只、54个飞行器和遍布全球的观测站。

登月计划

太空计划也推动了应用卫星（application satellite）的发展，这些卫星可用于提供天气信息、通信信号中继、绘制未知地形、更新环境信息、监测大气组成成分等。全球定位系统（Global Positioning System，GPS）是一个由24颗卫星组成的卫星群。这些卫星可以实时地向全世界广播自己的位置、速度和时间信息。GPS接收器能够接收广播信息并计算信号从GPS卫星传送到接收器的时间。通过4个卫星传送的信息，接收器中的微处理器就可以精确计算出自身的位置信息。其位置精确度随计算方法不同而变化，可以精确到米级甚至是厘米级。

卫星

⊖ 1磅=0.453 592 4千克

计算机辅助设计与制造（Computer Aided Design and Manufacturing，CAD/CAM）也是一个显著的工程成果。CAD/CAM 大大提高了各种制造过程的速度和效率，因此促生了新一轮的工业革命。CAD 使得用计算机完成设计成为现实，它可以生成电路图、零部件清单和计算机模拟结果。CAM 依据设计结果来控制机械或工业机器人完成制造、装配和移动的工作。

计算机辅助设计

大型喷气式客机（jumbo jet）起源于 1969 年开始投入使用的美国空军 C-5A 型运输机。喷气式客机的成功很大程度上要归功于高旁路涡扇发动机，它使得飞机使用更少的燃料而飞得更远，并且噪音比之前的喷气式引擎更小。其引擎的核心工作原理与涡轮喷气式飞机很相似，都是压缩机叶片输动空气使其进入引擎的燃烧仓内，受热膨胀的气体推动引擎向前行驶，同时使涡轮机旋转带动压缩机和引擎前面的大扇叶高速转动。旋转的扇叶产生了大量的引擎推进力，从而推动飞机前进。

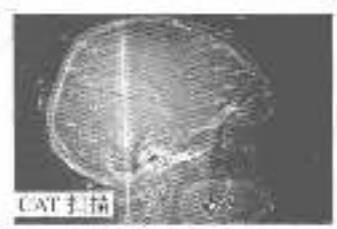
大型喷气式客机

飞行器工业也是第一个发展和应用高级复合材料（advanced composite material）的工业。高级复合材料能够将多种材料的特性组合在一起，其组合的方式是利用一种材料强化另一种材料的纤维结构。飞行器和宇宙飞船上使用的高级复合材料具有重量更轻、承重更大和耐热性更强的特点。如今的复合材料市场也在户外运动商品领域蔓延开来，举例来说，使用凯芙拉纤维编织层来提高速降滑雪板的强度并降低重量，用石墨和环氧树

造来制造高尔夫球杆，比钢铁材质的传统球杆性能更佳。复合材料也被用在人造假肢的制造上。

高级复合材料

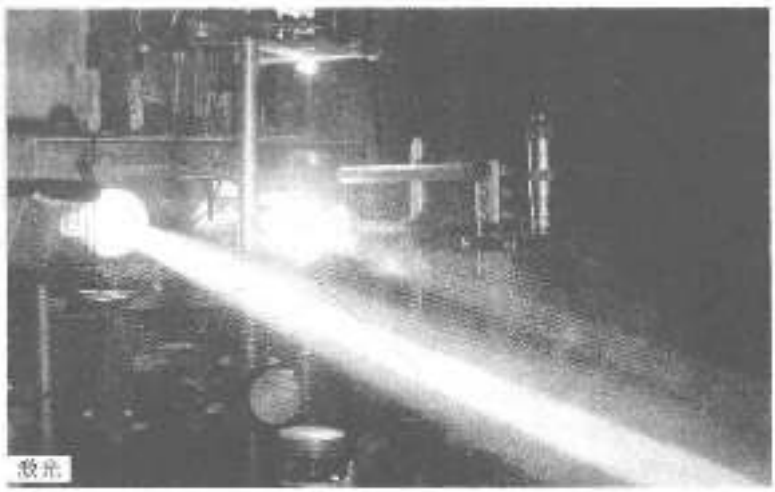
计算机轴断层摄影（Computerized Axial Tomography，CAT）扫描仪是结合了医药、生物工程和计算机科学三门学科技术所取得的发明成果。这种仪器能够生成特定物体的三维图像，或是不同角度所得的二维X光切片。每一束X光线计算出这个角度下物体的密度，然后再由复杂的计算机算法将各个角度的信息结合起来，最终得到该物体内部结构的清晰图像。CAT扫描现已广泛应用于肿瘤、血栓和大脑异常的诊断。


CAT扫描

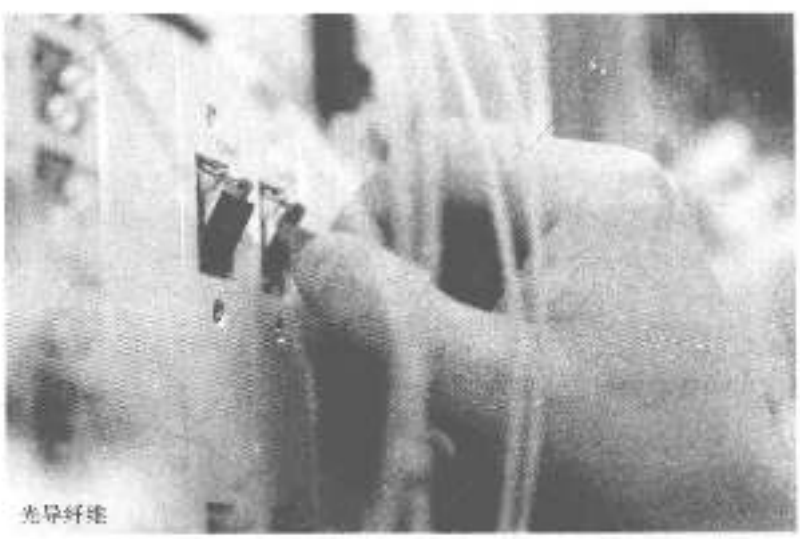
基因工程（genetic engineering）结合了基因学和工程学的工作，制造出了很多新产品，从胰岛素到生长素和抗病性蔬菜植物。基因工程的方法是切割一个有机体中有价值的基因物质，将其转移到另一个有机体中，并随着有机体的生长不断复制繁衍，从而在新的有机体中得到转移过来的基因。首例商业化基因工程产品是人工胰岛素，也就是目前广泛应用于临床治疗的优泌林重组人胰岛素注射液。现在还有学者在研究基因变异的微生物在降解有毒污染物和农药方面的应用。

基因工程

激光（laser）是有相同频率、可以被聚焦在很窄的光束中且定向传播的光波。二氧化碳激光被用来在各种材料上钻孔，包括陶瓷材料、复合材料等。激光也被用在医疗中，比如修复分离的视网膜、修补血管漏洞、消除脑肿瘤和实施细微的耳内手术等。三维图片又称为全息摄像，也是激光促生的。


激光

光导通信中大量使用光导纤维（optical fiber）。光导纤维是透明的丝状玻璃，比人的头发还要纤细。这种光导纤维与无线电波或是电话线里的电波相比能够传送更多的信息，并且不会产生造成干扰通信的电磁波。海底光导纤维电缆使得各大洲建立了通信通道。光导纤维也被用在医疗中，外科医生可以将其置入患者体内进行检查或是激光手术。 [8]


光导纤维

1.1.2 不断变化的工程环境

21世纪的工程师处在一个需要各式各样的非技术性技巧和能力的大环境之中。计算机将会成为工程人员最主要的计算工具，同时它对非技术性能力的提升也发挥着很重要的作用。

工程师需要具备很强的沟通技巧（communication skill），包括口语表达和书面表达。计算机提供了很多用于帮助编写摘要和准备书面材料的软件，例如图表绘制软件，我们可以利用这些软件制作技术性报告和演示材料。在本章的最后列出了一些书面和口语表达的问题，以供读者练习这些重要技巧。

设计/处理/制造过程（design/process/manufacture path）包括了将一个想法从概念变成产品的整个过程，是工程师必须首先理解的。这个过程的每一步都会用到计算机，包括设计分析、机器控制、机械化装配、质量检查和市场分析。本书中有几个问题与这些主题相关，例如，在第4章中，我们将会创建程序来模拟使用多种部件组成的系统的可靠性。

未来的工程团队将会像现在的工程团队一样，都是跨学科团队（interdisciplinary team）。之前我们谈到的25年间10大工程成就已经突出表现了其跨学科的特性。学会团队内部交流并建立高效沟通的组织结构对于工程师来说十分重要。组织团队进行学习考核是提高工程团队技能的好方法。给团队中的每个成员布置特定的主题，这样某个成员就可以针对他们的主题以示例和测试的形式为整个团队来温习相关内容。

9

工程师还需要了解全球市场（world marketplace），包括理解不同的文化、政治系统和商业环境。有关这些主题的课程和外语课程将会帮助你加深理解，但国际交流项目可以帮你了解更广阔的世界，给你更宝贵的知识。

工程师是解决问题的人，但是问题往往都表达得不是十分明确。因此工程师必须能够从问题讨论中提取问题的描述，然后确定重要事项，不仅包括建立秩序，还要学习在混乱中找出各种关联。这就意味着不仅要分析（analyzing）数据，更要通过很多碎片信息来整合（synthesizing）出一个解决方案。信息的集成和问题的分解同样重要。问题的解决方案除了包括对问题的抽象，还包括在问题产生的环境中进行实践性学习。

问题的解决方案必须放在特定的社会环境（social context）下考虑，有时需要根据环境问题修改解决方案。工程师在公布试验结果、质量认证和设计局限性方面要有伦理道德意识并谨慎考虑。伦理问题非常难以解决，而现如今的很多新兴科技成果正在引发很多伦理问题，例如人类基因图谱的绘制会带来很多潜在的伦理、合法性和社会舆论问题。基因疗法可以帮助医生治疗糖尿病，能否同样用来提高运动员的体能呢？是否能将未出生孩子的身心健康状况告诉他的父母呢？一个人的基因代码是否该被当作隐私进行保护呢？类似这样的很多复杂分歧都会出现。科技进步将是一把双刃剑，一个新技术既可以造福人类也会带来很多麻烦和坏处。

本文给出的材料仅仅是工程师建立知识、自信和理解力的第一步。下面两节中，我们将分别介绍目前工程问题中使用的计算机系统和解决问题的方法论，其中方法论部分将贯穿全书，在后续的每一章里用C语言解决工程问题时都会用到。

1.2 计算机系统：硬件与软件

在我们开始介绍C语言之前，需要简单介绍一下计算机的工作过程，这有助于那些不经常使用计算机的读者了解其原理。计算机（computer）是用来执行程序的机器，程序（program）是为了实现特定功能而设计的一系列指令操作。计算机硬件（hardware）指的是计算机的设备，比如笔记本电脑、U盘、鼠标、键盘、平面显示器和打印机。计算机软件（software）指的是各式各样的程序。程序描述了我们希望计算机执行的操作和步骤，程序可以是我们自己编写的，也可以是下载或是购买的，比如计算机游戏。计算机的软件和硬件都可以组成一个自包的个体，类似于笔记本电脑的软硬件能够集成为独立的产品；计算机也可以通过网络（network）或是互联网（Internet）与其他软件和硬件连接后协同工作。现在广泛使用的云计算（cloud computing）就是通过网络连接到远端的硬件、软件和大数据集的服务模式。

1.2.1 计算机硬件

所有计算机的内部结构基本上都是一致的。如图 1-1 所示，处理器（processor）部件控制着其他所有的部件共同工作，它接受输入的数据（来自键盘或是数据文件）并且将其存入存储器，它还负责理解和执行程序中的指令。例如对两个数值的相加操作，处理器会从存储器中取出数值，然后将其发送至算术逻辑单元（Arithmetic Logic Unit，ALU）。算术逻辑单元执行加法操作，之后处理器将结果存入存储器（memory）中。处理单元和算术逻辑单元使用的是内部存储器，包括只读存储器（Read-Only Memory，ROM）和随机存取存储器（Random Access Memory，RAM），数据也可以存储在外部存储设备中，比如移动硬盘或是 U 盘。处理器和算术逻辑单元组合的整体叫作中央处理单元（Central Processing Unit，CPU）。微处理器（microprocessor）是将一个完整 CPU 集成在一个邮票大小的集成电路芯片中，其中包含了上百万个电子元器件组件。

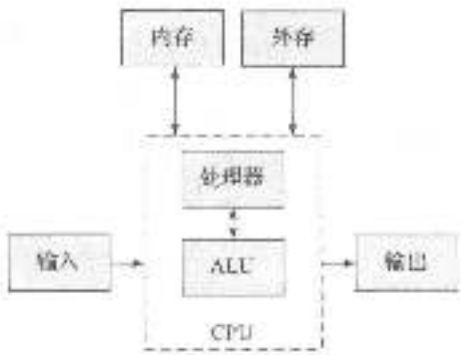


图 1-1

10

如今许多廉价的打印机都采用喷墨技术，可以轻易地打印出文件的黑白副本或彩色副本。我们也可以将信息存储在各种各样的数码存储设备中，如 CD 和 DVD。打印的副本被称作硬拷贝（hard copy），数码副本叫作电子拷贝（electronic copy）或软拷贝（soft copy）。如今的很多打印机还有很多其他的功能，比如复印、传真和扫描等。

各种计算机设备的大小、形状和形式各不相同。事实上，如今的大部分手机也可以被称为计算机。这些手机都含有 CPU，能够存储可执行程序。除此之外，智能手机（smartphone）还含有图像处理器和大量 RAM，并向多核（多处理器）、低能耗 CPU 的方向发展。现在很多家庭都有个人计算机，用来处理电子邮件、花销管理和游戏等，丰富了人们的生活。台式机是配有分立式的显示屏和键盘的计算机，而笔记本电脑将所有硬件都集合并尽力压缩体积和重量，是一种便携的计算机。对于一些人来说，平板电脑（如 iPad）和智能手机甚至可以取代台式机和笔记本电脑。

1.2.2 计算机软件

计算机软件中包含着我们想让计算机执行的指令或命令。计算机软件可以分为以下几类：操作系统、软件工具、桌面应用程序和编译器。图 1-2 展示了这几类软件和计算机硬件之间的关系。下面我们讨论每一类软件的细节。

11

1. 操作系统

有一些软件（如操作系统）是在购买计算机硬件的时候就自带的。操作系统（operating system）为用户提供了一个便捷高效的使用硬件的接口，让用户可以选择和运行系统上的应用软件。操作系统中提供硬件和软件应用程序之间接口的部分又称为内核（kernel）。常见的桌面操作系统有 Windows、Mac OS、UNIX 和 Linux，智能手机操作系统有 Android（Linux 的一个变种）和 iOS（UNIX 的一个变种）。

操作系统中往往还带有一些服务程序（utility），通过服务程序你可以实现一些基础的功

能操作，比如打印文件、将文件从一个文件夹复制到另一个文件夹、列出文件夹内容等。大多数操作系统通过图标和菜单等形式简化这些服务程序的使用模式。

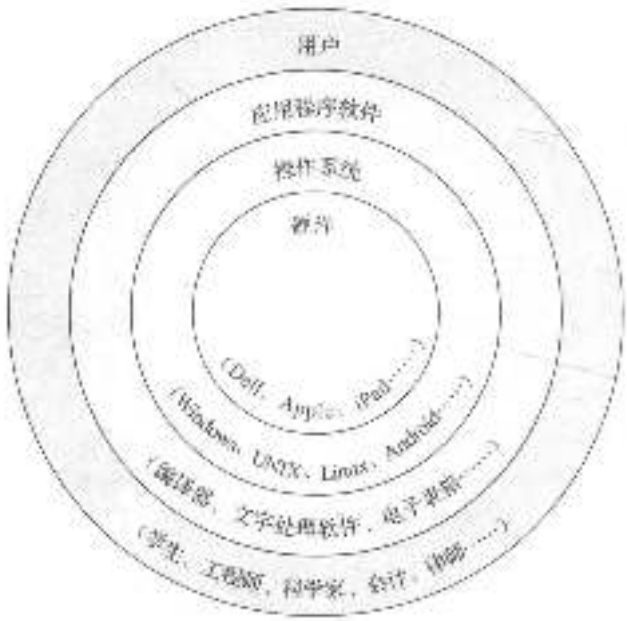


图 1-2

2. 软件工具

软件工具是用来执行一些常见操作的程序，例如，微软的 Word 文字处理器（word processor）软件是用于输入文字和设置版式的程序，你可以用它来存储互联网上下载的文档信息，也可以编辑数学公式等，还可以检查文本的语法和拼写错误。大多数的文字处理器软件都可以创建复杂格式的文档，包含表格、图片，也可以支持双栏排版。这些功能可以让用户在自己家里通过一台笔记本电脑实现桌面印刷的工作。

12 像微软的 Excel 这样的电子表格软件将数据放在表格里显示，从而可以减轻处理数据时的工作量。电子表格软件最初用于财经学和会计学中，后来开始大量用于解决科学和工程问题。大部分的电子表格软件都有绘图功能，可以将数据转换成为图表来分析和显示信息，大大提高了分析的效率。数据库管理工具（database management tool）可以用来在大数据集合中分析和获取信息，这个过程又被称为数据挖掘。

另一种重要的工具软件是数学计算工具（mathematical computation tool），这类工具包括 MATLAB、Mathematica 和 Maple。这些工具中有非常强大的数学计算命令，并且在绘制图表方面有丰富的功能。这种将计算和可视化相结合的工具对于工程师来说十分有用。

对于利用软件工具可以解决的工程问题，直接使用这些软件工具会比用计算机语言开发程序要更高效。如今软件工具和计算机语言之间的界限越来越模糊了，一些强大的软件工具不但有自己的编程语言，还有一些特别有用的扩展功能。（例如，这里将 MATLAB 作为一个软件工具，而很多人认为 MATLAB 是一种编程语言。）

3. 计算机语言

计算机语言可以分为 5 代，并且是逐代演进的。第一代计算机语言是机器语言。机器语言（machine language）与计算机硬件紧密结合在一起，它常常是以 0 和 1 组成的二进制（binary）

中来表示（每个二进制位又叫作一个比特（bit））。因此，机器语言又叫作二进制语言。

汇编语言（assembly language）是第二代语言。它是针对某种特殊的计算机硬件而设计的，指令用符号表示，而不再是二进制。汇编语言中可用的语句很少，因此用汇编语言来写程序变得繁琐而枯燥。此外，使用汇编语言必须要了解特定硬件的相关信息。智能仪器类的系统通常要求其微处理器上的程序运行速度很快，这些程序称为实时程序（real-time program）。实时程序经常使用汇编语言来编写，为的是能够充分利用计算机硬件的特性来提高运行速度。

高级语言（high-level language）是第三代语言，使用的是像英语一样的指令。C、C++、C# 和 Java 都属于高级语言。用高级语言写程序显然比机器语言和汇编语言更简单，但是每一种高级程序语言都包含了大量的指令和使用这些指令的语法（syntax）规则。

C 语言是一种通用编程语言，它的前身是 BCPL 和 B 语言。这两种语言是在 20 世纪 60 年代晚期由贝尔实验室开发的。1972 年，贝尔实验室的 Dennis Ritchie 开发了第一个 C 语言编译器并投入使用。因为这种语言独立于硬件架构，因此广泛用于系统开发。由于 C 语言在工业界和学术界的广泛使用，使得对这种语言的标准定义显得尤为重要。1983 年，美国国家标准协会（American National Standards Institute，ANSI）成立了一个委员会，制定了独立于机器的明确的 C 语言标准。1989 年，ANSI C 的标准正式出台。本书中使用的正是这种标准化的 C 语言。

贝尔实验室的 Bjarne Stroustrup 在 20 世纪 80 年代早期开发了所以 C++ 语言，它扩充了 C 语言的内容，是 C 语言的超集。因为其面向对象的特性的重要性，所以 C++ 被当作一种全新的编程语言来对待。Java 是一个纯粹的面向对象的语言，其在编程中的易用性使得它在基于互联网的应用开发和网络通信设备开发中十分有效。

由于 C 语言强大的指令功能和数据结构，它依然是很多工程师和科学家的首选。C 语言可以用于操作系统的开发，并且从 C 语言过渡到 C++ 和 Java 也非常简单。因此本书的目标是通过使用大量实用的、现实世界的具体事例来涵盖这门语言的主要特征，从而打牢 C 语言的基础。

第四代语言与人类语言（或者自然语言）更加接近，通常是专为某个应用领域设计的，如数据库开发。第五代语言描述的不再是算法，而是约束，这类语言主要用于人工智能（Artificial Intelligence，AI）领域的研究。

4. 运行计算机程序

用高级程序语言（如 C 语言）编写的程序在被计算机运行之前必须要先翻译为机器语言，编译器（compiler）就是来执行这个翻译操作的特殊程序。因此，为了编写和运行 C 程序，必须要有 C 编译器。C 编译器一般是随着操作系统发布的，是针对特定操作系统的独立软件包。

如果编译器在编译过程中发现了错误（又叫作 bug），相应的错误信息会显示出来，必须调整程序的语句然后重新编译。在这个过程中发现的错误叫作编译错误（compiler error）或叫作编译时错误。例如，如果想要将存储在 sum 变量中的值除以 3，正确的表达式是 sum/3；如果错用了反斜线表示为 sum\3，就会出现编译错误。在程序能够无错误编译之前通常要经过多次编译、修正语句（debug），重编译的过程。当没有编译错误时，编译器会生成一个机器语言程序，该程序运行时执行的就是 C 语言编写的操作指令。编译前的 C 语言程序被称作是源程序（source program），而之后生成的机器语言版本叫作目标程序。源程序

和目标程序表述的是相同的操作，只是源程序是用高级程序语言编写的，目标程序（object program）是由机器语言组成的。

程序成功编译后，目标程序就可以生成。要想运行（execution）目标程序，还需要目标程序与其他关联的机器语言链接起来，并将程序存入存储器中以准备开始运行。目标程序开始运行后，可能会出现新的错误——运行错误，也叫作运行时错误或逻辑错误（logic error），它们统称为程序漏洞。运行错误常常会导致程序意外终止，例如程序语句中如果出现了除数为0的情况，就会引发运行错误。有些运行错误不中断程序执行，但是会造成计算结果出现错误。这些错误可能是由程序员在编写关键步骤时或是程序处理数据时的错误造成的。当程序语句出错而造成运行错误时，我们必须改正源程序中的错误，然后重新执行编译操作。即使一个程序顺利执行完成，我们也必须仔细检查结果以确保万无一失。计算机会准确地执行我们的指令，但如果指令存在错误，计算机便会执行错误（但是语法正确）的操作步骤并得出错误的答案。

14

图1-3中展示了编译、链接/加载和运行的过程。汇编语言编程开发的过程稍有不同，将汇编语言转换成机器语言的工作是由汇编器（assembler）完成的，对应的过程叫作汇编、链接/加载和运行。

图1-3 程序的编译/链接/运行

5. 软件生命周期

一套基于计算机的解决方案的成本可以通过硬件和软件的成本来估算，而这些成本的大部分都集中在软件部分，因此，就需要着重了解软件的开发过程。

软件项目的开发通常遵循确定的顺序或者循环规律，这个过程一般称为软件生命周期（software life cycle）。这个过程中的步骤主要包括项目定义、详细规范、编写代码、模块测试、集成测试和维护。（本书后面章节会详细介绍这些步骤。）软件维护在软件系统成本中占据了很大部分，它包括软件功能增强、软件使用时错误修正和新型软硬件的适配等。进行维护的难易程度取决于项目最初的定义和设计规范，因为这些步骤为项目其他部分奠定了基础。因此，在利用软件解决问题的过程中，需要特别强调的是在编写代码和测试之前，一定要仔细确定项目的定义和详细规范。

其中一种有效降低软件开发时间和成本的方式是软件原型（software prototype）的开发。不同于等到软件系统开发完成之后用户才可以使用的方式，软件原型的开发是在其生命周期的前期完成的，它并不具有软件最终所需的所有功能，但用户可以在生命周期的早期使用它来进行项目设计目标和细节的修正。尽量将改动控制在软件的生命周期的早期，有助于有效地控制项目的时间和成本。为了加快开发速度，常常在早期使用工具软件（如MATLAB）来开发软件原型，之后再用另一种程序语言来开发最终的系统。

作为工程师，我们常常会遇到为已有软件修改或增加额外功能的情况。无论是用软件工

具还是高级程序语言开发的已有软件，如果代码结构很清晰、可读性强，并且附加的文档清晰易读且是最新的，那么这些修改工作便会简单很多。因此，我们强调培养好的编程习惯的重要性，以确保编写的程序可读性更高，文档更清晰。 15

1.3 工程问题求解方法论

解决问题是工程学课堂上最重要的部分，也是很多其他课程的关键部分，比如计算机科学、数学、物理和化学等课程。因此，找到一种通用的解决问题的方法是十分重要的。如果该方法对于这些不同领域都很适用，那么就不必每接触一个新的领域都要再学习新的解决方法了。解决工程问题的过程同样也适用于其他领域，当然，前提是假设利用计算机来解决该领域的问题。

本书中我们将用到的问题求解过程或方法论有以下5步：

1）清晰地陈述问题。

2）描述输入和输出的信息。

3）用一小组简单的数据进行手工计算（或使用计算器）。

4）设计算法并转换成计算机程序。

5）用大量、多样的数据进行检验。

接下来利用计算平面上两点间距离的例子来介绍每一步骤的具体做法。

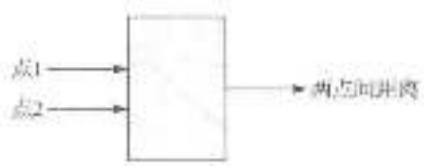
1. 问题陈述

解决问题的第一步就是将问题陈述清楚，一个清晰、简洁而不产生歧义的问题陈述至关重要。本例中，问题陈述是这样的：

计算出平面上两点间的直线距离。

2. 输入／输出描述

第二步就是确定已知的信息和需要计算出的目标结果值。已知信息代表了该问题的输入，而目标结果就是输出，这些被称为问题的输入和输出。对于很多问题来说，表示输入和输出的图示是非常有用的。在这个阶段，图中描述的都是抽象概念，因为此时并没有定义计算给出结果的方法步骤，而只是给出了用于计算输出结果值的相关输入信息。该例子中的I/O图示例如下：



16

3. 手动演算示例

解决问题的第三步就是借助计算器计算或手动演算一组简单的数据。这是很重要的一步，即使在解决一些简单问题时也不能忽视这一过程。在这一过程中，将得出解决方案的具体细节。如果无法获取一组数据并计算出结果，就表示还没有找到解决问题的方法，那就无法进行下一步的操作。这时应该再仔细分析一遍题目，或是查阅一些参考资料。在本例子中手动计算操作如下：

给点 p_1 和 p_2 赋如下坐标值：

$$p_1 = (1,5),\quad p_2 = (4,7)$$

我们要计算出两点间距离，也就是直角三角形的斜边长度，如图 1-4 所示，利用勾股定理可以计算出距离值：

$$\begin{aligned}\text{distance} &= \sqrt{(\text{side}_1)^2 + (\text{side}_2)^2}\\ &= \sqrt{(4-1)^2 + (7-5)^2}\\ &= \sqrt{13}\\ &= 3.61\end{aligned}$$

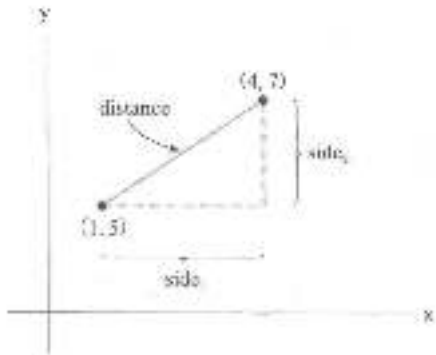


图 1-4

17

4. 算法设计

当你可以用一组简单数据计算出问题的结果时，就可以开始设计算法（algorithm）或者是列出一个逐步解决问题的流程了。对于像这例子一样的简单问题，流程中列出的就是需要逐个执行的操作。这种将方案逐条列出来的方式可以把复杂问题分解成若干小问题。在本例中计算两点问题的算法流程如下。

分解提纲

1）给两点赋值。

2）计算出两点确定的直角三角形的两直角边长度。

3）计算出两点间距，即直角三角形斜边的长度。

4）输出结果值。

将分解提纲（decomposition outline）转化成 C 语言指令，然后就可以用计算机来进行计算。在以下所示的解决方案中，你会发现代码中的指令与前面手动计算的过程十分类似。这些指令的详细介绍将会在第 2 章给出。

```
/*-----------------------------------------------------------*/
/*   程序 chapter1_1                                         */
/*                                                           */
/*   该程序计算两点间的距离                                  */

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  声明和初始化变量  */
   double x1=1, y1=5, x2=4, y2=7,
          side_1, side_2, distance;

   /*  计算直角三角形的边  */
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);

   /*  打印距离  */
   printf("The distance between the two points is "
          "%5.2f \n",distance);

   /*  退出程序  */
```

```
    return 0;
}
/*-----------------------------------------------------------------*/
```

18

5. 测试

解决问题的最后一步就是进行测试。因为前面已经运行过手动计算并知道计算结果，所以首先用在手动演算步骤中使用的数值来进行测试。运行该程序后，计算机会显示以下输出：

```
The distance between the two points is  3.61
```

该输出结果与手动计算出的结果完全一致。如果不一致，那么应该返回去检查代码，找到错误。此外，还应该用其他的数据来检验，从而确保该解决方案的正确性。

本例中展示的开发程序以解决问题的步骤在后续章节的解决应用问题环节中将会反复用到。

本章小结

本章展示了近年来杰出的工程成果，这些成果都印证了工程应用的多样性。同时，还讨论了成为一个优秀工程师需要具备的许多非专业性技能。由于大多数的工程问题都是通过计算机解决的，因此也简单介绍了计算机系统的基本组成部分，包括计算机硬件和软件。最后还介绍了解决问题的一个5步方法论，可以使用该方法论指导计算机程序开发以解决问题。该5步法包括：

1）清晰地陈述问题。

2）描述输入和输出的信息。

3）用一小组简单的数据进行手工计算（或使用计算器）。

4）设计算法并转换成计算机程序。

5）用大量、多样的数据进行检验。

这个过程在本书中会多次用到。下面，就要进入本书的主题——犯罪现场调查背后的科技手段。

关键术语

algorithm（算法）
ANSI C（美国国家标准协会C语言标准）
arithmetic logic unit（ALU，算术逻辑单元）
assembler（汇编器）
assembly language（汇编语言）
binary（二进制）
bit（比特）
bug（漏洞）
central processing unit（CPU，中央处理单元）
cloud computing（云计算）
compiler（编译器）
compiler error（编译错误）
linking/loading（链接/加载）
logic error（逻辑错误）
computer（计算机）
database management tool（数据库管理工具）
debug（修正错误）
debugger（调试器）
decomposition outline（分解提纲）
desktop publishing（桌面印刷）
electronic copy（电子副本）
execution（运行）
hardware（硬件）
high-level language（高级语言）
I/O diagram（输入/输出图示）
kernel（内核）
real-time program（实时程序）
software（软件）

19

machine language（机器语言）
memory（内存）
microprocessor（微处理器）
network（网络）
object program（目标程序）
operating system（操作系统）
personal computer（PC，个人计算机）
problem-solving process（问题求解过程）
processor（处理器）
program（程序）
software life cycle（软件生命周期）
software maintenance（软件维护）
software prototype（软件原型）
software tool（软件工具）
source program（源程序）
spreadsheet（电子表格）
syntax（语法）
utility（实用程序）
word processor（文字处理软件）

习题

简述题

判断题

判断下列语句的正（T）误（F）。

1. CPU是由ALU、内存和处理器组成的。 T F
2. 链接/加载步骤是为目标程序的运行做准备工作。 T F
3. 算法是分步骤地描述问题解决方案，而计算机程序只需一步就能解决问题。 T F
4. 计算机程序就是对算法的实现。 T F
5. 一个实用程序的作用是将C语言转化为二进制。 T F
6. 微处理器就是非常小的处理器。 T F
7. 内部存储器和外部存储器之间可通过ALU来实现数据交流。 T F
8. 电子表格能够以图表的形式操作对象。 T F
9. 计算机辅助设计只能用于设计微型计算机。 T F
10. 文字处理器可以实现文本的输入和编辑。 T F
11. 数学软件工具能够很好地处理计算和可视化问题。 T F
12. 如果要修改逻辑错误，只需要检查程序的执行步骤即可。 T F
13. 在编译步骤里会检查出程序的所有漏洞。 T F
14. 算法给出了对应问题的解决步骤。 T F
15. 计算机程序是用来解决问题的一串指令。 T F
16. 如果找到一组数据输入到程序中能够运行成功，则表明它已经完全通过了测试。 T F
17. 内核属于硬件的一部分。 T F
18. C语言是高级程序语言的一种。 T F
19. 在如今的软件系统开销中，软件维护不属于一项重要开销。 T F
20. 用机器语言写成的程序可以表示为二进制字符串。 T F

多选题

选出符合下列各题的最佳选项。

21. 指令和数据被存储在（　　）里。

(a) 算术逻辑单元（ALU） (b) 控制单元（处理器）

(c) 中央处理器（CPU）　　(d) 键盘

22. 操作系统是（　　）。

(a) 由用户设计的一种软件　　(b) 一个在用户和硬件之间便捷高效的交互接口

(c) 能够执行常用操作的一组实用程序　　(d) 一组软件工具

23. 源代码是（　　）。

(a) 编译处理后的结果

(b) 从处理器获取信息的进程

(c) 以一种计算机语言来解决指定问题的一组指令

(d) 存储在内存中的数据

(e) 从键盘输入的数值

24. 目标代码是（　　）。

(a) 将源代码经过编译处理后的结果　　(b) 从处理器获取信息的进程

(c) 一个计算机程序　　(d) 一个进程，包含着用来解决指定问题的命令列表

(e) 链接/加载运行后的结果

25. 所谓算法就是（　　）。

(a) 解决某一指定问题的一系列解决步骤　　(b) 计算机能够理解的指令集合

(c) 允许用户输入文本信息的代码　　(d) 逐步操作的解决步骤

(e) 为了解决问题而作出的一系列数学公式和推导

26. 硬拷贝是指（　　）。

(a) 存储在硬盘上的信息　　(b) 打印出来的纸质信息

(c) 显示在屏幕上的信息　　(d) 一种计算机程序

(e) 以上都是

21

27. 计算机硬件包括（　　）。

(a) 键盘　　(b) 打印机

(c) 磁盘驱动器　　(d) 终端显示屏

(e) 以上都是

28. 以下（　　）属于软件。

(a) 打印机　　(b) 显示屏

(c) 一段计算机代码　　(d) 以上都是

29. 高级语言（　　）。

(a) 有助于进行实时程序设计　　(b) 是第二代计算机语言

(c) 又称作自然语言　　(d) 具有类似英语的表达方式

30. 源程序和目标程序的区别在于（　　）。

(a) 源程序中可能会存在一些漏洞，但目标程序中绝不可能存在漏洞

(b) 源程序是原始代码，目标程序是修改后的代码

(c) 源程序是用高级语言写成，而目标程序是机器语言

(d) 目标程序其实也是一种源程序

(e) 源程序可以被执行，而目标程序不能被执行

31. 在开始解决问题时，首先应该（　　）。

(a) 设计算法　　(b) 编写程序

(c) 编译源程序　　(d) 链接以生成目标程序

32. 手动演算示例指的是（　　）。

(a) 通过手算来执行算法

(b) 用一组简单数据来将问题解决方案的全部细节执行一遍

(c) 列出问题的解决方案

(d) 将问题的解决方案继续扩展

(e) 用计算器将算法逐步验证

33. 计算机程序指的是（　　）。

(a) 输入和输出设备等组件的集合

(b) 用来解决问题的一组指令

(c) 由一种计算机可以理解的语言编写，并由计算机来执行的一组指令

(d) 通过一系列步骤来解决问题的解题过程

(e) 将问题分解为一系列简单步骤的分解提纲

选择匹配题

为下列各题中的定义选择出对应正确的术语。

算法	语法
ANSI C	硬件
算术逻辑单元（ALU）	输入设备
中央处理单元（CPU）	逻辑错误
云计算	机器语言
编译	内存
调试	微处理器
自然语言	软件维护
网络	电子表格
操作系统	（计算机语言的）语法
输出设备	系统软件
程序	文字处理器
软件生命周期	

22

34. _____ 一组告诉计算机应该执行何种操作的指令集合

35. _____ 计算机的一个机械部件

36. _____ 计算机的大脑

37. _____ 用来显示程序结果的设备

38. _____ 计算机运算所需要的编译器和其他程序

39. _____ 求解问题的操作步骤

40. _____ 将C程序转化为机器语言的过程

41. _____ 一种软件工具，专门用来处理存储在网格或表中的数据

42. _____ 定义在程序中应该如何使用标点和语句的一套规则

43. _____ 用户和计算机硬件之间的交互界面

44. _____ 计算机的一部分，用来执行数学计算

45. _____ 将错误从程序中剔除的过程

46. _____ 在程序执行过程中发现的错误
47. _____ 由美国国家标准协会指定的C语言定义
48. _____ 包含在一个集成电路芯片的中央处理单元
49. _____ 远程获取海量信息的技术
50. _____ 程序的二进制表示

附加题

通过完成下列问题中所给的任务，不仅可以更加深入地了解本章介绍的主题，还可以进一步提升书面沟通能力。（说不定你的导师还会从中挑选出一些报告来作为课堂演讲。）每篇报告要求至少包含两篇参考文献。使用一款文字处理软件来编写主题报告。如果对软件的操作还不熟悉，可以向导师寻求软件的帮助手册或是参加简单的集中式培训。学会使用学校计算机系统中可用的文字处理软件，并完成以下主题报告。

51. 在下列给出的工程应用成果中挑选一个作为主题，写一份简要的主题报告。

移动	应用卫星
微处理器	CAD/CAM
CAT 扫描	复合材料
大型喷气式客机	激光器
光纤技术	基因工程产品

52. 从这些伟大的工程应用成果中挑选一个，并从伦理道德的视角展开讨论，写一份简要的主题报告。报告中要列出几种对所选问题可能存在的看待方式。 23

简单的 C 程序

犯罪现场调查：法医人类学

法医人类学是将生物人类学知识与身份鉴别和死亡环境调查结合起来的研究领域。法医人类学家不仅要经常面对骨骼残骸，还要面对腐烂、烧焦或者局部残骸。他们的工作甚至可能是单个骨骼，也可能是飞机事故、爆炸或者自然灾害造成的大规模死亡中的许多骨骼。比如，在世贸大厦的恐怖袭击中，总共死亡2016人，但只有289具尸体是完整的。还有2011年乔普林龙卷风造成的大规模死亡，当龙卷风经过乔普林、密苏里时有1英里[⊖]宽，使得162人死亡。法医人类学家经常和法医牙科学家（牙科专家）、放射专家、指纹专家及执法人员一起鉴定残骸，并且提供和死亡情况有关的线索。在本章后面，会讨论某几块特定骨骼的长度和人体身高的关系，然后给出分别用股骨（连接臀部和膝盖之间的大腿骨）和桡骨（连接肩部和手部的骨头）长度估计一个人身高的 C 语言程序。

24

学习目标

在本章，将要学到以下解决问题的方法：

- 简单算术运算。
- 在屏幕上输出信息。
- 用户从键盘输入信息。
- 线性插值。

2.1 程序结构

本节将分析一个特定 C 程序的结构，然后介绍 C 程序的一般结构。下面的程序是在第 1 章中引用过的，其功能是计算和输出两点之间的距离。

```
/*-----------------------------------------------------------*/
/*   程序 chapter1_1                                         */
/*                                                           */
/*   该程序计算两点之间的距离                                */
```

25

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

⊖ 1 英里 = 1.609 3 千米

```
int main(void)
{
   /*  声明和初始化变量  */
   double x1=1, y1=5, x2=4, y2=7,
          side_1, side_2, distance;

   /*  计算直角三角形的边  */
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);

   /*  打印距离  */
   printf("The distance between the two points is "
          "%5.2f \n",distance);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

下面简要说明该程序中的语句，每条语句的详细内容将在本章后面叙述。

本程序的前4行是注释（comment）。注释给出了程序的名字（chapter1_1）和功能。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter1_1                                                */
/*                                                                 */
/*  该程序计算两点间的距离                                         */
```

注释以字符“/*”开始，以字符“*/”结束。注释可以自己占一行，也可以与程序语句在同一行。还可以延续几行。本程序中的每行都是一段单独的注释，因为每行都是以“/*”开始，以“*/”结束。虽然在一段程序中注释是可有可无的，但是好的编程习惯是让注释始终贯穿整个程序，这样可以提高程序的可读性，并且注释本身也可以成为程序的文档，记录设计的过程和计算的原理。本书的程序都是在程序开头的注释中给出程序的名字和功能，并且在程序始末都贯穿对程序语句的解释性注释。虽然ANSI C规定了注释和语句可以从一行的任何地方开始书写，但是在本书中，程序的起始注释总是从一行的开头位置写起的。

预处理指令（preprocessor directive）是指在程序编译之前要执行的指令。最常用的预处理指令是在程序中插入附加语句，这种指令以#include开头，其后则为包含附加语句的文件名称。该程序包含以下两条预处理指令：

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

这些语句表明将stdio.h和math.h两个文件中的所有语句包含到本文件中，在程序编译之前替换掉这两条语句。文件名两边的“<”和“>”符号说明该文件位含在C标准函数库（standard C library）中。C标准函数库集成在符合ANSI C标准的C语言编译器运行环境中。stdio.h文件包含了与输出语句有关的内容，math.h文件位含了与计算平方根的函数有关的内容，这些都在程序中用到了。文件名后面的.h扩展名说明该文件是头文件。更多关于头文件的内容在本章后面和第4章中讲述。预处理指令一般写在描述程序功能的起始注释之后。 [26]

每个C程序都有一组被称为main函数的语句。关键字int表示函数向操作系统返回一个整型值。关键字void表示函数从操作系统没有接收到任何信息。函数的主体是用花括号{}括起来的。为了便于识别函数主体，把花括号单独放一行。因此，在预处理指令下面的

两行语句说明了主函数的开始：

```
int main(void)
{
```

该程序的主函数包括两种类型的命令：声明和语句。声明（declaration）定义了要被语句使用的存储单元，因此，声明必须在语句之前。声明时给定初值（initial value）和不给定初值都可以。该程序的声明语句之前有一行注释：

```
/*  声明和初始化变量  */
double x1=1, y1=5, x2=4, y2=7,
       side_1, side_2, distance;
```

这些声明语句说明程序要用到7个变量，分别是x1、y1、x2、y2、side_1、side_2和distance。关键字double表示这些变量存储为双精度浮点值。这些变量可以存储高精度的数值，如12.5和-0.000 5。此外，这条语句还说明x1被初始化（赋初值）为1，y1被初始化为5，x2被初始化为4，y2被初始化为7。side_1、side_2和distance没有赋初值，注意不能把没有赋值的变量初值认为是0。因为声明语句太长，所以分为两行来写。第二行的缩进表明它是上一行的延续。缩进可以使程序样式更美观，并且提高程序的可读性，但并不是必需的。

下面是示例程序中要执行的操作语句（statement）：

```
/*  计算直角三角形的边  */
side_1 = x2 - x1;
side_2 = y2 - y1;
distance = sqrt(side_1*side_1 + side_2*side_2);
/*  打印距离  */
printf("The distance between the two points is "
       "%5.2f \n",distance);
```

以上语句计算了由两点形成的直角三角形（见图1-4）中两条直角边的长度，然后计算直角三角形的斜边长度。这些语句的语法细节会在后面的章节中说明。distance被计算出来后，用printf语句将结果输出。由于输出语句太长，不能用一行写出，所以把它分成两行来写。第二行的缩进表明它是上一行的延续。注释用来解释计算和输出语句。注意，声明和语句都要以分号结尾。

27

示例程序用return 0;语句结束程序的执行，并且将控制权交还给操作系统：

```
/*  退出程序  */
return 0;
```

上面的语句表示向操作系统返回一个值0。返回0表明程序成功执行。

main函数的主体以右花括号结束，右花括号单独占一行，后面一行注释描述主函数的结束：

```
}
/*-----------------------------------------------------------------*/
```

注意，该程序用空行（也叫空白）将程序分为了不同的几部分。这些空行使得程序更具可读性，而且更容易修改。主函数中的声明和语句的缩进能够显示程序的结构。语句中空格的使用使程序风格统一，并且可读性更强。

仔细看过第1章的C程序后，可以将它的结构与C程序的一般结构（general form）进

行比较：

```
预处理指令
int main(void)
{
    声明;
    语句;
}
```

本章和后面章节中的程序都符合这个结构。

修改

1. 创建一个文件，写入本节的示例程序。建立文件时可以使用C编译器中的代码编辑工具或是任意一个文字处理软件。[⊖]然后编译和运行程序。读者应该会得到下面的输出：

```
The distance between the two points is  3.61
```

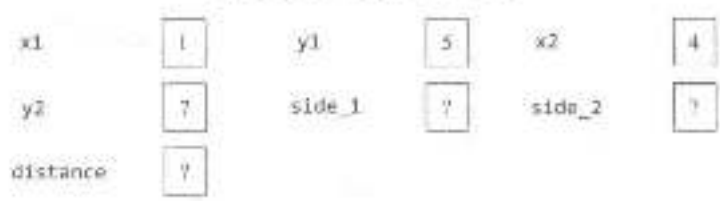
2. 将给出的两个点的坐标改成（-1，6）和（2，4）。运行改变坐标后的程序。distance改变了吗？请解释原因。
3. 将给出的两个点的坐标改成（1，0）和（5，7）。用计算器检查程序输出的结果。
4. 将给出的两个点的坐标改成（2，4）和（2，4）。程序输出正确结果了吗？

28

2.2 常量和变量

常量和变量是程序中使用的数值。常量（constant）是固定不变的量，例如C语句中的2、3.1416、-1.5、'a'或"hello"。变量（variable）是被分配了名字或者标识符（identifier）的存储单元。标识符用来引用存放在某个特定位置的存储单元的数值。存储单元和其对应标识符的关系可以类比为邮箱和它的主人的名字。存储单元（或者邮箱）里有一个值，利用标识符（或邮箱主人的名字）可以找到并访问这个值。下图展示了chapter1_1中声明语句完成后各变量、标识符及初始值的情况。

```
double x1=1, y1=5, x2=4, y2=7,
       side_1, side_2, distance;
```



没有被初始化的变量值是不确定的，用问号（?）表示。有时候这些值被称为垃圾值（garbage value），因为它们是以前程序运行后残留在内存中的数据。描述变量与其标识符及初始值的图被称为内存快照（memory snapshot）。通过内存快照可以看到程序执行中特定时刻的内存单元内容。上述内存快照展示了变量及由声明语句指定的变量内容。我们通常用内存快照来展现一条语句执行前后变量的内容变化，或者说明这条语句的功能。

下面是定义一个有效标识符的规则：

⊖ 如果使用文字处理软件生成一个文件，要确认把文件保存成了文本文件。

- 标识符必须以字母或者下划线（_）开头。
- 标识符中的字母可以是大写字母，也可以是小写字母。
- 标识符中可以含有数字，但是数字不能作为标识符的开头。
- 标识符的长度是任意的，但是任意两个标识符的前31个字符不能完全相同。

C语言标识符区分大小写（case sensitive）。大写字母和小写字母在C语言中是不一样的，所以Total、TOTAL和total表示三个不同的变量。还有，要注意distance_in_miles_from_earth_to_mars和distance_in_miles_from_earth_to_venus会被当成同一个变量，因为两个字符串前31个字符是相同的。C中还包括一些关键字（key word）。关键字对于C编译器而言有特殊的含义，因此是不能用作标识符的。关键字的完整列表在表2-1中给出。

表2-1 关键字

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Distance、x_1、X_sum、average_measurement和initial_time都是有效的标识符。下面几个是无效的标识符：1x（以数字开头）、switch（switch是关键字）、$sum（包含了无效字符$）和rate%（包含了无效字符%）。

[29]

标识符应该仔细命名，使其名字能表达变量的含义，最好还能表明变量的计量单位。例如，一个变量表示华氏温度，标识符可以命名为temp_F或者degrees_F；一个变量表示角，标识符可以命名为theta_rad，来表明角使用弧度制计量单位，也可以命名为theta_deg，表明角使用角度制计量单位。

在main函数开头（或者其他C函数开头）的声明，不仅要写出程序中所使用变量的标识符，还要指定变量的类型。数据类型的使用方法将在我们讨论完科学计数法后讲述。

练习

下面的名称哪些是有效标识符？如果是无效标识符，请说明原因，并且将它修改为有效标识符。

1. density	2. area	3. time
4. xsum	5. x_sum	6. tax-rate
7. perimeter	8. sec^2	9. degrees_C
10. break	11. #123	12. x&y
13. count	14. void	15. f(x)
16. f2	17. Final_Value	18. w1.1
19. reference1	20. reference_1	21. m/s

2.2.1 科学计数法

浮点（floating-point）数可以是整数，也可以是非整数，例如2.5、-0.004和15.0。浮点值的科学计数法（scientific notation）表示为尾数乘以10的幂的形式，其中尾数的绝对值大于等于1.0且小于10。例如，科学计数法下，25.6写为2.56×10^1，-0.004写为-4.0×10^{-3}，1.5写为1.5×10^0。指数计数法（exponential notation）中，用e来隔开尾数和10的指数。因此，指数表示法中，25.6写为2.56e1，-0.004写为-4.0e-3，1.5写为1.5e0。 30

尾数的小数部分的位数决定了数值的精度，指数的位数决定了数值的范围，尾数和指数共同决定了数的大小。因此，尾数部分精度值为2位，指数范围为-8 ~ 7，就可以表示2.33×10^5（233 000）和5.92×10^{-8}（0.000 000 059 2）这样的数据。对于工程问题中用到的许多数值类型来说，这个精度和指数范围是不够的。例如，从火星到太阳的距离是141 517 510英里或者$1.415\ 175\ 1 \times 10^8$英里[⊖]，为了表示这个数，科学计数法下，尾数部分至少需要7位小数，指数部分至少要达到8。

练习

用科学计数法表示下面的数。

1. 35.004　　2. 0.000 42

3. -50 000　　4. 3.157 23

5. -0.099 9　　6. 10 000 002.8

用浮点形式表示下面的数。

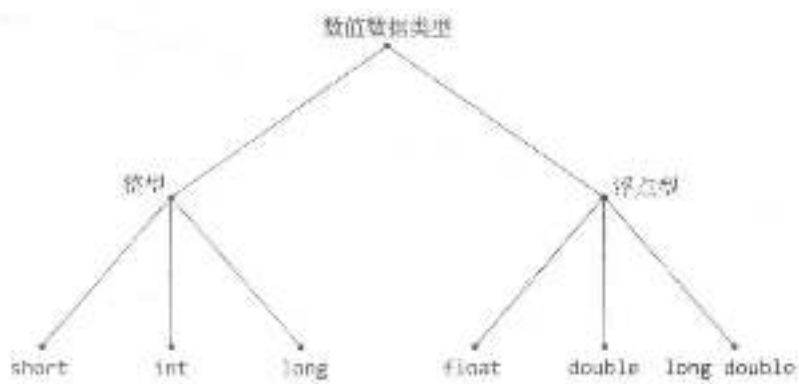
7. 1.03e-5　　8. -1.05e5

9. -3.552e6　　10. 6.67e-4

11. 9.0e-2　　12. -2.2e-2

2.2.2 数值数据类型

数值数据类型用来说明存储在变量中的数的类型。在C语言中，数字不是整数就是浮点数。下图展示了后面要讲到的数值数据类型。



31

对于有符号整数，短整型、整型、长整型的类型标识符（type specifier）分别为short、int和long，这些数据类型表示的数据范围是系统相关的（system dependent），这意味着

⊖ 1英里=1.609 3千米

在不同的系统中相同的数据类型可表示的数据范围可能不同。本章最后部分给出了一个程序，可以用来检测系统数值数据类型的范围。在大多数系统中，短整型和整型的范围是 −32 768 ~ 32 767，长整型可以表示的范围是 −2 147 483 648 ~ 2 147 483 647。（范围的边界值，如 32 767 和 2 147 483 647，是将二进制转化为十进制而得到的）。C 语言允许在整型标识符前加上 unsigned 限定符，无符号整数只能表示正数。有符号整数和无符号数可以表示的数的个数相同，但范围不同。例如，unsigned short 能够表示的数的范围是 0 ~ 65 535，但是 short 能表示的数的范围是 −32 768 ~ 32 767，两者都能表示 65 536 个数。

浮点型的类型标识符有 float（单精度）、double（双精度）和 long double（扩展精度）。下面是 Chapter1_1 中的语句，定义了 7 个双精度类型的变量：

```
double x1=1, y1=5, x2=4, y2=7,
       side_1, side_2, distance;
```

float、double 和 long double 区别在于能够表示的数的精度（或者准确度）和范围。精度和范围也是系统相关的。表 2-2 展现了 Microsoft Visual C++ 编译器中整型和浮点型的精度和范围，该编译器能够处理 C 程序和 C++ 程序（C++ 是 C 的扩展）。2.10 节中给出了一个程序，可以得到计算机系统的这些信息。在大多数系统中，double 类型所能精确到的小数位数是 float 类型的两倍。

表 2-2 数据类型范围示例[1]

整数	
short	最大值 = 32 767
int	最大值 = 2 147 483 647
long	最大值 = 2 147 483 647
浮点数	
float	6 位精度 最大指数为 38 最大值为 3.402 823e + 38
double	15 位精度 最大指数为 308 最大值为 1.797 693e + 308
long double	15 位精度 最大指数为 308 最大值为 1.797 693e + 308

[32]

① Microsoft Visual C++ 2010 Express 编译器

除此之外，double 类型比 float 类型表示的数的范围更大，long double 类型有更高的精度和更大的范围，但这也是系统相关的。float 类型常量，如 2.3，可以看成 double 类型常量，为了区分 float 类型常量和 long double 类型常量，在 float 类型常量后面加上字母（或后缀）F，在 long double 类型常量后面加上字母 L。所以，2.3F 和 2.3L 分别表示一个 float 类型常量和一个 long float 类型常量。

2.2.3 字符型数据

在解决工程问题的程序设计中，字符型数据是一种重要数据类型，经常用于表示和操作各种类型的文本。为了更好地应用字符型（character）数据，我们必须了解它在计算机内存中

的表示方法。计算机内部都是以一系列二进制数字（0和1）来存储内容的，因此每个字符都有其对应的二进制编码（binary code）。最常见的二进制编码是ASCII（American Standard Code for Information Interchange）和EBCDIC（Extended Binary Coded Decimal Interchange Code）。在后面的讨论中，我们用ASCII码来表示字符。表2-3展示了一些字符的ASCII编码的二进制形式，以及对应的十进制整数形式。例如，字符'a'的ASCII编码用二进制表示为1100001，等于整数97。ASCII编码中总共有128个字符，完整的ASCII编码表见附录B。

字符型数据可以表示成常量，也可以表示成变量。字符型常量放在单引号内，例如'A'、'a'和'3'。用于存储字符的变量的数据类型必须被定义为整型或者字符型，字符型的类型说明符是char。

字符以二进制形式存储在内存中后，这个二进制数据既可以被计算机解读为字符，也可以被解读为整数。因此，在存储字符时，可以声明一个字符变量存储其ASCII值，也可以声明一个整型变量存储相对应的整数值。要特别注意的是，一个字符的ASCII码的二进制表示法和一个整数的二进制表示法是不同的。从表2-3可以看到，字符'3'的ASCII码二进制表示为0110011，它和整数51的二进制表示法相同。因此，用字符方式计算和用整数方式计算得到的结果是不同的。下面的程序展示了字符型数据的使用方法，程序的详细内容将在本章后面说明。

表2-3 ASCII码示例

字符	ASCII码	相等的整数值
newline, '\n'	0001010	10
%	0100101	37
3	0110011	51
A	1000001	65
a	1100001	97
b	1100010	98
c	1100011	99

33

```
/*-----------------------------------------------------------------*/
/*  程序chapter2_1                                                 */
/*                                                                 */
/*  该程序打印字符型和整型的两个值                                 */

#include <stdio.h>

int main(void)
{
   /*  声明和初始化变量  */
   char ch='a';
   int i=97;

   /*  以字符形式打印这两个值  */
   printf("value of ch: %c; value of i: %c \n",ch,i);

   /*  以整型打印这两个值  */
   printf("value of ch: %i; value of i: %i \n",ch,i);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

本程序的输出是：

```
value of ch: a; value of i: a
value of ch: 97; value of i: 97
```

2.2.4 符号常量

符号常量（symbolic constant）是在预编译指令中定义的，用于给常量分配一个标识符。符号常量可以在C程序的任何部位声明，编译器会把声明语句之后的语句中出现的标识符替换为它所代表的常量。工程中的常数一般用符号常量代替，如 π 和 g（重力加速度）。下面的预编译指令将值3.141 593赋给变量PI：

```
#define PI 3.141593
```

如果语句中要用到 π 的值，可以用符号常量标识符PI代替3.141 593，就比如下面计算圆面积的语句：

```
area = PI*radius*radius;
```

符号常量标识符一般用大写字母（比如上例的PI而不是pi），以表明它是符号常量，同时定义标识符时最好定义为容易记忆的。最后，一条预编译指令只能定义一个符号常量，如果要定义好几个符号常量，那么就需要几条相应的预编译指令。注意在含#define的预编译指令结束时不用写分号。

下一节将讨论C语言中的赋值语句。赋值语句用于给变量赋予特定的值，因此也可以用来将常量赋值给某个变量，但相比起来用符号常量有更多的优点，这些优点我们在后面会讨论。

34

练习

给出下面需要定义成符号常量的常数值的预处理指令。

1. 光速：$c = 2.997\,92 \times 10^8$ m/s
2. 电子的电荷：$e = 1.602\,177 \times 10^{-19}$ C
3. 阿伏伽德罗常数：$N_A = 6.022 \times 10^{23}$ mol^{-1}
4. 重力加速度：$g = 9.8$ m/s^2
5. 重力加速度：$g = 32$ ft/s^2
6. 地球质量：$M_e = 5.98 \times 10^{24}$ kg
7. 月球半径：$r = 1.74 \times 10^6$ m
8. 长度单位：Unit_Length = 'm'
9. 时间单位：Unit_Length = 's'

2.3 赋值语句

赋值语句（assignment statement）用来给标识符赋值。赋值语句的一般格式为

```
标识符 = 表达式;
```

其中表达式（expression）可以是常量、变量或者运算结果。看下面的两组语句，都是声明了变量sum、x1和ch，并且给变量赋了值。

```
double sum=10.5;          double sum;
int x1=3;                 int x1;
char ch='a';              char ch;
                          ...
                          sum = 10.5;
                          x1 = 3;
                          ch = 'a';
```

两组语句执行结果一样，sum 的值是 10.5，x1 的值是 3，ch 的值是 'a'，如下面的内存快照所示：

在上面的例子中，左边的语句定义变量的同时对变量进行了初始化，右边的赋值语句可以放在程序的任何地方，用来改变变量的值。

C 允许多重赋值，就如下面的语句，给 x，y 和 z 都赋值为 0。

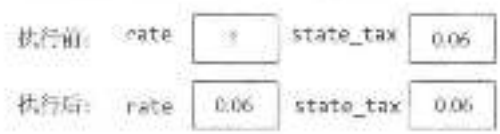
```
x = y = z = 0;
```

多重赋值在本节结尾介绍。 35

也可以用赋值语句将一个变量的值赋给另一个变量：

```
rate = state_tax;
```

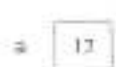
赋值语句中的等号读作“被赋值为……”，因此，赋值语句的含义是将 state_tax 的值赋给 rate。如果 state_tax 的值是 0.06，那么在赋值语句执行后，rate 的值也是 0.06，state_tax 的值不变。因此赋值语句执行前后内存快照如下所示：



如果给变量赋的值和变量的数据类型不同，在赋值语句执行过程中就会发生数据类型转换。有时数据类型转换会使信息丢失。例如，思考下面的声明语句和赋值语句：

```
int a;
...
a = 12.8;
```

因为 a 被定义为整型，所以它不能存储小数，因此，赋值语句执行后的内存快照如下所示：



为了保证在数据转换时不发生错误，我们按照如下的顺序（从高到低）排列所有的数据类型，并以此来限制数据转换的方向：

```
高  long double
    double
    float
    long integer
    integer
低  short integer
```

如果一个数据是按上面的顺序转换为一个比它高的数据类型，则信息不会丢失。但是如果转换为比它低的数据类型，信息可能会丢失。因此，从 integer 型转换为 double 型将会正常转换，但从 float 型转换为 integer 型将会导致信息丢失或者结果错误。一般而言，我们只按照不会导致潜在的数据转换问题的方式使用赋值语句，也就是只向更高的数据类型进行转换。（无符号整型不包含在上面的转化列表中，因为对无符号整型数据来说，两个方向的转换都可能引起错误。）

2.3.1 算术运算符

赋值语句可以将算术运算的结果赋值给变量，下面的语句可以计算正方形的面积：

36

```
area_square = side*side;
```

用 * 表示乘号，符号 + 和 - 分别表示加号和减号，符号 / 表示除法。因此，下面两条语句都是有效的计算三角形面积的公式。

```
area_triangle = 0.5*base*height;
area_triangle = (base*height)/2;
```

其中第二条语句中的圆括号不是必需的，但使用圆括号可以增强可读性。

看下面的赋值语句：

```
x = x + 1;
```

在代数中，这条语句是非法的，因为一个数不可能等于它本身加 1。但是在赋值语句中，它的含义不是相等，而是把 x+1 的值赋给 x，因此上面的语句就是将存在 x 中的值加 1。如果变量 x 是 5，在语句执行后，变量 x 的值就变为 6。

C 语言也包括取模（modulus）运算符（%），用来计算两个整数相除的余数。例如，5%2 等于 1，6%3 等于 0，2%7 等于 2（2/7 的商是 0，余数是 2）。当 a 和 b 都是整数时，a/b 计算的是商，a%b 计算的是余数。例如，a 等于 9，b 等于 4，那么 a/b 的值是 2，a%b 的值是 1。但是如果 b 等于 0，那么不管是 a/b 还是 a%b，语句执行都会发生错误，因为计算机不能执行除数为 0 的操作。如果 a 和 b 有一个是负数，那么 a%b 的结果是系统相关的。

取模运算常用于判断一个数是否为另一个数的倍数。例如，如果 a%2 等于 0，那么 a 就是偶数，否则 a 就是奇数。如果 a%5 等于 0，那么 a 就是 5 的倍数。在工程问题解决中经常用到取模运算。

前面提到的 5 个运算符（+、-、*、/、%）都是双目运算符（binary operator）——需要两个数据参与运算的运算符。C 语言也包含单目运算符（unary operator）——只需要单个数据参与的运算符。例如，当加和减用在像 -x 这样的表达式中就是单目运算符。

双目运算的结果和操作数的数据类型一样。例如，如果 a 和 b 都是 double 型，那么 a/b 的结果也是 double 型的，同样的，如果 a 和 b 都是 integer 型，那么 a/b 的结果也是 integer 型。但有时候整数相除会得到不准确的结果，因为相除后结果的小数部分会被舍弃，因此得到的结果是被截断后的值，而不是完整的值。所以，5/3 等于 1，3/6 等于 0。

不同数据类型的值之间的运算称为混合运算（mixed operation）。在运算前，两个变量中较低的数据类型会被自动转换为较高的数据类型（就是在赋值语句中提到的数据类型转换高低规则），使得相同数据类型的值进行运算。例如，float 型和 int 型相运算，在运算执行前，int 型会被转换成 float 型，运算后的结果也是 float 型。

假如要计算一组整数的平均值，如果这组整数的总和与个数被存储在了变量 sum 和 count 中，下面的语句似乎能计算出正确的均值：

37

```
int sum, count;
float average;
...
average = sum/count;
```

然而，两个 integer 型数据相除后得到的是 integer 型的结果。除法的结果会在赋值时被转换成 float 型。因此，如果 sum 等于 18，count 等于 5，运算后 average 等于 3.0，而不是 3.6。为了使计算的和更准确，可以使用强制转换运算符（cast operator）。强制转换运算符是将特定值转换为指定类型的一种单目运算符。下面的例子中，将强制转换（float）添加在 sum 之前：

```
average = (float)sum/count;
```

在除法执行之前，sum 的值被转换为 float 型，接下来的除法是 float 型和 integer 型间的混合运算，因此 count 的值被转换为 float 型，最后除法的运算结果也是 float 型，并且存储在 average 中。如果 sum 的值是 18，count 的值是 5，此时计算后得到更准确的 average 的值，是 3.6。注意，强制转换运算符只影响计算时的值，不影响存储在变量 sum 中的值。

练习

给出下面语句计算后的值。

```
1. int a=27, b=6, c;
   ...
   c = b%a;

2. int a=27, b=6;
   float c;
   ...
   c = a/(float)b;

3. int a;
   float b=6, c=18.6;
   ...
   a = c/b;

4. int b=6;
   float a, c=18.6;
   ...
   a = (int)c/b;
```

2.3.2 运算符优先级

当表达式中包含一个以上的算术运算符，就要确定运算符的计算顺序。表 2-4 展示了算术运算符的优先级（precedence）。C 语言中的运算顺序和代数运算顺序是一样的。括号中的先运算，如果括号是嵌套的，那么最里面括号中的先运算，单目运算在双目运算 *、/、% 之前运算，二元加减最后运算。如果表达式中有几个优先级相同的运算符，变量或常量与运算符按照表 2-4 中的指定顺序结合。例如下面的表达式：

38

```
a*b + b/c*d
```

表 2-4 算术运算符优先级

优先级	运算符	结合性
1	括号（）	从内向外
2	单目运算符：+、-	从右至左
3	双目运算符：*、/、%	从左至右
4	双目运算符：+、-	从左至右

因为乘法和除法有相同的优先级，又因为结合性（assocativity）（对操作进行分组的顺序）是从左到右，所以这个表达式的计算顺序可以表示如下：

```
(a*b) + ((b/c)*d)
```

优先顺序并没有指定 a*b 要在（b/c）*d 之前运算，这种类型的运算顺序是系统相关的（但是不影响运算结果）。

算术表达式中的空格是编程风格的问题，有些人喜欢每个运算符周围都放空格，本书只在二元加减运算符周围放空格，因为二元加减是最后运算的。编程时可以选择自己喜欢的空格使用风格，但是选择之后最好一直使用同一种风格。

假设要计算梯形的面积，已经声明了4个double型变量：base、height_1、height_2 到 area，假设变量 base、height_1 和 height_2 都已赋值，能够正确计算梯形面积的语句如下：

```
area = 0.5*base*(height_1 + height_2);
```

假设省略表达式中的括号：

```
area = 0.5*base*height_1 + height_2;
```

这个表达式将按照下面的表达式执行：

```
area = ((0.5*base)*height_1) + height_2;
```

注意，虽然得到的是错误结果，但是将不会产生任何错误提示信息。因此，使用C语言写表达式时要特别仔细。在复杂的表达式中，可以多使用圆括号来表明运算顺序，这种方式可以避免混淆，而且确保得到想要的计算结果。

注意，C语言中没有幂运算符，例如 x^4，本章后面会介绍一个数学函数来进行幂运算。对于指数是整数的幂运算，比如 a^2，可以用乘法 a*a 代替。

一个长的计算公式可以分为几条语句。例如下面的计算：

$$f = \frac{x^3 - 2x^2 + x - 6.3}{x^2 + 0.05005x - 3.14}$$

39

如果用一条语句来描述这个表达式，就会太长，不容易读：

```
f = (x*x*x - 2*x*x + x - 6.3)/(x*x + 0.05005*x - 3.14);
```

我们可以将这条语句写为两行：

```
f = (x*x*x - 2*x*x + x - 6.3)/
    (x*x + 0.05005*x - 3.14);
```

另一种解决方法是将分子和分母分开来算：

```
numerator = x*x*x - 2*x*x + x - 6.3;
denominator = x*x + 0.05005*x - 3.14;
f = numerator/denominator;
```

为了计算出 f 的正确结果，变量 x、numerator、denominator 和 f 都必须是浮点型变量。

练习

在练习1～3中，给出用于计算指定值的C语句。假设表达式中用到的标识符都已被声明成double类型并赋予了合适的值。使用的重力加速度常量值为 $g = 9.806\,65\ m/s^2$。

1. 移动距离：

$$\text{Distance} = x_0 + v_0 t + \frac{1}{2}at^2$$

2. 绳子的拉力：

$$\text{Tension} = \frac{2m_1 m_2}{m_1 + m_2} \times g$$

3. 管道不增液体压力：

$$P_2 = P_1 + \frac{\rho v_1^2 (A_2^2 - A_1^2)}{2A_2^2}$$

在练习4～6中，写出C语句描述的数学方程式。假设下面的符号常量都已被定义，其中G的单位是$m^3/(kg \cdot s^2)$。

```
#define PI 3.141593
#define G 6.67259e-11
```

4. 向心加速度

```
centripetal = 4*PI*PI*r/(T*T);
```

5. 势能

```
potential_energy = -G*M_E*m/r;
```

6. 势能变化量

```
change = G*M_E*m*(1/R_E - 1/(R_E + h));
```

40

2.3.3 上溢和下溢

存储在计算机中的数值都有一个允许范围，如果计算结果超出了允许范围，就会发生错误。例如，假设浮点数的指数的允许范围是-38～38，这个范围对于大多数计算都适用，但是有可能某个表达式的结果会超过这个范围。例如，执行下面的命令：

```
x = 2.5e30;
y = 1.0e30;
z = x*y;
```

x和y的值都在允许的范围内，但z的值为2.5e60，是超过范围的。这类错误称为指数上溢（overflow），因为算术运算结果的指数太大，无法存储到分配给变量的内存中。发生指数上溢的行为是系统相关的。

指数下溢（underflow）是类似的错误，是由于算术运算结果的指数太小，以至于不能存储在分配给变量的内存中。假设浮点数的指数的允许范围与前面的例子相同，下面是一个指数下溢的例子：

```
x = 2.5e-30;
y = 1.0e30;
z = x/y;
```

x和y的值在允许范围内，但是z的值是2.5e-60。因为指数比允许的最小值小，所以导致了指数下溢。同样，指数下溢的行为也是系统相关的。在一些系统中，指数下溢的运算结果被存储为0。

2.3.4 自增运算符和自减运算符

C语言中一个变量的自增或自减可以通过一些单目运算符来实现，但是这些运算符不能用于常量或者表达式。自增运算符 ++ 和自减运算符 -- 可以放在前缀（prefix）位置（标识符之前），例如 ++count，也可以放在后缀（postfix）位置，如 count++。如果一个变量用了自增或自减运算符，就等价于将自身加1或者减1后的值赋给自身。因此，语句

```
y--;
```

等价于语句

```
y = y - 1;
```

如果在表达式中使用自增或者自减运算符，那么一定要仔细分析这个表达式。如果运算符被放在前缀位置，那么该变量的值将先被修改，然后用新的值计算表达式的剩余部分。如果运算符被放在后缀位置，那么该变量的值先被用来计算表达式的剩余部分，然后变量的值才会被修改。因此执行语句

41

```
w = ++x - y;                                    (2.1)
```

等价于执行下面的语句：

```
x = x + 1;
w = x - y;
```

同样，语句

```
w = x++ - y;                                    (2.2)
```

等价于下面的语句：

```
w = x - y;
x = x + 1;
```

假设x的值是5，y的值是3，在执行（2.1）或（2.2）后，x的值增加为6。但是，（2.1）执行后w的值为3，（2.2）执行后w的值为2。

自增、自减运算符和其他单目运算符的运算优先级是一样的，如果表达式中有好几个单目运算符，它们的结合性是从右向左。

2.3.5 缩写赋值运算符

C语言支持用缩写的形式简化简单的赋值语句。例如，下面每组中包含的两条语句都是等价的：

```
x = x + 3;
x += 3;

sum = sum + x;
sum += x;

d = d/4.5;
d /= 4.5;

r = r%2;
r %= 2;
```

事实上，任何形如

标识符 = 标识符 运算符 表达式;

的语句都可以写为

标识符 运算符 = 表达式;

经常使用缩写赋值语句是因为它比较短。

在本章前面，使用了下面的多重赋值（multiple-assignment）语句：

```
x = y = z = 0;
```

该语句的表示很清晰，但是下面语句的表示就不那么明确了：

```
a = b += c + d;
```

为了正确计算，我们使用表2-5的规则。表2-5显示赋值运算最后计算，并且其结合性从右至左。因此这条语句等价于下面的语句

```
a = (b += (c + d));
```

42

表2-5 算术和赋值运算符的优先级

优先级	运算符	结合性
1	括号：( )	从内向外
2	单目运算符：+ - ++ -- (type)	从右至左
3	双目运算符：*/%	从左至右
4	双目运算符：+ -	从左至右
5	赋值运算符：= += -= *= /= %=	从右至左

如果将上述缩写形式写为正常形式，可以写为

```
a = (b = b + (c + d));
```

或者

```
b = b + (c + d);
a = b;
```

虽然这条语句可以很好地练习运算符优先级和结合性，但是严重破坏了程序的可读性。因此，在多重赋值语句中，不推荐使用缩写赋值语句。需要提到的是，本书的空格使用习惯是在缩写运算符或者多重赋值运算符两侧加入空格，因为这些运算都是在算术运算之后进行的。

练习

给出每条语句执行后的内存快照。假设在语句执行之前 x = 2，y = 4，且所有变量都是整型。

1. z = x++*y;　　2. z = ++x*y;　　3. x += y;　　4. y %= x;

2.4 标准输入和输出

我们已经讨论了变量声明语句，并且能够利用这些变量去计算新的数据，现在介绍一种能够输出打印这些变量的语句。除此之外，本节还会介绍一种语句，能够在程序执行时读取键盘输入以改变变量的值。为了能使用这些语句，在程序中必须包含下面的预编译指令

43

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

这条指令告诉编译器，这段代码中将使用标准C库中输入/输出相关的函数。

2.4.1 输出函数printf

输出函数printf可以在屏幕上输出值和说明性文字。例如，下面的语句输出一个名称为angle、数据类型为double的数，同时还会输出它的单位：

```
printf("Angle = %f radians \n",angle);
```

这条输出语句包含两个参数，控制字符串和打印变量的标识符。控制字符串（control string）要用双引号引起来，它可以包括文本或者转换说明符，也可以两者都包括。转换说明符（conversion specifier）来说明要输出变量的类型。在上面的例子中，控制字符串指定的字符Angle= 被输出出来，接下来的字符（%f）是一个转换说明符，它表示将会输出一个变量的值。然后输出字符串radians。接下来的字符（\n）是换行指示符，换行符之后会在屏幕新起一行输出后面的内容。这条输出语句中的第二个参数是angle，它和控制字符串中的转换说明符相匹配，因此，根据说明符%f，angle的值被输出来。如果angle的值是2.84，这条输出语句的输出就是：

```
Angle = 2.840000 radians
```

我们已经分析了简单的输出语句和它的输出结果，现在来更深入地了解转换说明符。

为了能够正确打印变量的值，需要根据变量的数据类型从表2-6中选取相应的转换说明符。例如，要输出short型或者int型的数字，使用说明符%i（整数）或者%d（十进制数），两种说明符得到的结果是一样的。要输出long型的数字，使用%li或%ld说明符。要输出
44 float型或者double型，要使用说明符%f（浮点格式）、%e（指数格式，如2.3e+02）或者%E（指数格式，如2.3E+02）。说明符%g（通用格式）会根据输出值自身的大小，来选择用标识符%f或是标识符%e输出。标识符%G和%g的功能基本一样，只是使用%g标识符显示时，当输出值需要使用指数表示法时会自动使用%e标识符，而%G则会使用%E。

表2-6 输出语句的转换说明符

变量类型	输出类型	说明符
	整数值	
short, int	int	%i, %d
int	short	%hi, %hd
long	long	%li, %ld
int	unsigned int	%u
int	unsigned short	%hu
long	unsigned long	%lu
	浮点数值	
float, double	double	%f, %e, %E, %G, %G
long double	long double	%Lf, %Le, %LE, %Lg, %LG
	字符型值	
char	char	%c

选择正确的说明符后，还可以添加其他控制信息。可以指定最小字段宽度（field width），也可以控制输出数据的精度。控制串中可以同时设置字段宽度和精度（precision），也可以只指定其中一个。如果精度说明符没有给定，标识符 %f 的默认精度值是 6。一个数的小数部分会被四舍五入到指定精度，因此，如果转换说明符是 %.2f，14.516 78 会被输出为 14.52。%5i 表明要输出的 short 或者 int 型数据占据的最小字段宽度是 5，也就是说，如果输出值的字段宽度大于 5，那么它输出时可以增加字段宽度；如果输出值小于设定的宽度，那么输出值就会右对齐（right justify），而用空格填充输出值左边的多余位数。如果想要指定左对齐（left justify），就要在字段宽度前插入一个负号，如 %-8i。如果字段宽度前有一个正号，如 %+6f，那么输出值前面就会有一个正号被输出。

下面的列表展现了对于一个给定值，转换说明符和其对应的输出字段。其中字符 ␣ 用于表示字段中的一个空格。在这些示例中，假设列表中给定的值是 -145。

说明符	输出值
%i	-145
%4d	-145
%3i	-145
%6i	␣␣-145
%-6i	-145␣␣

下面的列表展现了当给定值是 double 型的 157.892 6 时，转换说明符和其对应输出字段：

说明符	输出值
%f	157.892 600
%6.2f	157.89
%+8.2f	␣+157.89
%7.5f	157.892 60
%e	1.578 926e+02
%.3E	1.579E+02
%g	157.893

注意最后两个转换说明符发生了四舍五入。

如果控制参数中包含三个转换说明符，那么就要有三个对应的标识符或表达式跟在控制串后面。就比如下面的语句：

```
printf("Results: x = %5.2f, y = %5.2f, z = %5.2f  \n",
       x,y,z+3);
```

它的输出是：

```
Results: x =  4.52, y =  0.15, z = -1.34
```

45

注意最后一个转换说明符是和算术表达式匹配，而不是变量。

在控制字符串中反斜杠（\）是转义符（escape character）。转义符和它后面的字母组合起来会有不同的解释，编译器会将转义符和它后面的字母当作一组来处理。例如，在前面我们已经知道 \n 表示另起一行，除此之外，\\ 表示在控制字符串中插入一个反斜杠，\" 表示在控制字符串中插入一个双引号，因此

```
printf("\"The End.\"\n");
```

的输出是一行

```
"The End."
```

下面是C语言中使用的其他一些转义符：

序列	字符含义
\a	警报（铃）字符
\b	退格
\f	换页
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符
\\	反斜杠
\?	问号
\'	单引号
\"	双引号

如果一条输出语句过长，则可以把它分成两行来写，但分的时候要保持语句的可读性。如果要把引号里面的文本分为两行来写，那么每行的文本都要用引号引起来。下面是几种将语句分开的正确写法：

```
printf("The distance between the points is %5.2f \n",
       distance);

printf("The distance between the points is"
       " %5.2f \n",distance);

printf("The distance between the "
       "points is %5.2f \n",distance);
```

合理地使用转换说明符，可以提高程序输出结果的可读性和可用性。在解决工程问题时，需要特别注意的是输出数据的数值的同时要输出它的单位。

虽然输出函数printf是要输出信息用的，但它也会有一个返回值，返回的是输出字符的个数。

46

练习

假设整型变量sum的值是65，双精度型变量average的值是12.368，字符型变量ch的值是'b'，写出下列语句的输出结果。

```
1.printf("Sum = %5i; Average = %7.1f \n", sum, average);
2.printf("Sum = %4i \n Average = %8.4f \n", sum, average);
3.printf("Sum and Average \n\n %d %.1f \n", sum, average);
4.printf("Character is %c; Sum is %c \n", ch, sum);
5.printf("Character is %i; Sum is %i \n", ch, sum);
```

```
6.printf("%7.2f is the average; \n", average);
  printf("%8d is the sum \n", sum);
7.printf("%7.2f is the average; ", average);
  printf("%8d is the sum \n", sum);
```

2.4.2 输入函数 scanf

使用scanf函数可以在程序执行时从键盘上输入数据。例如，假设一个程序要计算一段时间后新长出森林的英亩数，如果在程序中时间跨度是个常量，那么当需要改变时间时，就需要改变这个常量然后重新编译和执行，才能得到不同时期的输出值。这时使用输入函数就灵活得多，我们可以使用scanf函数来读取时间，而不用重新编译程序，只需要重新执行程序并输入想要计算的时间即可。

scanf函数的第一个参数是控制字符串，指定从键盘输入的变量的位数据类型。类型说明符在表2-7中给出，例如，integer型变量的说明符是%i或者%d，float型变量的说明符是%f、%e和%g，double型变量的说明符是%lf、%le和%lg。正确使用说明符是很重要的。例如，要读取double型的值，但是使用了说明符%f，这时就会发生错误。scanf函数中的其他参数是控制串中说明符所对应的存储单元，这些存储单元用地址运算符（address operator）&来表示。地址运算符是单目运算符，用于得出和它连在一起的变量标识符的内存地址。因此，如果从键盘上输入的是存储在整型变量year中的值，就可以使用以下语句来读取从键盘的输入：

47

```
scanf("%i",&year);
```

地址运算符的优先级和其他的单目运算符一样。如果一条语句中有好几个单目运算符，结合性为从右至左。scanf语句的常见错误是将变量标识符前的地址运算符&给丢了。

表2-7 输入语句的转换说明符

变量类型	说明符
整型值	
int	%i, %d
short	%hi, %hd
long int	%li, %ld
unsigned int	%u
unsigned short	%hu
unsigned long	%lu
浮点型值	
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG
字符型值	
char	%c

如果想要从键盘上读取多个数据，可以使用下面的语句：

```
scanf("%lf %c",&distance,&unit_length);
```

当这条语句被执行时，程序会从键盘上读取两个值，并将它们中的一个转换为double型，另一个转换为char型。当两个值从键盘输入时，中间至少要有一个空格将它们隔开。两个输入值可以在同一行，也可以在不同行。为了提示（prompt）程序使用者从键盘上输入值，一般在scanf语句前加一条printf语句，来描述用户要输入的信息。

```
printf("Enter the distance and the units (m for meters, f for "
       "feet): \n");
scanf("%lf %c",&distance,&unit_length);
```

printf语句中的控制字符串以换行符结束，所以用户输入的值会在提示语的下一行。在前一条语句执行后，用户要对提示语做出反应。这两条语句在屏幕上呈现的结果如下所示：

```
Enter the distance and the units (m for meters, f for feet):
10 m
```

如果用户输入的值不能转换成scanf语句中转换说明符所指定的类型，那么结果就是系统相关的。可能的转换错误包括，要输入整型的值，却输入了14.2，或者输入数字中夹杂有逗号，或者两个值之间没有空格。

虽然scanf函数的主要功能是读取从键盘中输入的值，它也可以返回输入值成功转换成指定数据类型的个数，这个返回值的具体使用方法会在后面的章节中讲到。

修改

写一个小程序，尝试向程序中加入以下错误，观察你的系统对这些错误会做出怎样的反应。

1. 0作为除数。
2. 输入转换错误：用1%说明符输入1 245，而不是1245。
3. 输入转换错误：用f%说明符输入一个整型变量。
4. 输入转换错误：用f%说明符输入一个double型变量。
5. 指数上溢错误：使用在本节中讨论的相关用例。
6. 指数下溢错误：使用在本节中讨论的相关用例。

2.5 解决应用问题：根据骨骼长度估算身高

本节中，将应用本章新学习到的语句来解决和法医人类学相关的问题。本章开头提到用骨骼残骸确定个人信息，现在来讨论如何用骨骼长度估算身高。

48

一个成人一般有206块骨头，最长的是股骨（在臀部和膝盖之间的大腿骨），最小的是中耳中的一块骨头，仅是手部就包含了54块骨头。股骨、胫骨（连接膝盖和踝关节的两块骨头中较长的那块）和肱骨（连接肩部和肘部的骨头）可以用来估计身高。骨头与骨头之间由韧带、肌腱、肌肉、软骨连接起来，组合在一起以保护人的器官，如心脏、肺和大脑。骨头大约占人体重的1/3。

股骨是人体中最大的一块骨头，它的长度可以用来估计人的身高，肱骨也可以用来估计身高。有许多利用骨头长度估计成人身高的方程式，这些方程式一般是先测量已知身高的人的骨头长度，然后找出这些数据的最小二乘线性模型（下章将具体介绍这种方法）。许多方程式会指定是用于推断女性身高，还是男性身高。下面是我们将会用到的方程式，这些方程式中骨头的长度单位是英寸[①]。

① 1英寸=0.025 4米

用股骨长度估计身高：

女性身高＝股骨长度 ×1.94+28.7

男性身高＝股骨长度 ×1.88+32

用肱骨长度估计身高：

女性身高＝肱骨长度 ×2.8+28.2

男性身高＝肱骨长度 ×2.9+27.9

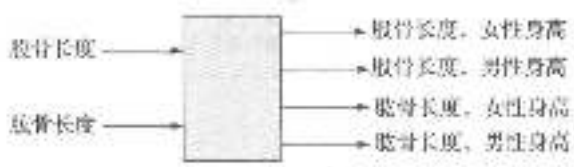
下面来设计一个C程序，让用户输入一个人的股骨长度和肱骨长度，程序将会分别用股骨长度和肱骨长度来计算出男性和女性的身高，并且输出计算值。

1. 问题陈述

分别用股骨长度和肱骨长度估算一个人的身高。

2. 输入/输出描述

下图表示出程序的输入是两块骨头的长度，输出高度由输入决定。因为不知道输入的骨头长度是女性的还是男性的，所以程序将同时估算男性和女性的身高。



49

3. 手动演算示例

如果股骨的长度是15in，肱骨的长度是12in，身高估算如下：

女性身高（股骨）＝股骨长度 ×1.94+28.7=57.8in[㊀]=4.82ft[㊁]=4ft9.8in

男性身高（股骨）＝股骨长度 ×1.88+32=60.2in=5.02ft=5ft24in

女性身高（肱骨）＝肱骨长度 ×2.8+28.2=61.8in=5.15ft=5ft1.8in

男性身高（肱骨）＝肱骨长度 ×2.9+27.9=62.7in=5.23ft=5ft2.76in

4. 算法设计

算法设计的第一步是将问题解决方案分解成一组可以程序执行的步骤。

分解提纲

1）读取股骨和肱骨的长度。

2）估算身高。

3）输出身高。

这个程序的结构很简单，可以将分解步骤直接转化为C程序。

```
/*-----------------------------------------------------------*/
/*   程序chapter2_2                                          */
/*                                                           */
/*   该程序通过肱骨长度和股骨长度估算人类身高                */
/*-----------------------------------------------------------*/

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

㊀ 1英寸(in)＝0.025 4米(m)

㊁ 1英尺(ft)＝0.304 8米(m)

```
int main(void)
{
   /*  声明变量  */
   double femur, femur_ht_f, femur_ht_m, humerus, humerus_ht_f,
   humerus_ht_m;

   /*  从键盘获取用户输入  */
   printf("Enter Values in Inches. \n");
   printf("Enter femur length: \n");
   scanf("%lf",&femur);
   printf("Enter humerus length: \n");
   scanf("%lf",&humerus);

   /*  计算身高估计值  */
   femur_ht_f = femur*1.94 + 28.7;
   femur_ht_m = femur*1.88 + 32;
   humerus_ht_f = humerus*2.8 + 28.2;
   humerus_ht_m = humerus*2.9 + 27.9;

   /*  打印身高估计值  */
   printf("\nHeight Estimates in Inches \n");
   printf("Femur Female Estimate:    %5.1f \n",femur_ht_f);
   printf("Femur Male Estimate:      %5.1f \n",femur_ht_m);
   printf("Humerus Female Estimate:  %5.1f \n",humerus_ht_f);
   printf("Humerus Male Estimate:    %5.1f \n",humerus_ht_m);

/*  退出程序  */
return 0;
}
/*-----------------------------------------------------------------*/
```

50

5. 测试

首先用手动演算示例中的数据来计算，下面是程序运行交互：

```
Enter Values in Inches.
Enter femur length:
15
Enter humerus length:
12

Height Estimates in Inches
Femur Female Estimate:        57.8
Femur Male Estimate:          60.2
Humerus Female Estimate:      61.8
Humerus Male Estimate:        62.7
```

程序计算结果和手动计算结果是一样的，所以接下来可以用其他数据来测试。如果程序计算结果和手动计算结果不同，那么就要检测是手动计算出错了还是程序出错了。

修改

下面的问题和本节由骨头长度估算人身高的程序有关。

1. 修改程序使得输出单位是英尺而不是英寸，输入单位还是英寸。输出结果为单位，如6.5ft。
2. 修改程序使得提示用户输入单位长度为英尺，然后在计算之前将输入值从英尺转换为以厘米为单位。输出值还是以英尺为单位，如6.5ft。
3. 修改程序使得程序读取以英寸为单位的值，分别输出以英尺和英寸为单位的两个值（注意，每一个身高估算都要有两个输出，以英尺为单位的输出和以英寸为单位的输出）。

51

4. 修改程序使得程序读取分别以英尺和英寸为单位的值，然后分别输出以英尺和英寸为单位的计算值

（注意，对于每一块骨头有两个输入值，对于每个身高估算也有两个输出值）。

5. 修改程序使得程序读取以厘米为单位的值，并且输出以厘米为单位的估算值（1in = 2.54cm）。

2.6 数值方法：线性插值

通过实验或者观察物理现象收集数据，是研究问题解决方法的重要的一环。我们通常假设输入数据和输出结果之间符合某个函数 $f(x)$，而观测到的数据就是对应函数曲线上的坐标点。对于一些不在原始数据集中的数据，把它当作函数 $f(x)$ 中的输入值 x，可以通过这些已有数据点来对 x 对应的函数值进行估算。例如，假设已有的数据点是 $(a, f(a))$ 和 $(c, f(c))$。如果想要估计 $f(b)$ 的值（$a<b<c$），我们可以使用线性插值（linear interpolation）法，假设 $f(a)$ 和 $f(c)$ 的坐标点之间通过一条直线连接，而 $f(b)$ 的坐标点就在这条直线上。如果假设点 $f(a)$ 和点 $f(c)$ 之间不是直线而是三次函数曲线，那么就可以使用三次样条插值法（cubic spline interpolation）来得到 $f(b)$ 的估计值。大多数的插值问题都可以采用这两种方法之一来解决。图 2-1 显示了包含 6 个点的数据集，它们分别通过直线段和三次曲线段连接。很显然，我们所选择的插值方法决定了采样点之间数据的估算结果。这一节我们主要讨论线性插值。

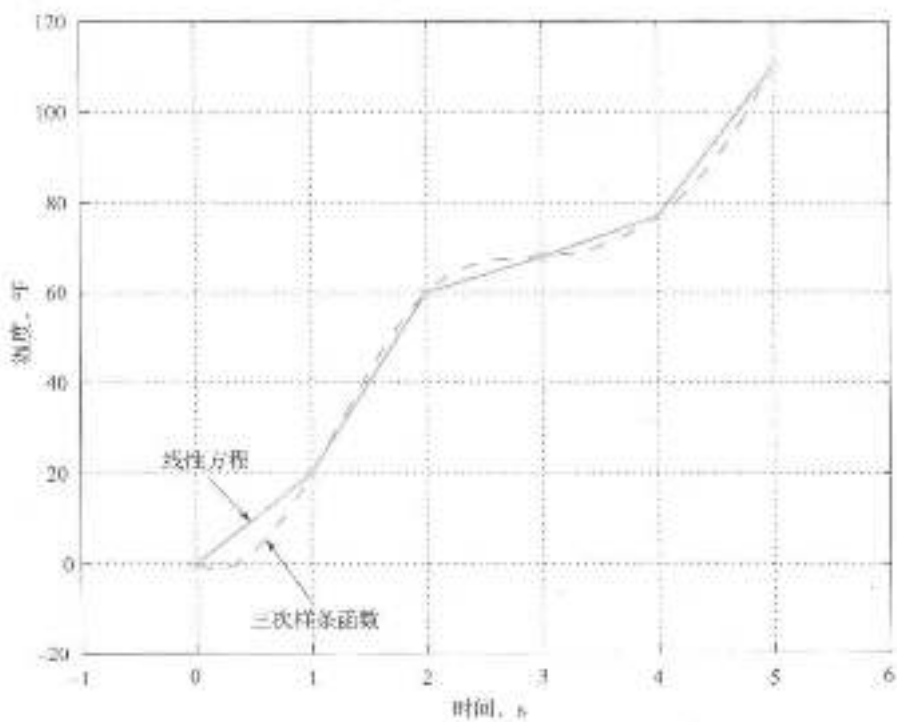


图 2-1 线性插值和三次样条插值

在图 2-2 中所示有两个数据点 $f(a)$ 和 $f(c)$。如果假设这两点间的函数关系是一条直线，那么就可以使用相似三角形的方程来计算出两点之间的任意一个 $f(b)$ 的值： 52

$$f(b) = f(a) + \frac{b-a}{c-a}[f(c) - f(a)]$$

这里还是假定 $a<b<c$。

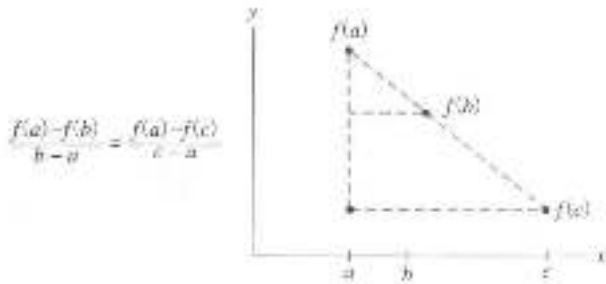


图 2-2 相似三角形

为了说明这种插值方程的使用方法，接着再展示下一个例子。假设有一台用在赛车上的发动机，在运行的不同时间从发动机汽缸盖上测量得到一组温度数据。温度测量数据表如下所示。这些数据通过直线段连接起来，如图 2-3 所示。

时间，s	温度，℉
0.0	0.0
1.0	20.0
2.0	60.0
3.0	68.0
4.0	77.0
5.0	110.0

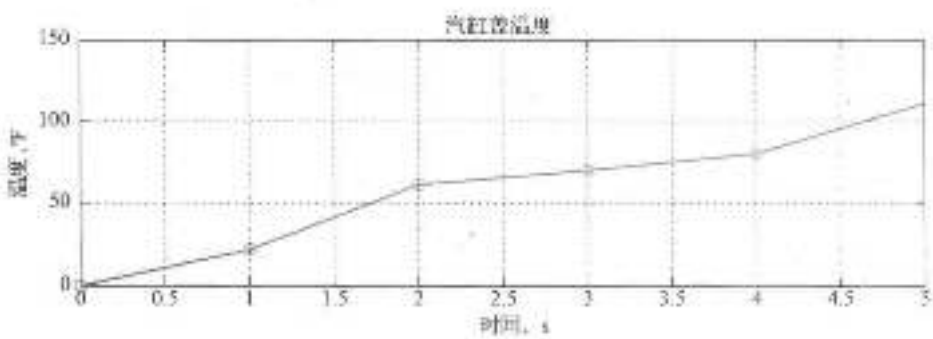


图 2-3 汽缸盖温度采集

假设现在要用插值法计算在 2.6s 时的温度值，那么代入刚才的方法中的情况如下：

a	2.0	60.0	$f(a)$
b	2.6	?	$f(b)$
c	3.0	68.0	$f(c)$

使用插值公式，可以得到

$$
\begin{aligned}
f(b) &= f(a) + \frac{b-a}{c-a}[f(c) - f(a)] \\
&= 60.0 + \frac{0.6}{1.0} \times 8.0 \\
&= 64.8
\end{aligned}
$$

53
~
54

在这个例子里，通过使用线性插值法来找到指定时间对应的温度值。我们同样也可以交换时间和温度的角色，令 x 轴表示温度，y 轴表示时间重新画出一条曲线。在这种情况下，

如果有一对已知数据点，其温度正好在指定温度的上下两侧，那么就可以用同样的过程来计算这个指定温度所发生的时间。

练习

假设现有如下数据集合，并且这些数据点都已绘制在图 2-4 中：

时间，s	温度，F
0.0	72.5
0.5	78.1
1.0	86.4
1.5	92.3
2.0	110.6
2.5	111.5
3.0	109.3
3.5	110.2
4.0	110.5
4.5	109.9
5.0	110.2

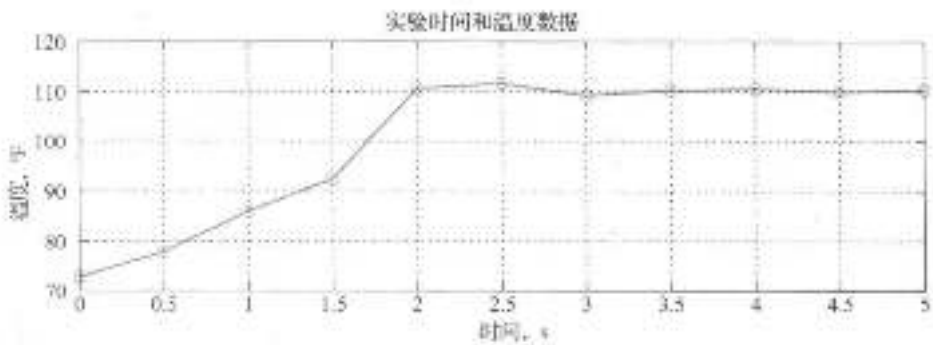


图 2-4 温度值

1. 通过线性插值法来计算以下时间点对应的温度值：

   0.3，1.25，2.36，4.48

2. 通过线性插值法来计算以下温度值所对应的时间值：

   81，96，100，106

3. 假设问题 2 中要计算的是温度 110°F 所对应的时间值。那么这个问题的难点在哪儿？在 110°F 处对应了几个时间点？使用线性插值法来找到每一个相应的时间值。（建议参考图 2-5，在该图中绘制了相关的数据点，x 轴代表温度，y 轴代表时间。）

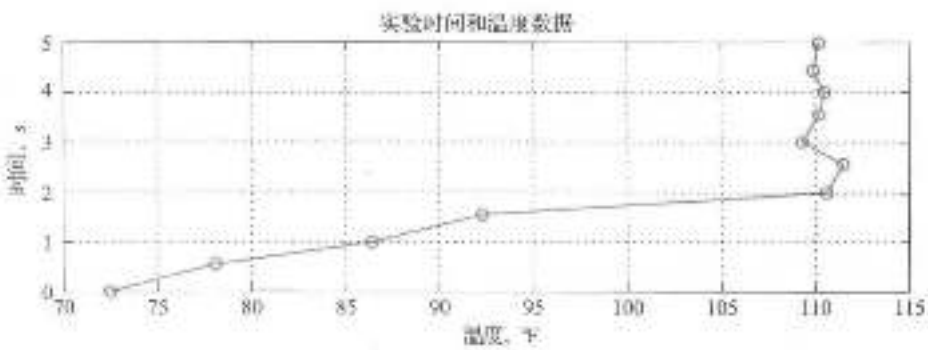


图 2-5

2.7 解决应用问题：海水的冰点

本章学习了一些算术运算的程序语句，并讨论了线性插值法，在这一节里将使用这些新方法来解决关于海水盐度（sulinity）的问题。

海水的盐度代表着海水中溶解物的数量。在海水中，大约有3.5%的溶解物（包含盐、金属和气体），而这些溶解物多是通过火山喷发和岩石风化而形成的，而海水的盐度就是用于表示这些溶解物的含量。海水溶解物的主要成分是氯离子（大约55%）和钠离子（大约30.6%），其余的成分里主要有硫酸根离子（7.7%）、镁离子（3.7%）、钙离子（1.2%）和钾离子（1.1%）。在整个海洋中盐度随着地区不同而变化，但基本上都处于千分之33到千分之38（ppt）之间，也就是3.3% ~ 3.8%之间。

通常采用检测水的导电性的仪器来测量盐度；水中溶解物越多，导电性就越好。检测海水盐度对严寒地区是很有必要的，因为这些地区的海水冻结温度严重依赖于其盐度的大小。下表包含了一组测量的盐度与相应的冻结温度的数据：

盐度（ppt）	冻结温度（℉）
0（纯水）	32
10	31.1
20	30.1
24.7	29.6
30	29.1
35	28.6

56

前面提供的数据分布如图2-6所示。

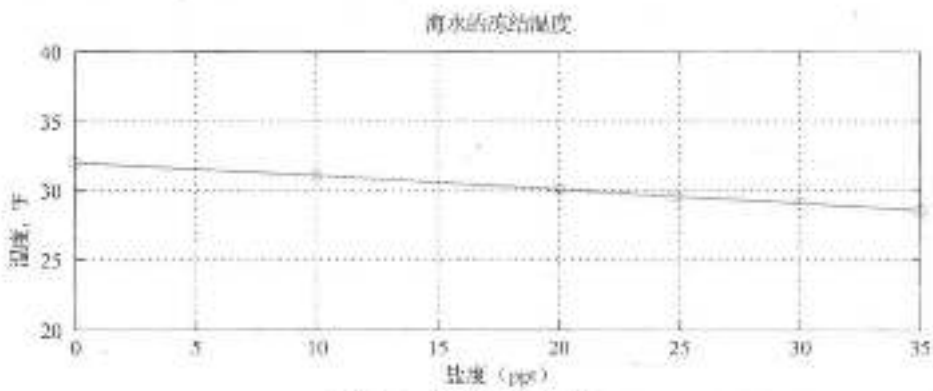


图2-6 海水的冻结温度

假设现在要用线性插值法来确定已测量了盐度值的水的冻结温度。编写这样一个程序，用户可以输入两组数据点（表示已知盐度值和相应的冻结温度）以及在这两点之间的一个盐度值，随后程序应该能计算出该盐度值对应的冻结温度。

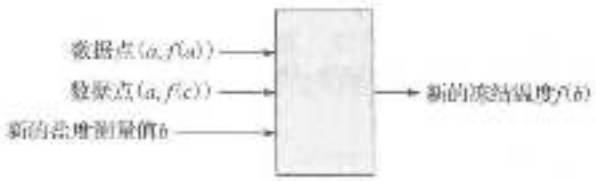
1. 问题陈述

用线性插值法来计算指定盐度对应的冻结温度值。

2. 输入/输出描述

下图展示了程序的输入：两组连续数据点 $(a, f(a))$ 和 $(c, f(c))$，以及一个新的盐度测量

值 b，输出则是相应的冰结温度：



57

3. 手动演算示例

假设现在要根据一个盐度测量值（33ppt）来确定相应的冰结温度。从已知数据可知该点落在 30ppt 和 35ppt 之间：

a	30	29.1	$f(a)$
b	33	?	$f(b)$
c	35	28.6	$f(c)$

使用线性插值公式，便可以计算 $f(b)$ 的值：

$$\begin{aligned} f(b) &= f(a) + (b - a)/(c - a)\cdot(f(c) - f(a)) \\ &= 29.1 + 3/5\cdot(28.6 - 29.1) \\ &= 28.8 \end{aligned}$$

正如之前所预期的，这个值刚好落在 $f(a)$ 和 $f(c)$ 之间。

4. 算法设计

设计算法首先要做的就是将问题的解决方法分解成一组连续的执行步骤：

分解提纲

1）接收相邻点的坐标以及新插入的盐度值。

2）计算新插入盐度值对应的冰结温度。

3）将计算出的冰结温度结果输出。

这个程序结构很简单，所以可以直接将分解步骤转换成 C 程序。

```
/*-----------------------------------------------------------------*/
/*   程序 chapter2_3                                               */
/*                                                                 */
/*   该程序使用线性插值计算海水的冰结温度值                          */

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  声明变量  */
   double a, f_a, b, f_b, c, f_c;

   /*  从键盘获取用户输入  */
   printf("Use ppt for salinity values. \n");
   printf("Use degrees F for temperatures. \n");
   printf("Enter first salinity and freezing temperature: \n");
   scanf("%lf %lf",&a,&f_a);
   printf("Enter second salinity and freezing temperature: \n");
   scanf("%lf %lf",&c,&f_c);
   printf("Enter new salinity: \n");
   scanf("%lf",&b);
```

58

```
    /*  使用线性插值计算新的冻结温度值  */
    f_b = f_a + (b-a)/(c-a)*(f_c - f_a);

    /*  打印新的冻结温度值  */
    printf("New freezing temperature in degrees F: %4.1f \n",f_b);

    /*  退出程序  */
    return 0;
}
/*------------------------------------------------------------------------*/
```

5. 测试

这里要通过示例中的数据来测试程序。测试过程如下：

```
Use ppt for salinity values.
Use degrees F for temperatures.
Enter first salinity and freezing temperature:
30 29.1
Enter second salinity and freezing temperature:
35 28.6
Enter new salinity:
33
New freezing temperature in degrees F: 28.8
```

这个计算结果同手动演算示例中的数据相符，所以接下来便可以用其他盐度值来测试程序。倘若新的输出值与手动演算的结果不符，那么就需要确认错误是发生在手动演算中还是在C程序中。

为了让线性插值法能正确执行，新的盐度测量值必须要介于输入的两组数据值之间。而在这个程序中，我们假定这个数据关系是确定的。在下一章里，将会学习使用新的C语言命令，来确保新的盐度测量值一定在第一、二组测量结果之间。

修改

本节通过设计程序，使用线性插值法计算新的冻结温度值。以下这些问题便是关于该程序的相关扩展。

1. 通过该程序来确定以下盐度值（ppt）对应的冻结温度值：

   3　　8.5　　19　　23.5　　26.8　　30.5

39 2. 修改程序，以使其得出的温度结果是用摄氏温度来表示。（$T_F = 9/5T_C + 32$，T_F 为华氏温度，T_C 为摄氏温度。）

3. 假设程序使用的数据中包含了摄氏温度表示的温度值，该程序需要修改吗？给出解释。

4. 将程序修改为通过插入温度值来求相应的盐度值。（可以参考图2-7中的曲线，x轴为温度值，y轴为盐度值。）

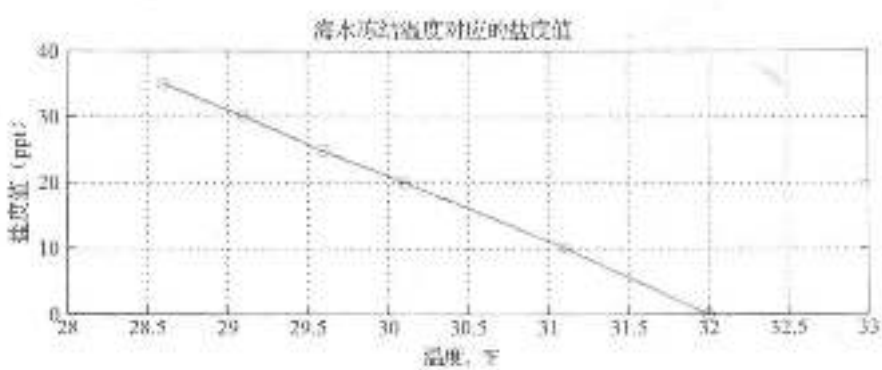


图2-7　海水冻结温度对应的盐度值

2.8 数学函数

在使用数学表达式解决工程问题的时候，除了需要加、减、乘、除以外，还会用到其他计算方法。例如，很多表达式会用到幂运算、对数运算、指数运算和三角函数等。在本节将讨论标准C语言库中的数学函数。在需要使用数学函数的程序中需要使用下面的预处理指令：

```
#include <math.h>
```

这条指令的意思是，该段源代码中包含对标准C语言库中的数学函数调用，当编译器处理这些函数的调用时，需要从math.h这个文件中获取函数定义等辅助信息。

在讨论相关函数的使用规则之前，这里先给出一个具体的例子。下面的式子是计算角theta的正弦值，并将结果存储在变量b中：

```
b = sin(theta);
```

这个sin函数本身是假设其参数是弧度。如果变量theta是一个角度值，那么可以用一个单独语句将角度值转换为弧度值（180° = π）。具体说明如下所示：

```
#define PI 3.141593
...
theta_rad = theta*PI/180;
b = sin(theta_rad);
```

这个转换操作同样也可以在如下的函数调用中进行说明：

```
b = sin(theta*PI/180);
```

60

人们通常更喜欢通过一个单独的说明语句来实现转换，因为这样更加便于理解。

一个函数调用，如sin(theta)，可以代表一个独立数值。函数名后面圆括号里的内容就是函数的输入值，通常被称为实参（parameter）或参数（argument）。一个函数可以包含一个或多个参数，或者没有参数，这取决于它的定义。如果一个函数包含多个参数，那么必须确保每个参数的顺序可以一一对应。同时，有些函数要求其参数必须是以特定单位出现。例如，三角函数就是假定它的参数是弧度值。大多数数学函数都假定其参数为double类型；如果需要用到其他类型的参数，那么在执行函数前一定要将其转换成double类型。

一个函数调用也可以作为另一个函数调用的参数。例如，下面的语句要计算x绝对值的对数值：

```
b = log(fabs(x));
```

当一个函数调用又作为一个函数的参数时，一定要确保每个函数的参数都是被该函数自己的括号围住。这种函数嵌套也称为函数的组合（composition）。

现在要讨论在工程计算中通常会用到的几类函数。其余的函数将会在余下章节里讨论相关课题的时候再进行介绍。在附录A中给出了标准C语言库中函数的更多信息。

2.8.1 基本数学函数

基本的数学函数（math function）中包括了一些常见的科学计算类函数，例如绝对值计算和开平方根等，也有一些函数专门用来实现四舍五入运算。这些数学函数假定其所有的参数都是double类型，并且都返回一个double类型的值；如果给定参数不是double类型，那么将会使用在2.3节介绍过的类型转换方法。现在将这些数学函数及其详细描述列出如下

所示：

fabs(x)	计算 x 的绝对值。
sqrt(x)	计算 x 的平方根，x ≥ 0。
pow(x,y)	进行指数运算，计算 x 的 y 次方的值，即 x^y。当 x=0 且 y ≤ 0 时，或者 x<0 且 y 不是整型时会报错。
ceil(x)	对 x 进行向上取整计算。例如，ceil(2.01) 的值为 3。
floor(x)	对 x 进行向下取整计算。例如，floor(2.01) 的值为 2。
exp(x)	计算 e^x 的值，其中 e 是自然对数的底数，约等于 2.718282。
log(x)	计算 lnx 的值，也就是以 e 为底，x 的自然对数。当 x ≤ 0 时会报错。
log10(x)	计算 $\log_{10}x$ 的值，也就是以 10 为底，x 的常用对数。当 x ≤ 0 时会报错。

61

需要指出的是，负数和零的对数是不存在的，所以如果执行了以一个非正数为参数的对数函数就会出现报错。

此外，还有一个非常有用的数学函数，叫作 abs 函数。这个函数计算一个整数的绝对值，然后返回一个整型值。包含相关信息的头文件是 stdlib.h，在需要调用该函数的程序中须包含此头文件。

练习

估算下列表达式的值：

1. floor(-2.6)
2. ceil(-2.6)
3. pow(2,-3)
4. sqrt(floor(10.7))
5. fabs(-10*2.5)
6. floor(ceil(10.8))
7. log10(100)+log10(0.001)
8. fabs(pow(-2,5))

2.8.2 三角函数

三角函数（trigonometric function）假定其所有参数都是 double 类型，并且函数返回值也是 double 类型。另外，正如之前所说，三角函数假设其角度都是用弧度表示。将弧度转换为角度，或者将角度转换为弧度，通常使用下面的语句：

```
#define PI 3.141593
...
angle_deg = angle_rad*(180/PI);
angle_rad = angle_deg*(PI/180);
```

三角函数被包含在标准 C 语言库中，要使用这些函数就要在程序中引入头文件 math.h，这样预处理指令就会在程序执行前从标准 C 语言库中提取信息。下面就是这些函数功能的简要介绍：

sin(x)	计算 x 的正弦值，其中 x 是弧度值。
cos(x)	计算 x 的余弦值，其中 x 是弧度值。
tan(x)	计算 x 的正切值，其中 x 是弧度值。
asin(x)	计算 x 的反正弦值，其中 x 的值域必须为 [-1，1]。函数返回值为一个弧度值，其值域为 [-π/2，π/2]。
acos(x)	计算 x 的反余弦值，其中 x 的值域必须为 [-1，1]。函数返回值为一个弧度值，其值域为 [0，π]。

atan(x)　　计算 x 的反正切值。函数返回一个弧度值，其值域为 [−π/2, π/2]。
atan2(y,x)　计算 y/x 的反正切值。函数返回一个弧度值，其值域为 [−π, π]。

62

需要注意的是，函数 atan 通常返回一个第一象限或第四象限的弧度值，但是函数 atan2 返回的角度值可能落在任何一个象限，这取决于 x 和 y 的符号。因此在很多应用中，函数 atan2 都比 atan 要更受欢迎。

其他的三角函数和反三角函数可以通过下面的公式来计算：

$$\sec x = \frac{1}{\cos x} \qquad \operatorname{asec} x = \operatorname{acos}\left(\frac{1}{x}\right)$$

$$\csc x = \frac{1}{\sin x} \qquad \operatorname{acsc} x = \operatorname{asin}\left(\frac{1}{x}\right)$$

$$\cot x = \frac{1}{\tan x} \qquad \operatorname{acot} x = \operatorname{acos}\left(\frac{x}{\sqrt{1+x^2}}\right)$$

使用三角函数时错用角度值来代替弧度值是程序中的一个常见错误。

练习

在问题 1 – 3 中，给出赋值语句来计算如下所示的值，并假定所有变量都已被声明，且已经赋了适当的值。同时假定程序中已作出如下声明：

```
#define g 9.8
#define PI 3.141593
```

1. 速度的计算：

$$\text{Velocity} = \sqrt{v_0^2 + 2a(x - x_0)}$$

2. 长度收缩：

$$\text{Length} = k\sqrt{1 - \left(\frac{v}{c}\right)^2}$$

3. 一个空心圆筒的重心到参考面的距离：

$$\text{Center} = \frac{38.1972(r^3 - s^3)\sin a}{(r^2 - s^2) \cdot a}$$

在问题 4 – 6 中，根据下面的赋值语句给出相应的等式方程。

4. 电子振荡频率：

```
frequency = 1/sqrt(2*pi*c/L);
```

5. 炮弹射程：

```
range = (v0*v0/g)*sin(2*theta);
```

6. 斜面底部的磁流速度：

```
v = sqrt(2*g*h/(1 + I/(m*pow(r,2))));
```

63

*2.8.3 双曲函数

双曲函数（hyperbolic function）是关于自然指数函数 e^x 的函数；而反双曲函数就是自然对数函数 $\ln x$ 的函数。在解决一些专门应用问题的时候这些函数非常有用，比如设计某些数

字滤波器。C语言中囊括了几种双曲线方程函数，下面便对这些函数做了简要介绍：

sinh(x)　　计算x的双曲正弦值，等价于$\frac{e^x - e^{-x}}{2}$

cosh(x)　　计算x的双曲余弦值，等价于$\frac{e^x + e^{-x}}{2}$

tanh(x)　　计算x的双曲正切值，等价于$\frac{\sinh x}{\cosh x}$

除此之外，还有一些双曲函数和反双曲函数可以通过以下公式计算得到：

$$\coth x = \frac{\cosh x}{\sinh x} \quad (x \neq 0)$$

$$\operatorname{sech} x = \frac{1}{\cosh x}$$

$$\operatorname{csch} x = \frac{1}{\sinh x}$$

$$\operatorname{asinh} x = \ln(x + \sqrt{x^2 + 1})$$

$$\operatorname{acosh} x = \ln(x + \sqrt{x^2 - 1}) \quad (x \geq 1)$$

$$\operatorname{atanh} x = \frac{1}{2}\ln\left(\frac{1 + x}{1 - x}\right) \quad (|x| < 1)$$

$$\operatorname{acoth} x = \frac{1}{2}\ln\left(\frac{x + 1}{x - 1}\right) \quad (|x| > 1)$$

$$\operatorname{asech} x = \ln\left(\frac{1 + \sqrt{1 - x^2}}{x}\right) \quad (0 < x \leq 1)$$

$$\operatorname{acsch} x = \ln\left(\frac{1}{x} + \frac{\sqrt{1 + x^2}}{|x|}\right) \quad (x \neq 0)$$

很多双曲函数和反三角函数在参数的取值范围上都有限制。如果是由用户从键盘手动输入参数值，请千万要注意参数取值范围的限制。在下一章里，即将介绍相关的C语句，可以让你确定一个值是否在程序中的适当范围内。

64

练习

写出赋值语句来计算以下式子的值，假设x的值已给出。(假定如下式子中的x值在适当的取值范围之内。)

1. coth x　　2. sec x

3. csc x　　4. acoth x

5. acosh x　　6. acsc x

2.9 字符函数

有很多函数可以用来处理字符型数据。有些函数负责字符的输入和输出，有些函数可以将字符在大小写之间转换，还有些函数能够进行字符比较。在本节将要讨论这些字符函数。

2.9.1 字符输入/输出

尽管printf和scanf函数可以通过使用%c说明符来打印和读取字符型数据，C语言

同样提供了供字符输入输出的专门函数。getchar 函数能够从键盘输入来读取下一个字符，并返回该字符的整型值；而 putchar 函数能够将字符打印输出至计算机屏幕。

putchar 函数接收整型参数，并返回一个整型值。函数执行之后，会将整型参数对应的字符输出至计算机屏幕上。如果一行中连续出现几个 putchar 函数调用，那么打印的字符结果也会在一行内逐一输出，直到有换行符出现。如上所述，下面的语句执行的结果就是字母 ab 打印在一行，紧接着字母 c 打印在新的一行：

```
putchar('a');
putchar('b');
putchar('\n');
putchar('c');
```

如果将函数参数替代成这些字母对应的整型值，那么执行结果会是同样的输出信息（如表 2-3 所示）：

```
putchar(97);
putchar(98);
putchar(10);
putchar(99);
```

getchar 函数从键盘输入来读取下一个字符，并且返回该字符对应的整型值。如果一行中几个 getchar 函数被连续调用，那么输入字符的处理会持续进行，直到接收到文字结束符 EOF 为止。EOF 是一个特殊值，它是头文件 stdio.h 的一个符号常量。考虑到用 char 类型定义的变量只能表示 128 个 ASCII 字符，而 EOF 代表的值超出了这个范围。又因为一个整型变量可以表示超过 128 个字符，所以这里使用 int 类型变量而不是 char 类型变量来接收 getchar 函数的返回值，这样就能够接收并存储 EOF 标识符，从而辨别到数据的结束。下面的语句执行后会从键盘输入读取一个字符，并且将其打印在新的一行：

65

```
int c;
...
putchar('\n');
c = getchar();
putchar(c);
putchar('\n');
```

文字结束符 EOF 的实际值是系统相关的。通常是 control-z 表示 EOF，也就是说，在文本编辑时输入 EOF 需要在键盘上首先按住 control 键（ctrl），同时再按住 z 键。这种字符组合常常也被记作 ^z（control-z），但并不是指同时按住 ^ 键和 z 键。EOF 的用法将在第 3 章详细介绍。

2.9.2 字符比较

标准 C 语言库还包含其他关于字符操作的函数。这些字符函数（character function）通常被分为两类：一类函数专门负责字符大小写之间的转换，另一类被用作字符比较。如果需要在程序中调用这些函数，那么将会用到下面的预处理命令：

```
#include <ctype.h>
```

下面的语句是将存储在变量 ch 中的小写字母转化为大写字母，而后将转换结果存储在变量 upper 中：

```
upper = toupper(ch);
```

如果 ch 是一个小写字母，那么函数 toupper 将会返回相应的大写字母；否则就返回 ch 本身，此时变量 ch 没有任何变化。

每个字符函数都要求其参数是整型的，并且都将返回一个整型值。其中，对于字符比较函数，如果比较结果为 true，则返回一个非零值；否则返回零。现将这些函数及其功能介绍如下：

tolower(ch)	如果 ch 是大写字母，该函数返回对应的小写字母；否则返回 ch 本身。
toupper(ch)	如果 ch 是小写字母，该函数返回对应的大写字母；否则返回 ch 本身。
isdigit(ch)	如果 ch 是十进制数，该函数返回一个非零值；否则返回零。
islower(ch)	如果 ch 是小写字母，该函数返回一个非零值；否则返回零。
isupper(ch)	如果 ch 是大写字母，该函数返回一个非零值；否则返回零。
isalpha(ch)	如果 ch 是大写字母或小写字母，该函数返回一个非零值；否则返回零。
isalnum(ch)	如果 ch 是一个字母字符或数字字符，该函数返回一个非零值；否则返回零。
iscntrl(ch)	如果 ch 是一个控制字符，该函数返回一个非零值；否则返回零。（控制字符（control character）的整型编码是 0 – 31 和 127。）
isgraph(ch)	如果 ch 是一个可打印字符（这是相对于不可打印的字符而言的，比如控制字符或 tab 字符），该函数返回一个非零值；否则返回零。（可打印字符的整型编码是 32 – 126。）
isprint(ch)	如果 ch 是一个可打印字符（包括空格字符），该函数返回一个非零值；否则返回零。
ispunct(ch)	如果 ch 是一个可打印字符（不包括空格、字母和数字），该函数返回一个非零值；否则返回零。
isspace(ch)	如果 ch 是空格、换页符、换行符、回车、水平制表符或垂直制表符（这些字符通常也被称为空白字符），该函数返回一个非零值；否则返回零。
isxdigit(ch)	如果 ch 是一个十六进制数，也就是说 ch 由一个十进制数或一个 A ~ F 的字母字符（或者 a ~ f）表示。该函数返回一个非零值；否则返回零。

66

练习

写出下列语句所产生的输出行。

```
1. putchar('x');
2. putchar(65);
3. putchar(tolower(65));
4. putchar('\n');
   putchar(97);
   putchar('c');
   putchar('\n');
   putchar(toupper('c'));
```

2.10 解决应用问题：速度计算

开式转子喷气发动机是一项具有很高的应用价值和广阔的市场前景的推进技术。例

如，测试程序已经表明这项推进技术使燃料消耗显著减少。然而在研发过程中同样遇到很多技术性挑战。例如，国家对商业引擎产生的噪声有严格的限制准则，因为机场附近往往是住宅区，所以需要增加额外的噪声处理措施。在进行实际飞行测验之前，需要在风洞中完成对比例模型和全尺寸发动机的测试。（有关风洞测试的问题分析会在本章的结尾进行讨论。）

67

在开式转子飞机的飞行测验里，试飞员首先将引擎功率推到 40 000N，使 20 000kg 的飞机获得 180m/s 的巡航速度。随后继续加大推力，将引擎功率推到 60 000N，这时飞机开始加速。随着飞机航速提升，空气阻力会与航速的平方成比例增加。最后飞机会达到一个新的巡航速度，此时的发动机推力与空气阻力相抵消。从推进引擎的油门被重新设定以加大推力开始，直到飞机达到一个新的航速，这段时间大约是 120s，此时的航速、加速度与该段时间的估算函数由如下的方程来描述：

$$\text{Velocity} = 0.00001\ \text{time}^3 - 0.00488\ \text{time}^2 + 0.75795\ \text{time} + 181.3566$$

$$\text{Acceleration} = 3 - 0.000062\ \text{velocity}^2$$

图 2-8 绘出了上述函数的曲线。应该注意的是，当飞机速度达到一个新航速的时候，加速度趋近于零。

编写一个程序，让用户输入一个时间值，该时间值代表从引擎功率提升开始经过的时间（以秒为单位）。计算并输出飞机在一个新时间点时相应的加速度和航速的值。

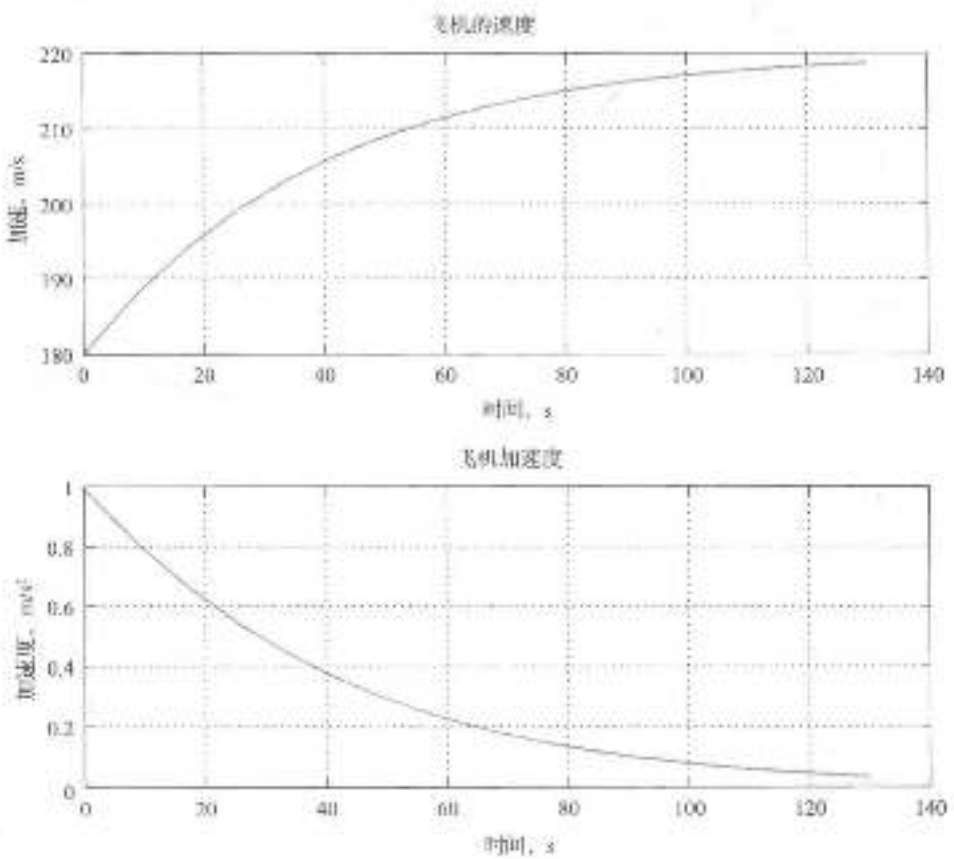


图 2-8 飞机的速度和加速度

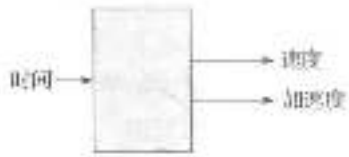
68

1. 问题陈述

计算飞机在发动引擎功率后的速度和加速度。

2. 输入/输出描述

如下图所示，程序的输入是一个时间值，程序输出是新产生的速度和加速度值：



3. 手动演算示例

假设新的时间值为 50 秒。使用速度和加速度的关系式，可以计算出所求值：

$$\text{Velocity} = 208.3\ \text{m/s}$$
$$\text{Acceleration} = 0.31\ \text{m/s}^2$$

4. 算法设计

在设计算法时，首先要做的是将问题解决方案分解成一系列连续的执行步骤：

分解提纲

1）读取新的时间值。

2）计算相应的速度和加速度值。

3）输出新的速度和加速度值。

由于这是个比较简单的程序，因此可以将分解步骤直接转化成 C 程序语句：

```
/*-----------------------------------------------------------------*/
/*   程序 chapter2_4                                               */
/*                                                                 */
/*   本程序估算一个指定时间点上的速度值和加速度值                  */

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  变量声明  */
   double time, velocity, acceleration;
   /*  从键盘输入一个时间值  */
   printf("Enter new time value in seconds: \n");
   scanf("%lf",&time);

   /*  计算速度和加速度  */
   velocity = 0.00001*pow(time,3) - 0.00488*pow(time,2)
              + 0.75795*time + 181.3566;
   acceleration = 3 - 0.000062*velocity*velocity;

   /*  输出速度和加速度值  */
   printf("Velocity = %8.3f m/s \n",velocity);
   printf("Acceleration = %8.3f m/s^2 \n",acceleration);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

5. 测试

首先使用手动演算示例中提供的数据来测试程序。程序执行产生如下交互：

```
Enter new time value in seconds:
50
Velocity =  208.304 m/s
Acceleration =  0.310 m/s^2
```

因为计算得到的值与手动演算示例中的数据相符，所以接下来便可以用其他时间值来测试程序。如果计算值与手动演算数据不符，那么应该确认错误是出现在手动演算数据上还是出现在程序中。

修改

本节主要讨论了关于计算速度和加速度的问题，而下面这些问题是由此延伸而来：

1. 连续输入不同的时间值，直到取得一个在 210m/s – 211m/s 之间的速度值为止。
2. 连续输入不同的时间值，直到取得一个在 $0.5m/s^2$ – $0.6m/s^2$ 之间的加速度值为止。
3. 修改程序，使得输入的时间单位从秒变为分钟，同时要注意，方程中的时间单位仍为秒。
4. 修改程序，使得输出值的单位变为英尺/秒和英尺/秒^2（1 米 = 39.37 英寸）。

70

2.11 系统边界

2.2 节提供了一个表格，里面给出了整型变量和浮点数变量的最大值（在 Microsoft Visual C++ 2010 Express compiler 中）。通过使用下面的程序，也能得到类似的针对自己系统的表格。注意，该程序包含 3 个头文件。其中，因为程序要调用输出函数，所以一定要包含头文件 stdio.h；因为程序需要使用整型变量范围的相关信息，所以头文件 limits.h 是必需的；因为程序需要用到浮点类型变量范围的相关信息，所以头文件 float.h 也是必需的。附录 A 包含了更多有关常量和边界的内容，并且这些特性是系统相关的。

```
/*-----------------------------------------------------------*/
/*  程序 chapter2_5                                          */
/*                                                           */
/*  本程序输出系统各种限制值                                 */

#include <stdio.h>
#include <limits.h>
#include <float.h>

int main(void)
{
   /*  打印类型的最大值  */
   printf("short maximum: %i \n",SHRT_MAX);
   printf("int maximum: %i \n",INT_MAX);
   printf("long maximum: %li \n\n",LONG_MAX);

   /*  打印浮点型的精确度、范围和最大值  */
   printf("float precision digits: %i \n",FLT_DIG);
   printf("float maximum exponent: %i \n",
          FLT_MAX_10_EXP);
   printf("float maximum: %e \n\n",FLT_MAX);

   /*  打印double型的精确度、范围和最大值  */
```

```
    printf("double precision digits: %i \n",DBL_DIG);
    printf("double maximum exponent: %i \n",
           DBL_MAX_10_EXP);
    printf("double maximum: %e \n\n",DBL_MAX);

    /*  打印long型的精确度、范围和最大值  */
    printf("long double precision digits: %i \n",LDBL_DIG);
    printf("long double maximum exponent: %i \n",
           LDBL_MAX_10_EXP);
    printf("long double maximum: %Le \n\n",LDBL_MAX);

    /*  程序结束  */
    return 0;
}
/*-----------------------------------------------------------*/
```

71

修改

在运行过该程序后，改变转换说明符，使下面这些数值按全精度打印，而非默认的6位数字精度。

1. 浮点型最大值　　2. 双精度型最大值　　3. 长双精度型最大值

本章小结

在本章中，我们提供了一些C语句，并实现了几个计算和打印数值的简单程序。与此同时，还给程实现了在程序运行时读取键盘上输入的信息。本章所提出的计算包括标准算术运算和大量函数，它们可以解决实际工程所需的各种类型的计算问题。同时，我们还对线性插值法进行了详细讨论并给出了程序用例。

关键术语

abbreviated assignment（缩写赋值）
address operator（地址运算符）
argument（参数）
ASCII code（ASCII码）
assignment statement（赋值语句）
associativity（结合性）
binary code（二进制码）
binary operator（二进制运算符）
case sensitive（大小写敏感）
cast operator（转换操作符）
character（字符）
character function（字符函数）
comment（注释）
composition（复合）
constant（常量）
control character（控制符）
control string（控制字符串）
initial value（初始值）
keyword（关键字）
left justify（左对齐）
linear interpolation（线性插值）
math function（数学函数）
memory snapshot（内存快照）
mixed operation（混合运算）
modulus（取模）
multiple assignment（多重赋值）
overflow（溢出）
parameter（形参）
postfix（后缀）
precedence（优先级）
precision（精度）
prefix（前缀）
preprocessor directive（预处理命令）
prompt（提示）

conversion specifier（转换说明符）
declaration（声明）
EBCDIC code（扩展二进制编码）
EOF character（文字结束符）
escape character（转义字符）
exponential notation（指数计数法）
expression（表达式）
field width（字段宽度）
floating-point value（浮点值）
garbage value（垃圾代码）
hyperbolic function（双曲函数）
identifier（标识符）
right justify（右对齐）
scientific notation（科学计数法）
Standard C library（标准C库）
statement（语句）
symbolic constant（符号常量）
system dependent（系统相关的）
trigonometric function（三角函数）
truncate（截断）
type specifier（类型说明符）
unary operator（单目运算符）
underflow（下溢）
variable（变量）

72

C语句总结

通过预处理命令来提供标准C库中相关文件的信息：

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

通过预处理命令来定义一个符号常量：

```
#define PI 3.141593
```

整型变量的声明：

```
short sum=0;
int year_1, year_2;
long k;
```

浮点型变量的声明：

```
float height_1, height_2;
double length=10, side1, side2;
long double distance, velocity;
```

赋值语句：

```
area = 0.5*base*(height_1 + height_2);
```

键盘输入语句：

```
scanf("%i",&year);
```

屏幕输出语句：

```
printf("The area is %f square feet. \n",area);
```

程序结束语句：

```
return 0;
```

从键盘输入读取一个字符的指令：

```
c = getchar();
```

输出一个字符到屏幕的指令：

```
putchar(c);
```

注意事项

1. 在程序中添加注释可以提高程序的可读性，同时也能记录程序执行的每一步。
2. 使用适当的空行和缩进来标识一个程序的结构。
3. 在可能的情况下变量名中应包含使用的单位。
4. 对于工程常数应该使用符号常量来标识，比如 π，同时还应将符号常量大写，以便于识别。
5. 在算术运算符和赋值运算符的两侧使用空格，且这种使用空格的风格应保持一致。
6. 在复杂的表达式中使用圆括号来提高可读性。
7. 对于长表达式应该将其分解成几条语句。
8. 在程序输出中要确保所有的数值都带有单位。
9. 从键盘输入数据时，要向用户显示提示信息，通知用户输入值和单位。

73

调试注意事项

1. 声明语句和 C 语句的结尾一定要加分号。
2. 预处理命令后面不加分号。
3. 如果可能，要避免可能会引起信息丢失的赋值。
4. 在长语句中加圆括号的时候要确保其语义不会发生变化。
5. 使用双精度或扩展精度来避免出现指数上溢或下溢。
6. 在 scanf 语句中要确保说明符与变量类型是匹配的。
7. 在 scanf 语句中，如果用户输入的变量类型不能正确转换成说明符的变量类型时就会报错。
8. 不要忘了在 scanf 语句中的标识符前加上地址运算符。
9. 在定义符号常量的时候后面不要加分号。
10. 在函数嵌套调用的时候，每个函数的参数一定要包含在对应函数的括号里。
11. 对数函数的参数一定不能为负数。
12. 在三角函数中一定要使用弧度角。
13. 要注意在很多反三角函数和双曲函数里，对输入值的取值范围是有规定的。
14. 数字和字符数字的整数表示是不同的。
15. 将 getchar 函数返回的结果保存在整型变量中，这样返回值是 EOF 时才不会出错。

习题

简述题

判断题

判断下面陈述的正（T）误（F）。

1. 一个程序是从 main 函数开始执行的。 T F
2. C 语言对大小写并不敏感。 T F
3. 声明语句可以放在程序中的任何位置。 T F
4. 普通语句和声明语句都必须以分号结尾。 T F

5. 整数除法的结果是一个近似结果。 T F

语法题

判断下面的声明语句的语法是否正确。如果不正确，请改正。

6. `int i, j, k,`

7. `float f1=11, f2=202.00;`

8. `DOUBLE D1, D2, D3;` 74

9. `float a1-a2;`

10. `int n, n_m;`

多选题

在下列题目中选择最佳答案。

11. 下列（　　）不是C语言关键字。

(a) const　　(b) goto

(c) static　　(d) when

(e) unsigned

12. 在声明语句中，类型说明符和变量名之间是被（　　）分隔开的。

(a) 句号　　(b) 空格

(c) 等号　　(d) 分号

(e) 以上都不是

13. 以下（　　）声明语句能将x、y和z正确定义成double型变量。

(a) `double x, y, z;`　　(b) `long double x, y, z;`

(c) `double x, y, z;`　　(d) `double x=y=z;`

(e) `double X, Y, Z;`

14. 在C语言中，双目运算符%是用来计算（　　）。

(a) 整数除法　　(b) 浮点数除法

(c) 整数除法的余数　　(d) 浮点数除法的余数

(e) 以上都不是

15. 下列（　　）赋值语句的结果为零。

(a) `result = 9%3 - 1;`　　(b) `result = 8%3 - 1;`

(c) `result = 2 - 5%2;`　　(d) `result = 2 - 6%2;`

(e) `result = 2 - 8%3;`

内存快照题

请写出在下列每组语句执行后相应的内存快照。

16.
```
int x1;
...
x1 = 3 + 4%5    5;
```

17.
```
double a=3.8, x;
int n=2, y;
...
x = (y = a/n)*2;
```
75

程序输出题

请写出下列语句执行后的结果：

```
18. float value_1=5.78263;
    ...
    printf ("value_1 = %5.3f",value_1);
19. double value_4=66.45832;
    ...
    printf ("value 4 = %10.2e",value 4);
20. int value_5=7750;
    ...
    printf ("value_5 = % + 6d",value_5);
```

编程题

转换问题。这组问题是关于将一个值从一个单位转换为另一个单位。每个程序应该提示给用户一个带有指定单位的值，随后打印出转换后的值，并带有新的单位。

21. 编写程序，将英里转换为千米。（1mile = 1.609 3440km）
22. 编写程序，将米转换为英里。（1mile = 1.609 3440km）
23. 编写程序，将磅转换为千克。（1kg = 2.205lb）
24. 编写程序，将牛顿转换为磅。（1lb = 4.448N）
25. 编写程序，将华氏温度（T_F）转换为朗肯温度（T_R）。（$T_F = T_R - 459.67°R$）
26. 编写程序，将摄氏温度（T_C）转换为朗肯温度（T_R）。（$T_F = T_R - 459.67°R$、$T_F = (9/5)\,T_C + 32°F$）
27. 编写程序，将开尔文温度（T_K）转换为华氏温度（T_F）。（$T_K = (9/5)\,T_K$，$T_F = T_R - 459.67°R$）

面积和体积问题。下面这些问题是关于根据用户的输入来计算面积或体积。每个程序首先提示用户输入所需的变量值。

28. 编写程序，计算一个长、宽为 a 和 b 的矩形的面积。（$A = a \times b$）
29. 编写程序，计算一个底为 b，高为 h 的三角形的面积。（$A = 1/2\,(b \times h)$）
30. 编写程序，计算一个半径为 r 的圆的面积。（$A = \pi r^2$）
31. 编写程序，计算一个扇形的面积，其中扇形的角度 θ 用弧度表示。（$A = r^2\theta/2$，其中 θ 为弧度角）
32. 编写程序，计算一个扇形的面积，其中扇形的角度 d 用角度表示。（$A = r^2\theta/2$，其中 θ 为弧度角）
33. 编写程序，计算一个椭圆的面积，其中半轴长为 a 和 b。（$A = \pi a \times b$）
34. 编写程序，计算一个半径为 r 的球面面积。（$A = 4\pi r^2$）
35. 编写程序，计算一个半径为 r 的球的体积。（$V = (4/3)\pi r^2$）
36. 编写程序，计算一个底面半径为 r，高为 h 的圆柱体体积。（$V = \pi r^2 h$）

计算氨基酸的分子质量。如表2-8所示，蛋白质中的氨基酸由氧原子、碳原子、氮原子、硫原子和氢原子组成，其中各个元素的原子量如下所示：

元素	原子量
氧元素	15.999 4
碳元素	12.011
氮元素	14.006 74
硫元素	32.066
氢元素	1.007 94

76

表 2-8 氨基酸分子

氨基酸	O	C	N	S	H
丙氨酸	2	3	1	0	7
精氨酸	2	6	4	0	15
天冬酰胺酸	3	4	2	0	8
天冬氨酸	4	4	1	0	6
半胱氨酸	2	3	1	1	7
谷氨酸	4	5	1	0	8
谷氨酰胺	3	5	2	0	10
甘氨酸	2	2	1	0	5
组氨酸	2	6	3	0	10
异亮氨酸	2	6	1	0	13
亮氨酸	2	6	1	0	13
赖氨酸	2	6	2	0	15
蛋氨酸	2	5	1	1	11
苯丙氨酸	2	9	1	0	11
脯氨酸	2	5	1	0	10
丝氨酸	3	3	1	0	7
苏氨酸	3	4	1	0	9
色氨酸	2	11	2	0	11
酪氨酸	3	9	1	0	11
缬氨酸	2	5	1	0	11

37. 编写程序，计算甘氨酸的分子量并打印结果。

38. 编写程序，计算谷氨酸和谷氨酰胺的分子量，并打印结果。

39. 编写程序，让用户分别输入某种氨基酸中 5 种元素的原子个数，然后计算出这种氨基酸的分子量，并打印结果。

40. 编写程序，让用户分别输入某种氨基酸中 5 种元素的原子个数，然后计算出这种氨基酸的平均原子量，并打印结果。

计算以 b 为底的对数值。为了计算以 b 为底 x 的对数值，可以使用如下关系式：

$$\log_b x = \frac{\log_a x}{\log_a b}$$

41. 编写程序，先读取一个正数，然后计算出其以 2 为底的对数值，并打印结果。例如，以 2 为底 8 的对数值是 3，因为有 $2^3 = 8$。

42. 编写程序，先读取一个整数，然后计算出其以 8 为底的对数值，并打印结果。例如，以 8 为底 64 的对数值为 2，因为有 $8^2 = 64$。

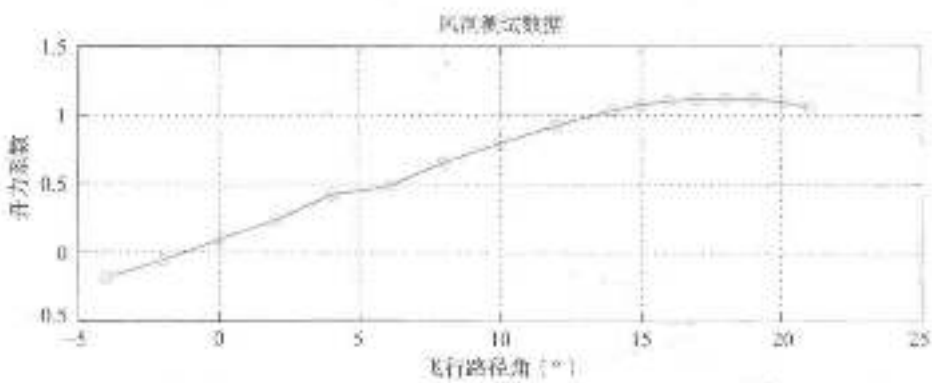
风洞。风洞是指一个可以产生不同风速或马赫数的试验室（马赫是风速除以音速）。精确的飞机比例模型可以被安装在支持压力测试的试验室中，并且以多种不同的风速和角度来对模型进行压力测试。在一个扩展风洞测试结束后，通过使用测试中收集的大量数据，我们可以确定一架新机型在不同航速和位置下的各种空气动力学性能表现的系数，比如升力和阻力等。图 2-9 中展示了一个风洞试验中收集的数据分布情况，其具体试验数据如下表所示：

飞行路径角（°）	升力系数
-4	-0.182
-2	-0.056
0	0.097
2	0.238
4	0.421
6	0.479
8	0.654
10	0.792
12	0.924
14	1.035
15	1.076
16	1.103
17	1.120
18	1.121
19	1.121
20	1.099
21	1.059

78 43. 假设现在要使用线性插值法来确定飞行路径角在 $-4° \sim 21°$ 之间的升力系数。编写程序，让用户自行输入四对数据点和四对点之间的飞行路径角，然后计算出相应的升力系数。

44. 修改43题中的程序，使其打印出的飞行路径角为弧度角。其中输入值也应该满足弧度角的表示范围。（$180° = \pi$）

45. 修改43题中的程序，使其插入一个新飞行路径角，而不是一个新系数。因此，用户需要输入四对数据点和四对点之间某个点的升力系数，然后计算出该点的飞行路径角，结果用角度值表示。



79 图2-9 风洞测试数据

控制结构和数据文件

犯罪现场调查：人脸识别与监控视频

人脸识别是一种常用的身份识别技术，它可以通过监控视频中的一幅图片或者一个单帧来辨别出一个人的脸部特征，从而实现身份识别。但是人脸识别技术并不能达到指纹识别那样的精确度。例如，在机场安全区域设防时，人脸识别通常会和其他的生物识别技术或者一些安全措施一同综合使用。例如，在安保系统中，首先对一张影像图片进行分析检测，在主数据库中选取三张匹配度较高的图片，然后提供给安保人员进行选择，最后由安保人员来决定发出进入请求的人是否具有进入权限。目前已经有很多商业应用在使用人脸识别。谷歌的 Picasa 数字图片管理器可以使用人脸识别来检索图片，从一组图片中找出与特定图片相匹配的人脸。当然还有实现一些使用人脸识别的应用，包括苹果公司的 iPhoto（一款照片管理器）、索尼公司的 Picture Motion Browser（为具有相同面部的图片加标签）以及 Facebook。在安全领域，人脸识别常常应用于解决犯罪问题和辨认恐怖分子等。人脸识别技术曾经从机场的监控视频中成功辨认出了一些 9·11 恐怖分子。在 2005 年的伦敦地铁爆炸事件中，这项技术同样帮助警方辨别、锁定了炸弹投放者。2001 年 1 月在美国佛罗里达州坦帕湾举行的第 35 届超级碗大赛中，人脸识别技术可谓起到了重要的作用，系统通过监控摄像头进行检测，在观众中识别出具有犯罪记录的人，并且有 19 名具有轻微犯罪背景的观众被认定为存在嫌疑。在本章后面的内容中，我们将对一种经典的面部识别算法进行详细讨论，并编写 C 程序实现对两张人脸图片的信息比对，从而确定它们是否来自同一个人。

学习目标

在本章，我们将学到以下解决问题的方法：

- 选择结构，允许我们在程序中提供几条可供选择的执行路径。
- 循环结构，只要条件成立就重复执行一系列步骤。
- 使用从数据文件中读取的信息和被写入数据文件的信息。
- 线性建模方法。

3.1 算法开发

第 2 章开发的 C 程序比较简单。程序步骤都是顺序执行的，而且通常是通过键盘输入来读取信息，计算出一个新结果并打印在屏幕上。而在解决工程问题时，绝大多数的解决方案都包含非常复杂的执行步骤，因此在解决问题的过程中，需要对算法开发的部分加以延伸和扩展。

3.1.1 自顶向下设计

自顶向下设计（top-down design）以顺序的方式给出了问题解决方案的宏观描述。然后我们对这种全局描述不断地重新定义、细化，直到所有步骤可以详细到能够转化为程序语句为止。

1. 分解提纲

在第1章和第2章里，通过使用分解提纲（decomposition outline）来得到一个初步的解决方案。分解得到几个顺序的执行步骤，可以逐条写出提纲，也可以画成简易流程图。对于那些比较简单的问题，就像第2章中介绍的那样，我们可以将问题的分解提纲直接转化为C语句：

81

分解提纲：

1）读取一个新的时间值。

2）计算相应的速度和加速度值。

3）将得出的速度和加速度结果打印到屏幕上。

然而对于绝大多数问题解决方案，我们需要将分解提纲细化成更细节的描述。这个过程又称为分治（divide-and-conquer）策略，我们希望将问题的解决方案不断分解、细化成越来越小的子问题然后逐一解决掉。我们常常会使用伪代码或流程图来描述这种逐步细化（stepwise refinement）的过程。

2. 用伪代码和流程图细化问题

我们使用伪代码或流程图来完成提纲到具体步骤的细化过程。伪代码（pseudocode）用类似人类语言的语句描述算法中的每个步骤，而流程图（flowchart）是用图示描述算法步骤。图3-1中展示了大多数算法中使用的基本步骤，同时给出了相应的伪代码表示法和流程图表示法。

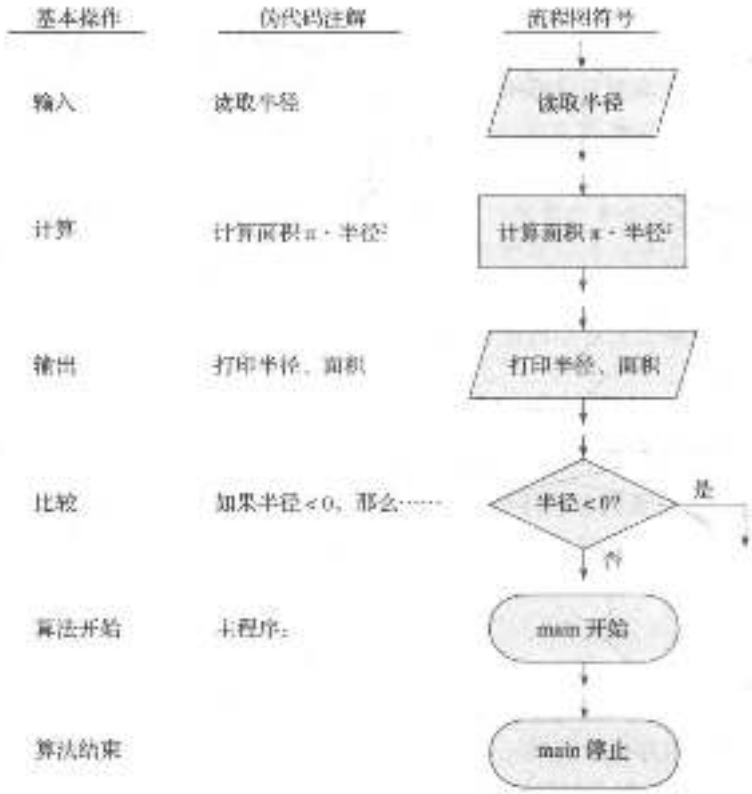


图3-1 伪代码注解和流程图符号

在解决问题时，常常通过伪代码和流程图来确定执行的步骤。这两种工具都很常用，但是在同一个问题中一般不同时使用。为了给出这两种工具的例子，下面的问题解决方案有些会使用伪代码，有些会使用流程图；至于这两种方法如何选择，通常是看个人习惯。在开发复杂问题的解决方案时，常常需要完成几个层次的伪代码和流程图，这就是我们之前讨论过的逐步细化的过程。分解提纲、伪代码和流程图都是对问题解决方案的模型化描述方法，因此并不唯一。在解决问题时，每个人都会设计出不同的分解提纲、不同的伪代码和流程图，而开发出的C程序因编程者而不尽相同，虽然他们解决的都是同一个问题。

3.1.2 结构化编程

结构化程序（structured program）是用简单控制结构组织问题解决方案的程序。简单结构通常分为三种：顺序结构、选择结构和循环结构。在顺序（sequence）结构中，代码按照编写的顺序来执行；在选择（selection）结构中，当条件为true时，会执行一组代码，当条件为false时，便执行另一组代码；在循环（repetition）结构中，只要条件为true，程序会不断地重复执行。下面将会对这三种简单结构一一进行讨论，并通过伪代码和流程图来举例说明。

1. 顺序结构

在顺序结构中，代码会依次按顺序执行。第2章涉及的程序都属于顺序结构。例如，线性插值法的程序伪代码如下：

```
[读取后的伪代码]
主函数：  read a, f_a
          read c, f_c
          read b
          set f_b to f_a + (b − a)/(c − a) · (f_c − f_a)
          print f_b
```

图3-2展示了计算开式转子发动机飞机的速度和加速度的程序的流程图。

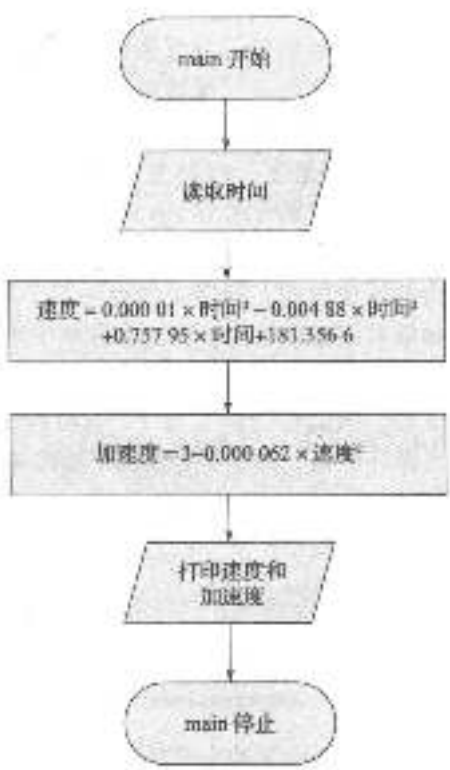


图3-2 2.10节中开式转子发动机飞机问题解决方案的流程图

2. 选择结构

在选择结构中包含了一个可以判断为true或false的条件（condition）语句。如果条件为true，则会立即执行一组语句；如果条件为false，则会执行另一组语句。例如，假设已经得到了一个分数的分子和分母的计算值，那么在计算分数值（进行除法）之前，需要确定分母一定不是个接近于0的数。因此判断条件就是来检验“分母是接近于0的数”。如果条件判断为true，那么就在屏幕上打印消息“无法计算出该分数值”。如果条件为false，也就是说分母并不是接近0的数，那么就计算并打印出该分数值。为了定义判定条件，需要定义“接近于0”。对于这个例子，我们假设“接近于0”表示绝对值小于0.000 1。下面给出了相应的伪代码描述：

82 ~ 83

```
if |denominator| < 0.0001
        print "Denominator close to zero"
else
        set fraction to numerator/denominator
        print fraction
```

图 3-3 展示了该选择结构的流程图。需要注意的是，在选择结构中同样也包含了顺序结构（即计算并打印分数值这部分），当条件判定为 false 时，便会执行这部分顺序结构的代码。本章后面将会介绍更多选择结构的变式。

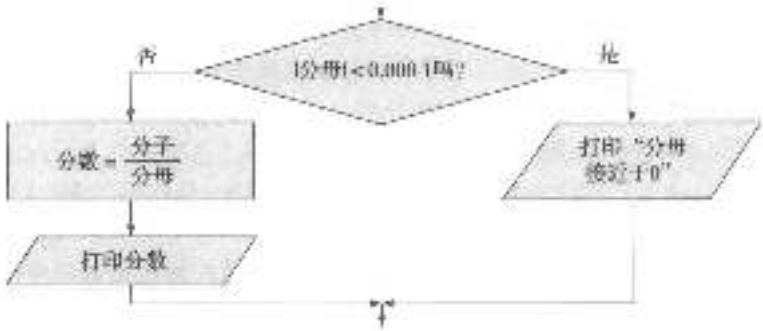


图 3-3 选择结构的流程图

3. 循环结构

在循环结构中，当条件判定为 true 时，便重复执行（或依次重复（loop）执行）一组代码。例如，给出一组时间值 0，1，2，…，10s，要计算相应的一组速度值。如果要设计一个顺序结构的程序，那么它的语句就是要依次计算时间为 0s 时的速度，时间为 1s 时的速度，时间为 2s 时的速度……直到时间为 10s 时的速度。虽然这个例子中只需要执行 11 条计算语句，但是如果要计算一大段时间内的速度值时，程序中就会一下子需要上百条语句了。如果采用循环结构，那么在程序中可以设定初始时间为 0，只要时间值小于等于 10，便计算并打印出相应的速度值，同时将时间自增 1。当时间值大于 10 时，便退出循环结构。图 3-4 是该循环结构的流程图，其伪代码描述如下所示：

84

```
set time to 0
while time ≤ 10
        compute velocity
        print velocity
        increment time by 1
```

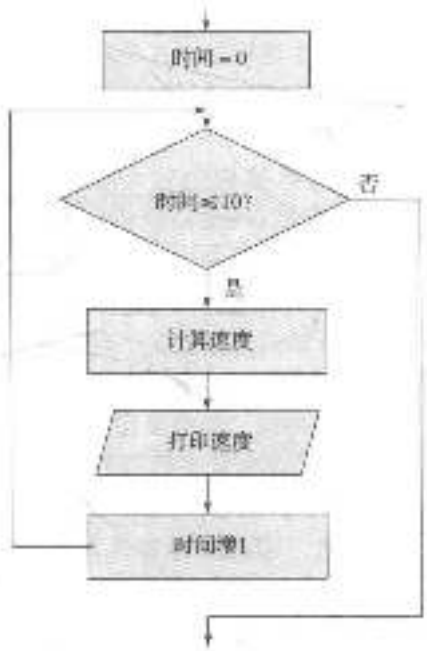


图 3-4 循环结构的流程图

在本章余下的小节里，将会介绍专门执行选择和循环操作的 C 语句，并且还会使用这些语句来开发示例程序。

3.1.3 多种解决方案评估

同一个问题可以具有多种解决思路。大多数情况下，虽然众多解决方案之中各有优劣，但是最佳方案往往不是唯一的。随着经验的增长，选出最优方案的过程变得更加轻松。要设计出好的解决方案需要很多要素，本书将对一些要素进行举例阐述。例如，一个好的解决方案必须具备良好的可读性；而过于简练的方案可读性往往较差，因此这样的方案就不太适合采纳。在编程时我们一定要注意避免那些虽然精巧，但是过于简练而导致难以理解的步骤。

当你在为一个问题设计解决思路时，最好先试着构思几个不同的解决思路。然后用分解提纲、伪代码或者流程图来简单描述这几个思路。随后从中挑选出一个你觉得最容易转化为C语句的方案。另外，有些算法专门对某种编程语言特别适用，所以你可能还要挑选一种比较适合于C语言的方案。除此之外，有的时候选择方案还要考虑其他一些因素，譬如执行速度和内存需求。

3.1.4 条件错误

在设计算法时，往往会假设输入数据始终是正确的。但是在实际应用中，程序的输入数据常常是错误的。因此，需要在计算开始之前检验输入数据的正确性，以保证不会引起程序执行异常。当程序执行计算时，也可能会出现引起执行异常的条件。假设现在要计算一个分数值，其中分母的值为0，或者计算海拔的时候结果为负数。这些例子都属于程序执行可能出现的条件错误（error condition，区别于算法中的错误）。

对于有些条件错误，使用本章介绍的语句，程序自身便可以进行检验。但是如果对所有可能发生条件错误的地方都进行检查，这样写出的程序会变得相当长，而且大部分的代码都是用来检查条件错误。因此，程序中的哪些条件错误是应该检查的呢？有些时候，问题描述中就会包含可能的条件错误信息，同时还包含这些条件错误的处理方法。不过，通常的问题描述中都不包含这些条件错误的信息。对于这种情况，你可以依据问题描述简单设计算法，然后提出一系列潜在的条件错误。然后尽可能和该程序的使用者进行讨论，分析会出现的条件错误的各种可能性并做出相应的处理。如果不能找到程序的使用者，那就只能对那些会引起常见错误的一些条件错误在程序中进行检查，然后同时提供一份程序的文档。文档中应当对可能捕获以及可能捕获不到的条件错误进行描述。

一旦你确定了要在算法中引入哪些条件错误的语句，便要确定当条件错误发生时应该如何处理。一般有两种可能：直接退出程序，或者尝试修正错误然后继续执行程序。不论哪种情况，你都要在屏幕上打印出错误信息，说明当前发生了什么错误，以及用户要采取何种操作。一定要保证错误信息越详细越好。比如不要简单地打印出“输入数据发生错误”，而是给出具体的说明信息，“温度越界了”、“时间值为负”或者“压力值超出了安全范围”等。

在本章将会讨论如何读写数据文件（data file），这些文件（类似于程序文件）包含了程序要用到的数据信息。有时程序员会编写一种叫作数据过滤器的程序来专门检查数据文件中是否存在条件错误，然后，使用该数据文件的程序便不必再检查同样的条件错误了。

3.1.5 测试数据的生成

生成测试数据（test data）是设计解决方案的一个重要环节。测试数据首先应该包含专门用来检查程序中条件错误的数据；同时，测试数据还应该负责测试程序中的每一条执行路

85 ~ 86

径。随着程序代码量越来越大，要生成负责检测整个程序的测试数据就会变得异常艰难。关于这个问题的研究叫作软件的确认与验证（validation and verification）。在这方面有大量的课程和书籍。

现在要对如何生成测试数据集给出一些建议。首先，使用手工演算中的数据进行测试。如果程序对于手工演算的数据的运行效果还不理想，那就说明这段程序还没有完全准备好，需要重新设计。一旦这些数据能使程序正确运行，那么再使用一些不同范围取值的数据继续测试。如果测试数据是在某一范围内取值，那么一定要确保使用这些数据去尽量测试边界条件或是极限条件。一旦程序在有效数据下可以运行，那么随后要考虑引入条件错误，以检测程序是否能够作出正确响应。一般来讲，并不会使用一个大数据集来做测试，通常都是使用多组较小规模的数据集来测试程序。

如果在测试程序中发现错误，应马上回到设计算法的阶段查找问题。在分解提纲、伪代码和流程图中将错误更正，然后再修改C程序。如果在程序中做了重大改动，那么整个程序都应该重新测试。有时一些改动确实会对程序造成影响，而我们却未必会考虑到。如果事先保留了大量的测试集和测试记录，那么重新测试就会非常简单，直接用这些数据集重复测试程序就行了。

最后还要介绍一种方法，叫作*程序走查*（program walkthrough）。这在产业界的大规模程序开发当中常常会用到。在程序走查中，负责解决复杂问题的算法设计者会对他们设计出的解决方案进行细节展示和讨论，而展示的对象则是一些并没有参与算法设计但是对于该问题相关的领域具有深入研究的专家学者。算法开发者与专家学者的互相探讨常常可以发现算法中的一些潜在问题，并且还会生成一些潜在的测试数据，以供软件编写完成后进行测试。在程序走查过后，最终的程序会很快完成，并且在准确度上具有极大的保障。在学习解决复杂问题时，你可以试着和同学一起进行一个简单的程序走查。

87

3.2 条件表达式

由于选择结构和循环结构中都要使用条件判断，所以在编写实现这两种结构的语句之前，一定要对判断条件加以讨论。所谓“条件”就是一个可以被判断为true或false的表达式，是一个由关系运算符连接的表达式；除此之外，一个条件也可以包含逻辑运算符。在本节将会对关系运算符和逻辑运算符进行介绍，还会讨论当它们出现在同一个条件里时的判断顺序是怎样的。

3.2.1 关系运算符

C语言中，*关系运算符*（relational operator）用来比较两个表达式。下表是对关系运算符的说明：

关系运算符	说明
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于
!=	不等于

关系运算符的任意一边都可以加空格，但是不能用空格将两个字符组成的运算符分隔开，比如 ==。

下面是一些判断条件的例子：

```
a < b
x+y >= 10.5
fabs(denominator) < 0.0001
```

给出判断条件中标识符的值，就可以计算出条件为 true 还是 false。例如，当 a 等于 5，b 等于 8.4，那么条件 a<b 为真。如果 x 等于 2.3，y 等于 4.1，那么条件 x+y>=10.5 为假。如果 denominator 等于 -0.002 5，那么条件 fabs (denominator) < 0.0001 为假。需要注意的是，空格是用在逻辑表达式中的关系运算符的周围，而不是用在判断条件的算术运算符周围的。

在 C 语言中，一个 true 条件的值为 1，而 false 条件的值为 0。因此，下面的表达式是合法的：

```
d = b>c;
```

如果 b>c，那么 d 的值为 1；否则 d 的值为 0。在条件中也可以使用一个单值。例如，下面的语句：

```
if (a)
   count++;
```

如果 a 的值为 0，那么显然这是一个 false 条件；如果 a 不为 0，那么条件为 true。所以，当 a 非零时，上面语句中的 count 值便自增 1。

88

3.2.2 逻辑运算符

在判断条件中同样可以使用逻辑运算符。但是逻辑运算符要比较的是判断条件，而非表达式。C 语言支持的逻辑运算符（logical operator）有三种：与、或、非。下表是对这三种逻辑运算符的说明：

逻辑运算符	表示符号
与	&&
或	||
非	!

例如，考虑下面的判断条件：

```
a<b && b<c
```

由于关系运算符的优先级要高于逻辑运算符，因此上面的条件应该读作“a 小于 b，并且 b 小于 c”。为了让逻辑语句的可读性更强，常常会在逻辑运算符周围插入空格，但是关系运算符周围通常不放空格。如果给出 a、b、c 的值，就可以判断出该条件为 true 还是 false。例如，当 a 等于 1，b 等于 5，c 等于 8，那么该条件为 true。如果 a 等于 -2，b 等于 9，c 等于 2，那么条件为 false。

如果 A 和 B 都为判断条件，那么使用逻辑运算符可以生成新的判断条件，如 A && B、A || B、!A 和 !B。其中，当且仅当 A 和 B 都为 true 时，条件 A && B 才为 true。对于条件 A || B，当 A 或 B 有一个为 true，或都为 true 时该条件为 true。而 ! 运算符则可以改变一个

条件的值。这样一来，只有当A为false时!A为true；B为false时!B为true。在表3-1中给出了对如上定义的总结。

表3-1 逻辑运算符

A	B	A && B	A || B	!A	!B
False	False	False	False	True	True
False	True	False	True	True	False
True	False	False	True	False	True
True	True	True	True	False	False

在C语言中，当一个包含逻辑运算符的表达式被执行时，编译器只会对表达式中有必要计算的部分做出判断。例如，如果A为false，那么表达式A && B肯定为false，此时就没有必要去判断B的值了。类似地，如果A为true，那么表达式A || B就一定为true，而不需要再去判断B。

3.2.3 优先级和结合性

一个条件可以包含多个逻辑运算符，比如下面的式子：

```
!(b==c || b==5.5)
```

式子中运算符的优先级从高到低依次是!、&&和||，但是使用圆括号便可以改变优先级。在上面的式子里，表达式b==c和b==5.5先被执行。假设b等于3，c等于5，那么这两个表达式都不为真，所以表达式b==c || b==5.5为false。接下来为false条件执行!运算，于是整个判断条件的值变为true。其中，运算符||和&&中间都不能使用空格分隔。在编程中一个经常出现的错误就是在一个逻辑表达式中错把==用成了=。

89

一个判断条件中可以同时包含算术运算符和关系运算符，同样也可以包含逻辑运算符。表3-2展示了一个判断条件中元素之间的优先级和结合性。

表3-2 算术运算符、关系运算符和逻辑运算符的运算优先级

优先级	运算符	结合性
1	()	从内部开始
2	++ -- + - !(type)	从右至左（一元）
3	* / %	从左至右
4	+ -	从左至右
5	< <= > >=	从左至右
6	== !=	从左至右
7	&&	从左至右
8	||	从左至右
9	= += -= *= /= %=	从右至左

练习

判断下面各题中的条件为true还是false。假设下面的变量已经声明过，并赋如下值：

a = 5.5　　b = 1.5　　k = -3

1. a < 10.0+k　　2. !(a == 3*b)　　3. a+b >= 6.5　　4. -k <= k+6

5. k != a-b　　6. a<10 && a>5　　7. b-k > a　　8. fabs(k)>3 || k<b-a

3.3 选择语句

在C程序中，if语句能够对判断条件进行检查，然后根据条件值为true还是false来执行后面的语句。C语言包含两种形式的if语句——简单if语句和if/else语句。C语言同样包含一种叫作switch的语句，用它可以检查多个条件，然后再根据这些条件的值是true还是false来确定该执行哪一组语句。

3.3.1 简单if语句

最简单的if语句具有下面的一般形式：

```
if（条件）
    语句1;
```

如果其中的条件为真，则执行语1；如果条件为假，则跳过语句1。在if语句中包含的语句应有缩进，这样一来使得程序中语句的结构更加清晰。 90

如果希望当一个条件判定为true时，就要执行几条特定语句（或者是顺序结构），此时需要用到复合语句（compound statement），或称为块，这是一组包含在大括号中的语句。大括号的位置则根据编程风格而定；下面给出了两种常见的形式：

```
第一种形式                第二种形式
if（条件）                if（条件）  {
{
    语句1;                    语句1;
    语句2;                    语句2;
    ...                       ...
    语句n;                    语句n;
                          }
```

本书一般都使用第一种形式，使每个大括号都自成一行。如此一来，虽然显得代码行数有些多，但是不小心漏掉某个括号就很容易发现。图3-5的流程图中展示了if语句的流程图，当条件为true时，左图中会执行一条语句，而右图中会执行多条语句。

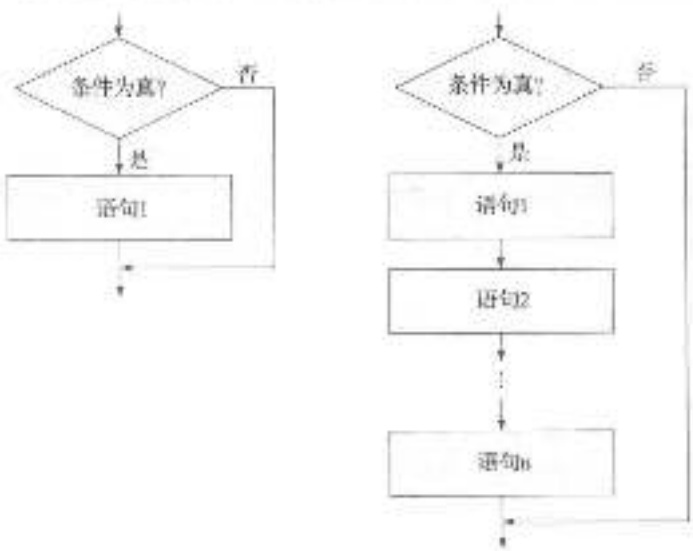


图3-5 选择语句的流程图

下面就是if语句一个具体示例：

```
if (a < 50)
{
   ++count;
   sum += a;
}
```

91

这段代码表示，如果 a 小于 50，则 count 自增 1，同时 sum 的值加 a；否则这两条语句会被跳过。

if 语句同样可以嵌套使用：在下面的例子中，一个 if 语句中包含了另一个 if 语句：

```
if (a < 50)
{
   ++count;
   sum += a;
   if (b > a)
      b = 0;
}
```

如果 a 小于 50，则 count 自增 1，同时 sum 值加 a。另外，如果 b 大于 a，则将 b 的值置为 0。如果 a 大于等于 50，则跳过这些语句。当嵌套使用 if 语句的时候一定要保证语句之间的缩进关系。

3.3.2 if/else 语句

if/else 语句可以根据 condition 的值来选择执行不同的语句。下面给出了 if/else 的一种最简单形式：

```
if (条件)
   语句1;
else
   语句2;
```

上面的语句 1 和语句 2 可以替换为复合语句。当然这两处其中之一也可以为空语句（empty statement），只写一个分号即可。如果语句 2 为空语句，那么此时 if/else 语句就变成了一个简单 if 语句。有些时候在语句 1 处使用空语句比较省事；当然，也可以将条件反转后把这些语句重新写成一个简单 if 语句。例如，下面的两组语句是等价的：

```
if (a < b)          if (a >= b)
   ;                    count++;
else
   count++;
```

现在考虑下面的 if/else 语句：

```
if (d <= 30)
   velocity = 0.425 + 0.00175*d*d;
else
   velocity = 0.625 + 0.12*d - 0.0025*d*d;
```

在这个例子中，当距离 d 小于等于 30 的时候，velocity 按照第一个赋值语句来计算；否则 velocity 按照第二个赋值语句来计算。该 if/else 语句的流程图如图 3-6 所示。

92

下面是另一个 if/else 语句的例子：

```
if (fabs(denominator) < 0.0001)
   printf("Denominator close to zero");
else
{
   x = numerator/denominator;
```

```
    printf("x = %f \n",x);
}
```

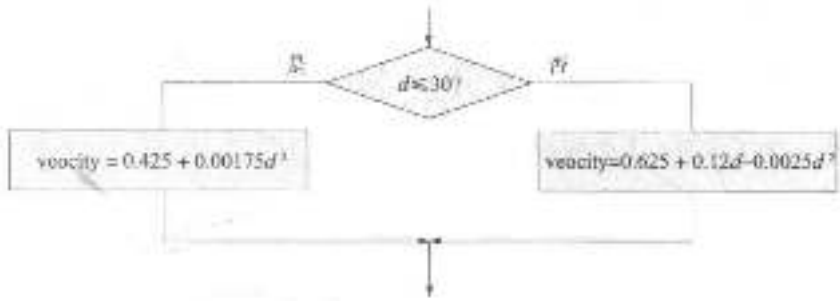


图 3-6 if/else 语句的流程图

在这个例子中，我们检测变量 denominator 的绝对值，如果这个值接近于 0，那么输出不能计算除法的信息。如果 denominator 的值不接近于 0，计算并输出 x 的值。这条语句的流程图在图 3-3 中已给出。

下面是一个嵌套使用 if/else 语句的例子：

```
if (x > y)
   if (y < z)
      k++;
   else
      m++;
else
   j++;
```

当 x>y 且 y<z 时，k 的值加 1。当 x>y 且 y>=z 时，m 的值加 1。当 x<=y 时 j 的值加 1。正确的运用缩进，这条语句是很容易明白的。如果去掉内部 if 语句的 else 部分，并且保持同样的缩进，那么这条语句就变为下面的语句：

```
if (x > y)
   if (y < z)
      k++;
else
   j++;
```

93

看起来好像是 x<=y 时 j 的值增加 1，但其实这是错误的。C 编译器会把 else 语句和距离它最近的 if 语句匹配，所以，不管是怎样缩进的，上一条语句将按下面的语句执行：

```
if (x > y)
   if (y < z)
      k++;
   else
      j++;
```

所以当 x>y 并且 y>=z 时，j 的值加 1。如果想要让 j 在 x<=y 时加 1，那么就需要用大括号把内部语句定义为一个语句块：

```
if (x > y)
{
   if (y < z)
      k++;
}
else
   j++;
```

为了避免在使用嵌套 if/else 语句时引起歧义和错误，应该习惯性地用大括号将要一起执行的语句定义为语句块。

C 语言允许使用条件运算符（conditional operator）来代替 if/else 语句。条件运算符是一个三元运算符，因为它含有三个操作数——条件，符合条件时要执行的语句和不符合条件时要执行的语句。运算符跟在条件后面，用 ? 表示，接着后面有两条语句，这两条语句中间用冒号隔开。为了说明，下面两条语句是等价的：

```
if (a<b)                a<b ? count++ : c = a + b;
   count++;
else
   c = a + b;
```

条件运算符（定义为 ?:）在赋值语句之前执行，如果一个表达式中包含多个条件运算符，其结合性是从右至左的。

本节给出了选择语句中常用的几种数值比较的方法。但是在进行浮点数比较时必须要谨慎。在第 2 章中已经提到，由于二进制和十进制之间的转换，浮点数有时会和预期的有些不同。例如，在本节前面，我们没有将 denominator 与 0 比较，而是用一个替代条件来判断 denominator 的绝对值是否比一个很小的值还小。类似的，如果想知道 y 的值是否接近于 10.5，应该使用类似于 fabs(y-10.5)<= 0.0001 这样的条件，而不是 y == 10.5。一般而言，不直接使用等号运算符进行浮点数的相等判断。

94

练习

针对练习 1 – 7，画出流程图来表示执行步骤，然后给出相应的 C 语句。假设变量已被声明，而且被合理赋值。

1. 如果 time 的值大于 15.0，将 time 的值增加 1.0。
2. 当 poly 的平方根小于 0.5 时，输出 poly 的值。
3. 如果 volt_1 和 volt_2 间的差值超过 10.0，输出 volt_1 和 volt_2 的值。
4. 如果 den 的值小于 0.05，将 result 的值设为 0；否则将 den/num 的结果赋值给 result。
5. 如果 x 的自然对数大于等于 3，将 0 赋值给 time，并且将 count 的值减 1。
6. 如果 dist 小于 50.0 而且 time 大于 10.0，将 time 的值增加 2；否则，将 time 的值增加 2.5。
7. 如果 dist 大于或者等于 100.0，将 time 的值增加 2.0；如果 dist 的值在 50 到 100 之间，将 time 的值增加 1；否则，将 time 的值增加 0.5。

3.3.3 switch 语句

switch 语句用于多重条件选择，经常用来代替嵌套 if/else 语句。在讨论 switch 语句之前，我们先给出一个使用嵌套 if/else 语句的例子，然后使用 switch 语句给出一个等价的解决方案。

假设我们从一个大型机器内部的传感器读取了温度，想要将信息输出到控制屏幕以告知操作员温度状态。如果状态码是 10，那么温度就太高，需要关掉设备；如果状态码是 11，那么操作员应该每 5 分钟检查一次温度；如果状态码是 13，操作员应该打开排风扇；如果是其他的状态码，设备就是在正常运行。下面的一组嵌套 if/else 语句可以在屏幕输出正确信息：

```
if (code == 10)
   printf("Too hot - turn equipment off \n");
else
{
```

```
    if (code == 11)
       printf("Caution - recheck in 5 minutes \n");
    else
    {
       if (code == 13)
          printf("Turn on circulating fan \n");
       else
          printf("Normal mode of operation \n");
    }
}
```

95

下面的 switch 语句是它的等价语句：

```
switch (code)
{
   case 10:
      printf("Too hot - turn equipment off \n");
      break;
   case 11:
      printf("Caution - recheck in 5 minutes \n");
      break;
   case 13:
      printf("Turn on circulating fan \n");
      break;
   default:
      printf("Normal temperature range \n");
      break;
}
语句1;
```

break 语句表示 switch 语句执行结束，接下来执行它后面的语句（语句 1），因此是跳过了花括号中剩下的语句。

嵌套 if/else 语句并不是总能转换成 switch 语句。但是，如果能转换，switch 语句更容易读。switch 语句所需要的分隔符也更简洁。实际上，如果 if/else 语句中的大括号，分号等使用不正确，编译器就不能执行所期望的语句。

switch 语句基于控制表达式（controlling expression）选择要执行的语句，控制表达式必须是整型或字符型。表达式后面一般跟着选择标签（case label）（label_1,lable_2,...）来决定要执行哪条语句，有有些语言中，把这种结构称为选择结构（case structure）。如果控制表达式的值与选择标签的值相等，对应的该标签的程序语句就会被执行。选择标签必须是唯一的常量。如果两个以上选择标签的值相同就会被认为是语法错误。如果其他语句都没有执行，那么执行 default 标签（default label），但是 default 标签并不是必需的。下面是代码：

```
switch （控制表达式）
{
   case label_1:
      语句;
   case label_2:
      语句;
   ...
   default:
      语句;
}
```

switch 结构语句中经常包含 break 语句。当 break 语句被执行时，程序执行将跳出 switch 结构，继续执行 switch 结构后面的语句。没有 break 语句，程序将会顺序执行选

96 中的那条 case 标签后的所有语句。

虽然从语法上说 default 标签在 switch 结构中是可有可无的，但是我们仍然建议写出 default 语句，用于明确指出当没有 case 标签与控制表达式相匹配时要执行的步骤。有时候在 default 语句中使用 break 语句，来通知程序继续执行 switch 结构后面的语句。

在同一语句中使用多个 case 标签是合法的，如下面所示：

```
switch (op_code)
{
   case 'N': case 'R':
      printf("Normal operating range \n");
      break;
   case 'M':
      printf("Maintenance needed \n");
      break;
   default:
      printf("Error in code value \n");
      break;
}
```

当一条语句中有一个以上的 case 标签时，类似用逻辑运算符 || 将 case 标签连接起来。在这个例子中，如果 op_code 等于 'N' 或者 'R'，第一条语句都会被执行。

练习

将下面的嵌套 if/else 语句转换为 switch 语句：

```
if (rank==1 || rank==2)
   printf("Lower division \n");
else
{
   if (rank==3 || rank==4)
      printf("Upper division \n");
   else
   {
      if (rank==5)
         printf("Graduate student \n");
      else
         printf("Invalid rank \n");
   }
}
```

3.4 解决应用问题：人脸识别

本节将运用本章学习的新语句来解决人脸识别相关的问题。

如图 3-7 所示，在进行脸部比对时一种有效的技术是比较脸部关键点距离的比率值。这些比率值中常用的一个是两眼间的距离除以鼻子和下巴间的距离。因为这些测量值是比率，所以可以对不同大小的图像进行计算。同一张脸在不同大小图像上计算结果应该是相似的。计算机程序在计算这些比率值之前首先需要在一张图像中定位人脸的位置，然后在人脸上定位眼睛和其他关键点的位置。如果图像中的人脸不是正对前方，而是转向不同的方向，那就需要对图像做额外的处理。

97

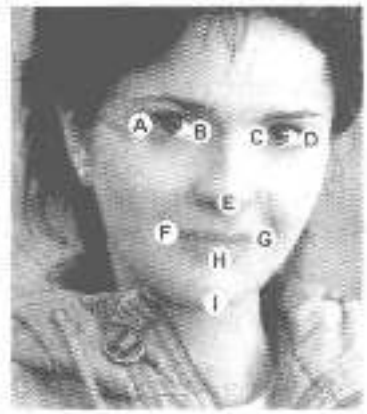


图 3-7 人脸识别的关键点

对于这个问题，假设有一张正面朝向相机的人脸图像，

想要确定是否其中两张图像来自同一个人。我们使用的技术是比较眼睛外边缘间距离与下巴下端和鼻子顶端间距离的比率。编写一个C程序读取每张脸的这两个距离，计算比率，然后确定那两张图像有相近的比率。

1. 问题陈述

给出三张脸部的信息，用比率确定哪两张脸是最相近的。

2. 输入/输出描述

如下图所示，程序的输入是三张不同图像的眼睛外边缘间的距离与下巴尖和鼻子顶端的距离。输出的是以这些距离比率为基础的最相近的两张图像的编号。

98

3. 手动演算示例

假设下面的距离是从三张图像中计算出来的，单位是cm：

	图像1	图像2	图像3
眼睛外边缘间的距离	5.7	6.0	6.0
鼻子到下巴的距离	5.3	5.0	5.6

然后可以计算每幅图片眼睛外部边缘距离与下巴和鼻子间距离的比率：

$$ratio_1 = 5.7/5.3 = 1.08$$
$$ratio_2 = 6.0/5.0 = 1.20$$
$$ratio_3 = 6.0/5.6 = 1.07$$

然后计算两两比率之差，使用绝对值使得每个差值都是正数：

$$diff_1_2 = |ratio_1 - ratio_2| = |1.08 - 1.20| = 0.12$$
$$diff_1_3 = |ratio_1 - ratio_3| = |1.08 - 1.07| = 0.01$$
$$diff_2_3 = |ratio_2 - ratio_3| = |1.20 - 1.07| = 0.13$$

使用比率计算出的差值最小的两张图像是最相近的。在这个例子中，图像1和图像3是最相近的。需要指出的是，可能计算出的这三个差值都很大，即使选择三个之中的最小值仍然不能得出一个良好的匹配结果。基于距离检测的商业化脸部识别系统中，为了提高精确度通常使用大量的距离值共同计算。

4. 算法设计

算法开发的第一步是将解决方案分解成一组顺序执行的步骤。

分解提纲

1）读取每幅图像的距离。

2）计算每幅图像的比率。

3）计算两两比率的差值。

4）找到最小的差值。

5）输出最佳匹配的对应图像编号。

这个程序的结构很简单，可以将分解提纲直接转换为C程序。

```
/*-----------------------------------------------------------------*/
/*  程序chapter3_1                                                 */
/*                                                                 */
/*  本程序使用两眼之间的距离以及鼻子与下巴之间的距离，目标是选择最相近  */
/*  的两张图像                                                     */
#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  声明变量  */
   double eyes_1, eyes_2, eyes_3, nose_chin_1, nose_chin_2,
          nose_chin_3, ratio_1, ratio_2, ratio_3, diff_1_2,
          diff_2_3, diff_1_3;

   /*  从键盘获取用户输入  */
   printf("Enter values in cm. \n");
   printf("Enter eye distance and nose-chin distance for image 1: \n");
   scanf("%lf %lf",&eyes_1,&nose_chin_1);
   printf("Enter eye distance and nose-chin distance for image 2: \n");
   scanf("%lf %lf",&eyes_2,&nose_chin_2);
   printf("Enter eye distance and nose-chin distance for image 3: \n");
   scanf("%lf %lf",&eyes_3,&nose_chin_3);

   /*  计算比率  */
   ratio_1 = eyes_1/nose_chin_1;
   ratio_2 = eyes_2/nose_chin_2;
   ratio_3 = eyes_3/nose_chin_3;

   /*  计算差值  */
   diff_1_2 = fabs(ratio_1 - ratio_2);
   diff_1_3 = fabs(ratio_1 - ratio_3);
   diff_2_3 = fabs(ratio_2 - ratio_3);

   /*  找到最小差值，并打印图像序号  */
   if ((diff_1_2 <= diff_1_3)) && (diff_1_2 <= diff_2_3)
      printf("Best match is between images 1 and 2 \n");
   if ((diff_1_3 <= diff_1_2)) && (diff_1_3 <= diff_2_3)
      printf("Best match is between images 1 and 3 \n");
   if ((diff_2_3 <= diff_1_3)) && (diff_2_3 <= diff_1_2)
      printf("Best match is between images 2 and 3 \n");

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

5. 测试

首先用手动演算示例中的数据测试程序。下面是程序交互过程：

```
Enter values in cm.
Enter eye distance and nose-chin distance for image 1:
5.7 5.3

Enter eye distance and nose-chin distance for image 2:
6.0 5.0
Enter eye distance and nose-chin distance for image 3:
6.0 5.6
```

```
Best match is between images 1 and 3
```

程序计算出的答案和手动演算的答案一致，因此可以用其他数据来测试程序。打印出三张不同人的图像，试着预测这个人脸识别系统会将哪两张图像匹配。用同一个人的两张不同图像和第三个人的图像来运行程序，看这种技术能否将同一个人的两张图像识别出来。

修改

以下这些问题和从三张图像中找两张最佳匹配的程序算法相关。

1. 修改程序使得能够输出三个差值。这给出了匹配精度的额外信息。差值越小，匹配得越好。
2. 如果其中两个比率的值是相同的最小值，会得到怎样的输出？如果三个比率都是相同的，会得到怎样的输出？编制一些能够产生这样结果的输入数据，并检验你的答案。
3. 修改程序，在解决方案中使用嵌套 if 语句而不是三个独立的 if 语句。如果使用这种方案，那么问题 2 的答案变了吗？
4. 修改程序，使得程序输出的是最不像的两张图像的图像编号。
5. 修改程序，使得程序输出 4 张图像中最佳匹配的两张。（注意这使得程序变长了，因为 4 张图像会有 6 种最佳匹配的可能，在第 5 章我们将会学习额外的 C 语言功能，使得我们能够解决这种问题而又不用增加代码。）

3.5 循环结构

循环用来实现重复结构。C 语言有三种循环结构——while 循环、do/while 循环和 for 循环。除此之外，C 语言还提供了两条在循环结构中来修改循环执行的语句——break 语句（在 switch 语句中使用过）和 continue 语句。 101

在学习循环结构之前，我们先给出两条调试的建议，用于方便地在包含循环语句的程序中找错误。当编译较长的程序时，产生大量的编译错误的情况很多。与其逐个分析每一个错误并一一修正，本书建议首先修改明显的语法错误，并在修改后立刻重新编译程序。一个错误往往会导致几条错误信息，而一些错误信息可能描述的是你的程序中不存在的错误，是由于前面的错误扰乱编译器而使得产生了错误的判断。

第二个调试建议和循环中的错误有关。如果你想确定循环的执行步骤是否和你想的一样，那么就在循环语句中写一条输出语句 printf，使得循环每次执行时都可以输出关键变量的内存快照。然后，如果发生了错误，屏幕上将会有很多已输出的信息用来帮助定位错误原因。

3.5.1 while 循环

下面是 while 循环的一般形式：

```
while (条件)
{
    语句;
}
```

在执行大括号里面的语句之前首先要对条件语句进行判断。如果条件为假，循环语句将会被跳过，执行 while 循环后面的语句。如果条件为真，循环语句会被执行，然后再一次判断条件。如果还为真，循环语句再一次被执行，然后再次回到条件判断。这个过程不断重复，直到条件为假时退出。循环中必须要有语句来改变条件中的变量，否则条件判断的结果将一直不变，那就意味着循环中的语句将永远不会被执行，或者循环永远不会结束。如果

while循环中的条件一直判定为真，它就是一个无限循环（infinite loop）。大多数系统中都会定义一个程序的可持续时间限制，如果一个程序的运行时间超过限制，就会产生一个运行时错误终止这个程序的执行。其他系统要求用户输入一组特殊字符，比如control+c（缩写为^c）来结束或者终止程序的执行。几乎每个人都会无意或有意地在程序中写出无限循环，所以一定要知道你的系统中用于终止程序执行的特殊字符。

下面的伪代码和程序使用while循环产生一个将角度转换为弧度的转换表（注意角度从0°开始，每次增加10°，一直到360°）：

```
[提炼后的伪代码]
主函数：设置 degrees 为 0
        while degrees <= 360
            将 degrees 转换成 radians
            print degrees, radians
            degrees 加 10
```

102

```
/*------------------------------------------------------------*/
/*  程序 chapter3_2                                           */
/*                                                            */
/*  本程序使用 while 循环结构实现角度到弧度的转换              */

#include <stdio.h>
#define PI 3.141591

int main(void)
{
   /*  声明和初始化变量  */
   int degrees=0;
   double radians;

   /*  循环打印弧度和角度  */
   printf("Degrees to Radians \n");
   while (degrees <= 360)
   {
      radians = degrees*PI/180;
      printf("%6i %9.6f \n",degrees,radians);
      degrees += 10;
   }

   /*  退出程序  */
   return 0;
}
/*------------------------------------------------------------*/
```

程序的前几行输出如下：

```
Degrees to Radians
     0  0.000000
    10  0.174533
    20  0.349066
     . . .
```

3.5.2 do/while 循环

do/while循环和while循环很相似，除了do/while循环的条件是在循环结尾检测而不是循环的开始处。在循环结尾检测条件可以确保do/while循环至少执行一次。但对于while循环，如果条件不满足，它可能根本不会执行。do/while循环的一般形式如下所示：

```
do
{
   语句;
} while (条件);
```

下面的伪代码和程序使用 do/while 循环而不是 while 循环输出角度到弧度的转换表： 103

```
[展示差的伪代码]
主函数：设置 degrees 为 0
        do
          将 degrees 转换成 radians
          print degrees, radians
          degrees 加 10
        while degrees <= 360
```

```
/*                                                          */
/*   程序 chapter3_3                                        */
/*                                                          */
/*   本程序使用 do-while 循环结构实现角度到弧度的转换       */

#include <stdio.h>
#define PI 3.141593

int main(void)
{
   /*  声明和初始化变量  */
   int degrees=0;
   double radians;

   /*  循环打印弧度和角度  */
   printf("Degrees to Radians \n");
   do
   {
      radians = degrees*PI/180;
      printf("%6i %9.6f \n",degrees,radians);
      degrees += 10;
   } while (degrees <= 360);

   /*  结束程序  */
   return 0;
}
/*----------------------------------------------------------*/
```

3.5.3 for 循环

许多程序要求循环中变量每次增加（或减少）相同的量，当变量随着循环逐步达到指定值，就退出循环，这种类型的循环可以用 while 循环实现，也可以很容易地用 for 循环（for loop）实现。for 循环的一般形式如下所示：

```
for (表达式1; 表达式2; 表达式3)
{
   语句;
}
```

104

表达式 1 用来初始化循环控制变量（loop-control variable），表达式 2 指定继续执行循环需要满足的条件，表达式 3 指定循环控制变量的修改。

比如，想要执行10次循环，就可以设定变量k的值从1开始每次增加1直到10，即可得出下面的for循环结构：

```
for (k=1; k<=10; k++)
{
   语句;
}
```

注意大括号的使用风格应保持一致。

如果想要执行一个循环使得变量n的值从20递减到0，每次减2，可以使用下面的循环结构：

```
for (n=20; n>=0; n-=2)
{
   语句;
}
```

这个for循环也可以用下面的形式来写：

```
for (n=20; n>=0; n=n-2)
{
   语句;
}
```

两种形式都是有效的，但是一般使用缩写形式，因为它更短。

利用下面的表达式可以计算出for循环的执行次数：

$$\text{floor}\left(\frac{\text{final value} - \text{initial value}}{\text{increment}}\right) + 1$$

如果这个值是负数，循环将不会被执行。因此，如果是下面的for循环结构：

```
for (k=5; k<=83; k+=4)
{
   语句;
}
```

它将被执行下面的次数：

$$\text{floor}\left(\frac{83 - 5}{4}\right) + 1 = \text{floor}\left(\frac{78}{4}\right) + 1 = 20$$

k的值是5，然后是9，接着是13，以此类推，直到最后的值81。当k的值是85时程序将不会被执行，因为当k的值等于85时循环条件不正确。

下面的语句是for循环的嵌套使用：

```
for (k=1; k<=3; k++)
   for (j=0; j<=1; j++)
      count++;
```

105

外部for循环语句将被执行3次。外部for循环每执行一次，内部for循环执行2次。因此，变量count将被增加6次。

下面的伪代码和程序能够打印出角度制和弧度制的转换对应关系表。在学习while循环时我们实现过一次，现在用for循环语句再来实现。（需要注意的是，使用while循环和使用for循环时，程序的伪代码是相同的。）

[转换后的伪代码]

主函数：设置degrees为0

```
        while degrees <= 360
            将 degrees 转换为 radians
            print degrees, radians
            degrees += 10
/*-----------------------------------------------------------------*/
/*   程序 chapter3_4                                               */
/*                                                                 */
/*   本程序使用 for 循环结构实现角度到弧度的转换                    */

#include <stdio.h>
#define PI 3.141593

int main(void)
{
   /*  声明变量  */
   int degrees;
   double radians;

   /*  循环打印角度和弧度  */
   printf("Degrees to Radians \n");
   for (degrees=0; degrees<=360; degrees+=10)
   {
      radians = degrees*PI/180;
      printf("%6i %9.6f \n",degrees,radians);
   }

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

在程序中可以看到，变量 degrees 在声明后没有必要进行初始化，因为在 for 循环语句中完成了对它的初始值设置。

在 for 循环的初始化语句和修改语句中可以加入多条语句，在下面的例子中可以看到循环中对两个变量的初始化和修改操作。

```
for (k=1, j=5; k<=10; k++, j++)
{
   sum_1 += k;
   sum_2 += j;
}
```

106

当在循环程序中使用了多条语句时，需要用逗号将它们分隔开。这些语句会按从左到右的顺序依次执行。逗号运算符的优先级最低，只是起到语句分隔的作用。

练习

计算出下列 for 循环执行的次数：

```
1. for (k=3; k<=20; k++)
   {
      语句;
   }
2. for (k=3; k<=20; ++k)
   {
      语句;
   }
```

```
3. for (count=-2; count<=14; count++)
   {
      语句;
   }
4. for (k=2; k>=-10; k--)
   {
      语句;
   }
5. for (time=10; time>=5; time++)
   {
      语句;
   }
```

6. 在执行完以下嵌套 for 循环语句后，count 的值是多少？

```
int k, j, count=0;
for (k=-1; k<=3; k++)
   for (j=3; j>=1; j--)
      count++;
```

3.5.4 break 语句和 continue 语句

前文中我们在 switch 语句中使用过 break 语句用于跳出当前 switch 的语句组，类似的，break 语句可以用在所有循环结构中，用来跳出正在执行的循环。相反，continue 语句是用来跳过循环中正在进行的这一轮语句（循环中的一轮可以称为一个迭代（iteration）），然后执行下一轮循环迭代操作。因此在 while 循环或 do/while 循环中，在 continue 语句执行之后要回到条件判断语句，用于决定循环中的语句是否要继续执行。在 for 循环中，循环控制变量会变化，然后再对继续重复操作的情况进行判断，从而决定是否要再执行一次。break 和 continue 语句用于出现错误后需要退出当前迭代或跳出整个循环的情况。

107

为了更好地理解 break 和 continue 语句的区别，请阅读以下循环语句。该循环语句会不断地从键盘读入用户输入的数值。

```
sum = 0;
for (k=1; k<=20; k++)
{
   scanf("%lf",&x);
   if (x > 10.0)
      break;
   sum += x;
}
printf("Sum = %f \n",sum);
```

这个循环读取了 20 个输入的数值，如果这 20 个数值全都小于等于 10.0，那么循环将会计算出这些值的和并输出结果。一旦用户输入了大于 10.0 的数值，那么 break 语句将会立即终止循环并执行输出操作，因此，最终输出的 sum 值是只有小于 10.0 的值的和，唯一的一次出现大于 10.0 的值并没有被计入在内。

接下来将上面的循环语句做一个轻微的改动：

```
sum = 0;
for (k=1; k<=20; k++)
{
   scanf("%lf",&x);
   if (x > 10.0)
      continue;
```

```
    sum += x;
  }
  printf("Sum = %f \n",sum);
```

在这个循环中，如果这 20 个数值全部小于等于 10.0，那么循环将会输出这些值的和。如果用户输入了大于 10.0 的数值，continue 语句会使其跳过该次循环中剩下步骤，并继续执行下一次循环。因此，最终输出的结果是这 20 个值中小于等于 10.0 的值之和。

3.6 解决应用问题：波互作用

海洋的状态是由风速和相应的波反映的，又被称为海况（sea state）。如表 3-3 所示，海况被划分为 0 到 12 等级，可以表示的最高风速超过 70mph（英里[①]/小时，飓风的速度）。比如 1 级海况表示海面会有微风（1 – 3mph）和轻微的水面波动，而 4 级海况时会有中度的微风（13 – 18mph），并且海面会有许多白浪，当达到 6 级时海面会有强风（25 – 31mph）并形成大波浪，白浪遍布海面且空中还会有海洋飞沫。

表 3-3 海况等级

等级	描述	风速（mph）	海面反应
0	平静	无风	海面光滑如镜
1	细风	1 – 3	轻微波动
2	轻风	4 – 7	轻微的海浪
3	微风	8 – 12	少量分散的白浪
4	和风	13 – 18	小波浪
5	劲风	19 – 24	许多白浪
6	强风	25 – 31	大浪
7	强劲的疾风	32 – 38	伴有白色泡沫的巨浪
8	狂风	39 – 46	接近中等高度的海浪
9	烈风	47 – 54	接近较高的海浪
10	暴风	55 – 63	接近非常高的大浪
11	猛烈的暴风	64 – 72	接近极高的巨浪
12	飓风	>73	布满白沫的滔天巨浪

资料来源：Harold V. Thurman and Elizabeth A. Burton, *Introductory Oceanography*, 9th ed. Prentice-Hall, Upper Saddle River, NJ, 2001.

如本章开头提到的，每个波都有其波峰和波谷，二者的垂直距离便是振幅，其水平距离称为波长（wavelength）。深水波常常是由海面上的风引起的，当水深大于波长的一半时，波便会出现，并且水深并不会影响深海中波的速度。而在浅水域中，当水深小于波长的 1/20 时便会出现浅水波，其中还包括潮汐波。并且浅水域中波速是由水深决定的——水越深，波速越大。而水深介于波长的 1/20 和 1/2 之间的过渡波的速度是由波长和水深共同决定的。由于海底的摩擦力，波速会下降，因而波高会随之增大。当水深达到波高的 1.3 倍时，水波就会涌上岸来。 108

在海洋环境中，有很多因素会导致波的生成，并且波会来自四面八方。不同的波发生干涉会生成一系列复杂的波，它们有着各种各样的波峰和波谷。假设同时有两个波，如果其中一个波的波峰和波谷恰好与另一个的一致，则它们相结合将会产生相长干涉，二者的波峰和波谷都会叠加，从而产生更高的波峰和更深的波谷，也就得到了更大的振幅。同理，如果是一个波的波峰遇到了另一个波的波谷，就会产生相消干涉，甚至出现波峰和波谷完全抵消的情况。在混合干涉中，如果两波有着不同的波长和高度，那么其干涉会很复杂，因为这样的

① 1 英里 = 1.609 344 千米

波相遇过程既会有相长干涉也会有相消干涉。

为了研究两个波的干涉模式（interference pattern），我们要设计一个程序来观察两个波的周期和高度，然后该程序会输出两个波的波长，然后计算出波的最大高度（假设两个波之间没有相位差）。在得出解决方案之前，首先分析正弦波。我们将会用正弦波来模拟海波中的波，并通过其周期来计算波长。

回忆一下正弦函数，它是角度的函数，函数值介于-1和+1之间，周期为2π，如图3-8所示。正弦曲线是表现正弦函数的一种方式，但它是关于时间的函数，而不是关于角度的函数，例如

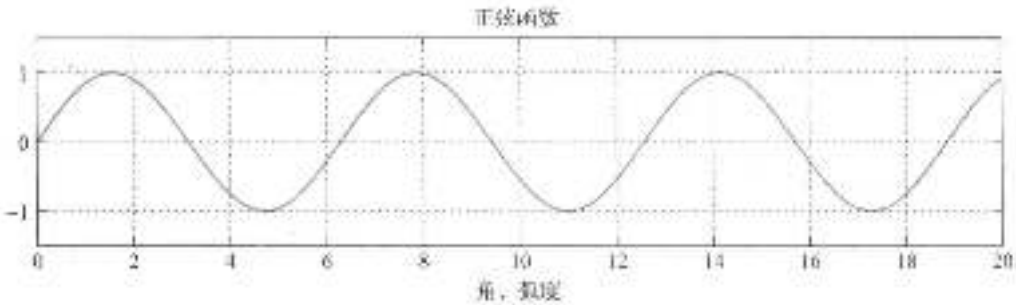


图3-8 正弦函数图像

$$s(t) = A\sin(2\pi ft + \phi)$$

在这个公式中，A是振幅，f是频率（单位是Hz，每秒的圈数），ϕ是相位差（单位是弧度）。正弦曲线的周期是$1/f$秒。

[109]

考虑以下三个函数：

$$s_1(t) = 3\sin(2\pi t)$$
$$s_2(t) = 5\sin(0.4\pi t)$$
$$s_3(t) = 5\sin(\pi t)$$

函数s_1的振幅是3，函数s_2和s_3的振幅都为5；函数s_1的频率是1Hz，因此其周期为1s，函数s_2的频率为0.2Hz，因此其周期为5s，函数s_3的频率是0.5Hz，因此其周期为2s；三个函数的相位均为0。图3-9展示了以上三个函数的图像。

正弦波有一个性质是两个正弦波叠加所得的波仍然有周期性，且其周期等于之前两个独立正弦波周期的最小公倍数（Least Common Multiple，LCM）。因为篇幅原因我们没有给出证明，但这个性质在分析波的干涉时非常有用。例如，如果周期分别为3s和6s的两波相叠加，最终得一周期为6s的波；如果周期分别为3s和5s的两波相叠加，最终得一周期为15s的波；如果周期分别为1/2s和1/3s的两波相叠加，最终得一周期为1s的波；如果周期分别为2/3s和2s的两波相叠加，最终得一周期为6s的波。计算任意两个数字的最小公倍数并不是一件很容易的事。我们的主要研究目标是波的干涉的特性，为简单起见，先假设输入程序的波的周期都是整数（单位为s）。既然如此，最小公倍数一定会小于等于这两个整数的乘积，因此如果我们直接将该乘积作为干涉后波的周期，那么最大的波峰一定会在某个时间点周期性的出现。图3-10展示了图3-9中三个函数两两叠加的结果，这里需要注意的是，s_1+s_2的周期是5（1和5的最小公倍数），s_2+s_3的周期是10（5和2的最小公倍数），s_1+s_3的周期是2（1和2的最小公倍数）。

现在我们需要考虑波长L和周期T之间的关系。对于一个深水中的波，该关系可以用如下表达式来表示：

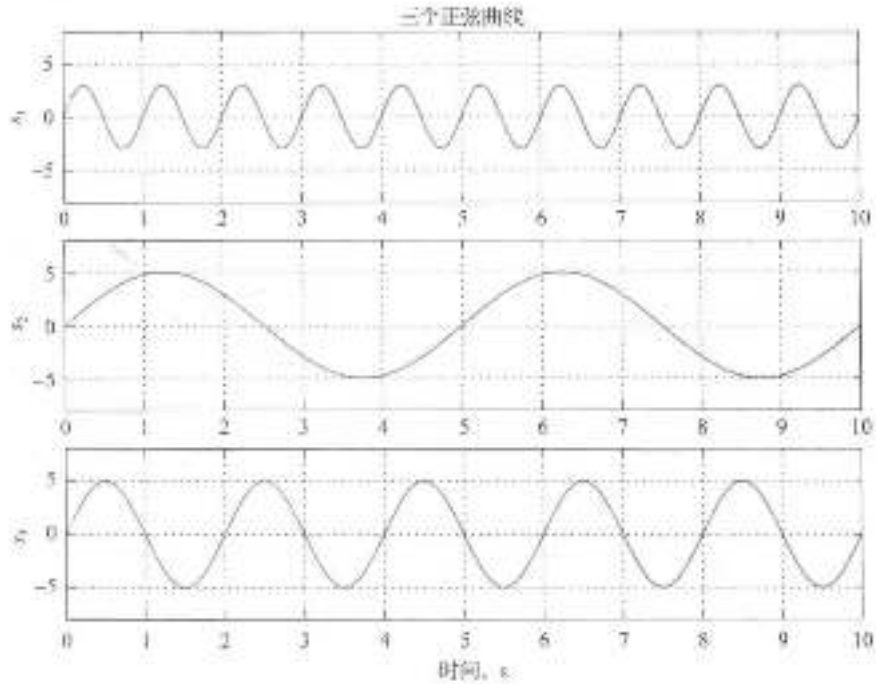


图 3-9 三个正弦曲线的图像

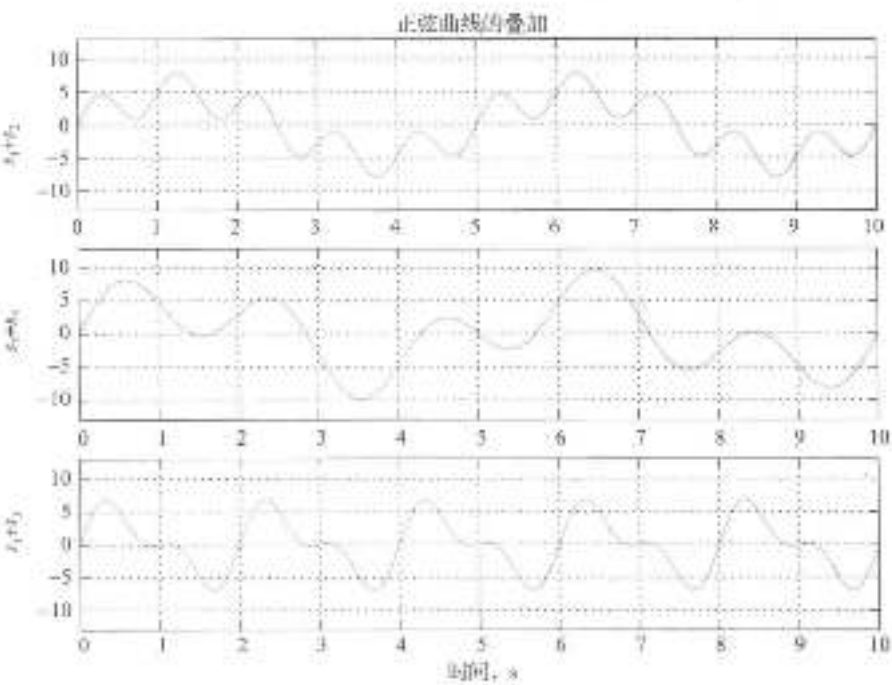


图 3-10 正弦曲线两两叠加的图像

$$L = 5.13T^2$$

其中 T 的单位是秒，L 的单位是英尺。此关系表达式并不适用于浅水波。现在我们已经学习了所有的相关知识，下面开始讨论如何解决本节开头提出的问题。

现在我们要写出一个程序，实现以下功能：让用户输入两束波的周期和振幅，且周期是

以秒为单位，振幅以英尺为单位，为简单起见周期数据设定为整数。程序计算并输出每束波的波长；以两个波的周期的乘积作为叠加后波的周期，计算出叠加波一个时间周期内的200个点；然后找出这200个点中的最大值，并将该值当作叠加波的振幅的估计值。叠加过程中假设两波相位差为0。

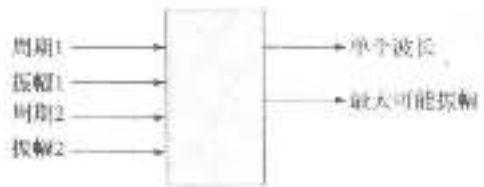
110 ~ 111

1. 问题陈述

计算并得出两波的波长和两波叠加后的最大可能振幅。

2. 输入/输出描述

下图表示了该问题中输入项为每束波的周期和振幅，输出为每束波的波长和合成波的最大可能振幅。



3. 手动演算示例

假设我们已经有了两束波的周期和振幅（波峰与波谷的垂直距离）：

	周期(s)	振幅(ft)
波1	4	0.5
波2	10	1.0

首先我们要用本节先前给出的等式计算出每束波的波长：

$$波长1 = 5.13T_1^2 = 82.08ft$$
$$波长2 = 5.13T_2^2 = 513ft$$

我们用两波周期的乘积作为叠加波的周期，在本例中是40s。在这个时间周期内均匀地取出200个点，因此这200个点中时间的单位增量为40/200=0.2s，在手动演算时我们使用其中的10个点，于是得出下列信息：

t	$w_1(t)$	$w_2(t)$	$w_1(t)+w_2(t)$
0	0	0	0
0.2000	0.0773	0.0627	0.1399
0.4000	0.1469	0.1243	0.2713
0.6000	0.2023	0.1841	0.3863
0.8000	0.2378	0.2409	0.4786
1.0000	0.2500	0.2939	0.5439
1.2000	0.2378	0.3423	0.5800
1.4000	0.2023	0.3853	0.5875
1.6000	0.1469	0.4222	0.5691
1.8000	0.0773	0.4524	0.5297

112

合成波的最大振幅为2×0.5875ft，即1.175ft。

4. 算法设计

开发算法的第一步是将问题的解决方法分解为一系列有序连贯的步骤：

分解提纲

1）读入两个需要合成的波的周期和振幅。

2）计算出两波的波长并输出。

3）计算出两波叠加所得合成波上的 200 个点并找出最大值。

4）输出最大值。

在分解提纲的第 3）步中需要用一个循环来实现计算出合成波的 200 个点。在该循环之前还需要计算出合成波的周期以而便于分析。因为假设了周期均为整数，所以可以使用周期的乘积作为合成波的周期。（实际值可能会比该值小。）我们需要仔细思考如何找到最大振幅。回想一下手动演算示例，当所有值都输出后找到最大值是容易的，但是在程序计算两波叠加的过程中，并不能同时得出所有数据，而只有当前循环中的数据信息。因此为了记录下最大值，必须声明一个单独的变量来存储它。每一次计算一个新的振幅时，都要将该值与暂存的最大值进行一次比较。如果新的值更大，则用它来替换之前的最大值。因此可以得出以下伪代码：

[提炼后的伪代码]

```
主函数：读取 wave 1 的周期和振幅
        读取 wave 2 的周期和振幅
        计算出每一个波的波长并打印结果
        设置叠加波的周期为 new period
        设置时间增量为 new period/200
        设置最大振幅 wavemax 为 0
        设置 time 为 0
        设置 steps 为 0
        while steps <= 199
          设置 sum 为 wave 1 + wave 2
          if sum > wavemax
            设置 wavemax 为 sum
          将 steps 加 1
        输出 wavemax
```

按照这段伪代码，就可以写出如下 C 语言代码：

```
/*-------------------------------------------------------------*/
/*  程序 chapter3_5                                            */
/*                                                             */
/*  本程序确定两个指定波合成的波的最大高度                      */
#include <stdio.h>
#include <math.h>
#define PI 3.141593

int main(void)
{
   /*  声明变量  */
   int k;
   double A1, A2, freq1, freq2, height1, height2, length1, length2;
   double T1, T2, w1, w2, sum, new_period, new_height, time_incr, t;
   double maxwave=0;

   /*  从键盘获取用户输入  */
```

```
    printf("Enter integer wave period (s) and wave height (ft) \n");
    printf("for wave 1: \n");
    scanf("%lf %lf",&T1,&height1);
    printf("Enter integer wave period (s) and wave height (ft) \n");
    printf("for wave 2: \n");
    scanf("%lf %lf",&T2,&height2);
    /*  确定共同波长  */
    length1 = 5.13*T1^T1;
    length2 = 5.13*T2^T2;
    printf("Wavelengths (in ft) are: %.2f %.2f \n",length1,length2);

    /*  确定合成波的周期  */
    new_period = T1*T2;

    /*  计算指定周期内合成波的200个点，并找到最大高度  */
    time_incr = new_period/200;
    A1 = height1/2;
    A2 = height2/2;
    freq1 = 1/T1;
    freq2 = 1/T2;
    for (k=0; k<=199; k++)
    {
       t = k*time_incr;
       w1 = A1*sin(2^PI*freq1^t);
       w2 = A2*sin(2^PI*freq2^t);
       sum = w1 + w2;
       if (sum > maxwave)
          maxwave = sum;
    }
    new_height = maxwave*2;

    /*  打印合成波的最大高度  */
    printf("Maximum combined wave height is %.2f ft \n",new_height);

    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
```

[114]

5. 测试

为了用之前手动演算的数据来测试程序，需要对程序做一些细微的调整让它只计算10个值。因此用以下代码替代 for 循环中的第一行：

```
for (k=0; k<=9; k++)
```

修改后的程序运行结果如下：

```
Enter integer wave period (s) and wave height (ft)
for wave 1:
4 0.5
Enter integer wave period (s) and wave height (ft)
for wave 2:
10 1
Wavelengths (in ft) are: 82.08 513.00
Maximum combined wave height is 1.18 ft
```

可以看出，计算结果与手动演算的一致，因此我们可以开始测试该程序。首先，必须把 for 循环语句改回原来的情况，以使其可以在循环中计算出200个值。该程序的输出如下：

```
Enter integer wave period (s) and wave height (ft)
for wave 1:
4 0.5
Enter integer wave period (s) and wave height (ft)
```

```
for wave 2:
10 1
wavelengths (in ft) are: 82.08 513.00
Maximum combined wave height is 1.46 ft
```

图 3-11 展示了该示例中的两个波及其叠加的情况。

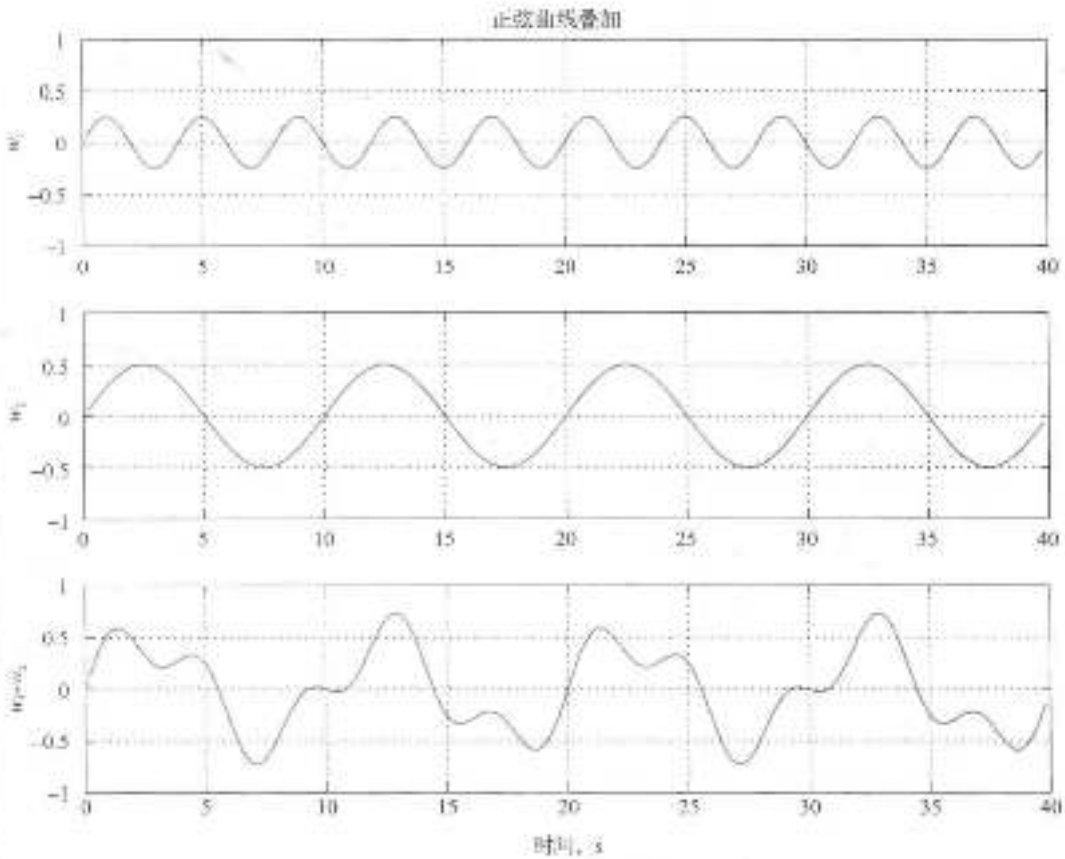


图 3-11 合成波的图像

修改

本节中计算振幅的最大值的程序中涉及了以下几个问题：

1. 修改程序，从而实现找到合成波的最大波峰值和最小波谷值。
2. 修改程序，从而实现让用户输入某个时间点，程序计算并输出该时间点时合成波的振幅。
3. 修改程序，从而实现让用户确定计算点的数量，并在原来限定的时间周期内计算叠加波的最大振幅。这会使答案与之前不同吗？请解释。
4. 修改程序，从而实现让用户可以设定两波之间的相位差。第一束波相位可以看作 0，这个相位差可以被当作第二束波的相位。用不同的相位差试验，看是否会得出不同的最大振幅。
5. 修改程序，将波长的单位从英尺换成米。记得修改程序从而能够准确计算出波长。

3.7 数据文件

工程问题的解决方案通常都包含大量数据。这些数据可能是由程序生成的输出数据，也

可能是程序所需的输入数据。然而，无论是在屏幕上打印出大批数据，还是从键盘读入大量数据都是不太可行的。所以通常需要使用数据文件来存储数据。这些数据文件与我们用来创建和存储C程序的程序文件相似。实际上，对于C语言编译器来说，一个C语言程序就是它的输入数据文件，而生成的目标程序就是C编译器的输出文件。本节将会对专门操作数据文件的C语句进行讨论，并且会给出数据文件的生成和读取的相关范例。

115 ~ 116

在调试有数据文件信息读取的程序时，使用回显命令echo（或print）来将读入的信息显示在屏幕上，以确认数据是被正确地读取了。如果显示数据的值全为0，或者是一些奇怪的数字，那么程序很可能找不到输入文件，这是因为程序无法访问对应的文件路径。此时解决办法有两种，一是将文件移动至程序能够读取到目录中；二是通过改变一些操作系统的参数，使程序能够找到文件。

在后文的例子中，将会使用包含传感器数据的文件进行读写操作。在这里假设传感器是地震仪。地震仪通常被埋在接近地表的地方并随时记录地球板块运动。这些传感器是极其敏感的，甚至可以在距离海洋数百英里的地方感知和记录潮汐运动。地震仪传感器遍布于世界各地以收集信息，然后通过卫星传送至中央区域的数据中心用以分析研究。科学家和工程师们试图通过研究这些振动信息和地震仪数据以预测地震。

3.7.1 输入/输出语句

程序中使用的每一个数据文件都必须要有一个与之关联的文件指针（file pointer）。如果一个程序使用了两个文件，那么这两个文件就分别需要不同的文件指针。文件指针的定义是通过FILE类型来声明的，如下所示：

```
FILE *sensor1;
```

上面的FILE数据类型是在头文件stdio.h中定义的，单词FILE必须是大写的，与头文件中的定义相对应。标识符前的星号*是说明该标识符是一个指针。后面的章节将会介绍更多关于指针的信息；在本节先只对文件指针相关的语句进行讲解。

在定义了一个文件指针之后，一定要为之关联一个指定的文件。使用fopen函数可以为一个文件指针分配指定的文件。该函数有两个参数，一个是文件名，另一个是说明该文件状态的字母，也称作文件打开模式（file open mode）；这两个参数都要使用双引号括住。在程序中如果要从一个文件中读取数据，那么文件打开模式为r，表示读文件。在程序中如果要将数据写入到一个文件里，那么文件打开模式为w，表示写文件。这样一来，下面的语句就是要指定一个文件指针sensor，关联到sensor的是一个名为sensor.txt的文件，我们将要从该文件中读取数据：

```
sensor = fopen("sensor1.txt","r");
```

如果一个数据文件无法打开，那么fopen函数会返回一个NULL值。（NULL是一个定义在<stdio.h>中的符号常量，其值为零字。）如果程序无法打开一个数据文件，比较常见的原因可能是程序无法找到该文件。因此，为了确保程序能准确找到对应的数据文件，一个好办法是去检查fopen函数的返回值，以确定文件被正确打开。下面的语句是打开一个关联了指针file1的文件；如果找不到该文件，屏幕上将会打印错误信息，同时if语句余下的部分将会自动跳过。

```
file1 = fopen(FILENAME,"r");
```

```
if (file1 == NULL)
   printf("Error opening input file \n");
else
{
   ...
}
```

117

如果你想确保程序能够找到数据文件，一个简单的方法是将数据存储在跟程序同一个文件夹下。

一旦输入文件和文件的指针被指定，就可以从该文件中读取数据，就像从键盘输入获取信息一样。不同之处在于，此时使用的函数是 fscanf，而不是 scanf。如果要对 sensor1.txt 文件的每一行都进行时间和传感器值的读取，可以通过下面的语句逐行进行读取，并把数据存储在变量 t 和 motion 中：

```
fscanf(sensor,"%lf %lf",&t,&motion);
```

注意，fscanf 函数与 scanf 函数的区别在于，fscanf 函数的第一个参数为文件指针。除此之外两个函数都是一样的。scanf 函数从键盘读取字符后将其转换为对应的值，而 fscanf 函数是将数据文件中的一行字符转换为对应的值。

如果是输出文件，可以通过函数 fprintf 将信息写入文件。fprintf 函数的第一个参数是文件指针，其余的参数则定义了变量以及写入文件的数据格式。例如，在本章前面讲到的计算两波叠加的程序，如果想要修改该程序，使之生成一个包含了输出数据的数据文件，可以使用一个文件指针 waves 来指向输出文件 waves1.txt，具体的做法如下面的语句所示：

```
FILE *waves;
...
waves = fopen("waves1.txt","w");
```

这样一来，在进行计算时，可以通过下面的语句来将结果写入输出文件中：

```
fprintf(waves,"%.2f %.2f %.2f %.2f\n","%f %f %f\n",
        t,w1,w2,sum);
```

其中的换行符的作用是在每组 4 个数值写入文件之后自动跳到新的一行。

在完成文件的读写操作之后，通常使用 fclose 函数来关闭文件；函数的参数就是文件指针。在上面的例子中，可以通过下面的语句来关闭这两个文件：

```
fclose(sensor);
fclose(waves);
```

在关闭输入文件和关闭输出文件之间没有差别。如果一个文件在主函数的 return 语句执行之后还没有被关闭，将由系统自动关闭文件。

如果经常使用同一个程序处理多个不同的文件，可以使用一个预处理命令来指定文件名。一旦需要修改程序中操作的文件名，修改预处理命令要比在语句中搜索 fopen 函数并逐一修改容易一些。下面的代码就是预处理命令和相应的 fopen 函数的一个例子：

```
#define FILENAME "sensor1.txt"
...
sensor = fopen(FILENAME,"r");
```

在本书的余下章节里，所有要用到文件操作的程序都会使用如上的组合语句。

118

3.7.2 读取数据文件

为了从一个数据文件中读取信息，就必须要了解一些文件的相关信息。显然文件名是必需的，这样才能用 fopen 语句将文件和指针关联在一起。当然文件存储的数据类型和顺序也是必须了解的，这样才可以正确声明相应的变量标识符以存储数据。最后，还需要知道文件中是否含有用于确定文件长度的特定信息。假设有一个文件已经全部读取完毕，如果此时再执行 fscanf 语句就会出错。为了避免这样的错误，就需要知道数据是何时全部读取完毕的。

数据文件一般有三种常见的结构。有些文件在被创建时，第一行会包含下面内容的总行数（每一行也称作一条记录）。例如，假设一个包含了传感器数据的文件拥有 150 组时间和传感器信息。这样的数据文件创建时就会在第一行写入数值 150，而余下的 150 行写入传感器数据。当需要读取该文件的数据时，首先就要读取文件第一行的数值，然后使用一个 for 循环来读取余下的信息。这种循环类型也称作计数循环。

文件结构的另一种形式是使用尾标记（trailer signal）或者哨兵标记（sentinel signal）。这些标记就是一些特殊的数值，用来标记一个文件中的最后一条记录。例如，一个具有标记的传感器数据文件包含有 150 行数据信息，在最后一行设置特殊标记，比如说时间和传感器的值都是 -999.0。为了避免数据混淆，这些标记必须与常规数据有明确的区分。在读取这种类型的文件时，通常使用 while 循环，只要数值不等于标记的取值条件就为 true。这类循环也称作标记控制循环。

第三种文件结构既没有在头一行包含有效数据量的记录，也没有使用尾标记或者哨兵标记。对于这种类型的数据文件，我们使用 fscanf 函数的返回值来确定文件何时读取完毕。在读取这类文件时，通常使用 while 循环，只要没到文件末尾条件就始终为 true。

由于有些操作系统对大小写敏感，所以在文件命名时一律采用小写字母以避免发生问题。在程序中使用的数据文件常常会发生更改，因此代码中使用的文件名应该易于查找和修改。因此，一般使用预处理命令来定义文件名；否则文件名就会嵌入程序中无法更改了。

下面将要给出一段程序，通过读取传感器数据来打印一份信息报告，包括读取的传感器的数量、数据的平均值、数据的最大值以及最小值。之前讨论的三种常见的文件格式将会在程序中使用到。

1. 指定记录数目

假设传感器数据文件的第一条记录是一个整数，它代表后面传感信息记录的总条数。下面的每一行数据都包含一对时间值和传感器数值，并且所有信息都从文件 sensor1.txt 中读取得来：

```
10
0.0  132.5
0.1  147.2
0.2  148.3
0.3  157.3
0.4  163.2
0.5  158.2
0.6  169.3
0.7  148.2
0.8  137.6
0.9  135.9
```

119

程序执行时首先读取数据点的个数，然后用它来指定要读取哪些时间点的数据，最后再计算数据信息。这个过程可以通过使用一个变量控制循环来轻松实现。在下面的伪代码和程序中，第一个数值将被用来初始化最大值 max 和最小值 min。如果直接将 min 设为 0，由于所有的传感器值都是大于 0 的，到最后程序将会错误地判定传感器最小值为 0。

根据上面的描述，得出伪代码和程序如下所示：

```
/*-------------------------------------------------------------*/
/*  程序 chapter3_6                                            */
/*                                                             */
/*  本程序的目的是根据一份数据文件而生成一个统计报告，该文件的第一行  */
/*  记录的是所有数据点的个数                                    */

#include <stdio.h>
#define FILENAME "sensor1.txt"
int main(void)
{
   /*  声明和初始化变量  */
   int num_data_pts, k;
   double time, motion, sum=0, max, min;
   FILE *sensor;

   /*  打开文件并读取数据点的个数  */
   sensor = fopen(FILENAME,"r");
   if (sensor == NULL)
      printf("Error opening input file.  \n");
   else
   {
      fscanf(sensor,"%d",&num_data_pts);
```

120

```
      /*  读取数据并计算统计信息  */
      for (k=1; k<=num_data_pts; k++)
      {
         fscanf(sensor,"%lf %lf",&time,&motion);
         if (k == 1)
            max = min = motion;
         sum += motion;
         if (motion > max)
            max = motion;
         if (motion < min)
            min = motion;
      }
      /*  打印统计信息  */
      printf("Number of sensor readings: %d \n",
             num_data_pts);
      printf("Average reading:           %.2f \n",
             sum/num_data_pts);
      printf("Maximum reading:           %.2f \n",max);
      printf("Minimum reading:           %.2f \n",min);
      /*  关闭文件并退出程序  */
      fclose(sensor);
   }
   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

程序执行后，根据文件 sensor1.txt 得出的统计报告应该打印如下：

```
Number of sensor readings: 10
Average reading:           149.77
Maximum reading:           169.30
Minimum reading:           132.50
```

121

2. 尾标记或哨兵标记

现有另一个数据文件 sensor2.txt，假设该文件中包含着与文件 sensor1.txt 相同的数据信息，唯一的区别是，并没有在文件的第一条记录给出有效数据记录的数目，而是在文件的最后一条记录中包含了一个尾标记。文件最后一行记录的时间是一个负值，这样便可以判断该条记录不是有效信息。而最后一行的第二个数据不能为空，因为读取数据的语句要求每行必须要读取两个值，不然程序就会报错。文件 sensor2.txt 中的数据如下所示：

```
0.0  132.5
0.1  147.2
0.2  148.3
0.3  157.3
0.4  163.2
0.5  158.2
0.6  169.3
0.7  148.2
0.8  137.6
0.9  135.9
-99  -99
```

现在要做的工作是读取并收集数据信息，直到遍历到文件末尾的尾标记为止。这个过程

可以通过使用一个 do/while 循环结构轻松实现，实现过程的伪代码及程序如下所示：

```
[提炼后的伪代码]
主函数：将 sum 置为零
        将点的个数置为零
        if 文件无法打开
          打印错误信息
        else
          读取 time, motion
          将 max 置为 motion
          将 min 置为 motion
          do
            将 motion 加进 sum 中
            if motion > max
              将 max 置为 motion
            if motion < min
              将 min 置为 motion
            点的个数的值自增 1
            读取 time, motion
          while time ≥ 0
          将 average 置为 sum/数据点的个数
          打印 average, max, min
```

122

```
/*-----------------------------------------------------------------*/
/*  程序 chapter3_7                                                */
/*                                                                 */
/*  本程序的目的是根据一份数据文件生成一个统计报告，该文件包含一个  */
/*  哨标记，其值为负数                                             */

#include <stdio.h>
#define FILENAME "sensor2.txt"

int main(void)
{
   /*  声明和初始化变量  */
   int num_data_pts=0;
   double time, motion, sum=0, max, min;
   FILE *sensor;

   /*  打开文件并读取第一组数据点  */
   sensor = fopen(FILENAME,"r");
   if (sensor == NULL)
      printf("Error opening input file.  \n");
   else
   {
      fscanf(sensor,"%lf %lf",&time,&motion);

      /*  用第一组数据点来初始化变量  */
      max = min = motion;

      /*  更新统计数据一直到遇到哨标记为止  */
      do
      {
         sum += motion;
         if (motion > max)
            max = motion;
```

```
            if (motion < min)
               min = motion;
            num_data_pts++;
            fscanf(sensor,"%lf %lf",&time,&motion);
         } while (time >= 0);

         /*  打印统计信息  */
         printf("Number of sensor readings: %d \n",
                num_data_pts);
         printf("Average reading:           %.2f \n",
                sum/num_data_pts);
         printf("Maximum reading:           %.2f \n",max);
         printf("Minimum reading:           %.2f \n",min);

         /*  关闭文件  */
         fclose(sensor);
      }

      /*  退出程序  */
      return 0;
   }
   /*-----------------------------------------------------------------*/
```

123

程序执行后，根据文件 sensor2.txt 打印出的统计报告同前面由文件 sensor1.txt 打印出的报告是完全一致的。

3. 文件结束标识符

每个数据文件的末尾都有一个文件结束标识符（end-of-file indicator），可以使用标准 C 库中的 feof 函数来检测何时会遍历到该标识符。同时，fscanf 函数也同样可以检测到何时遍历到文件的最后一组数据。由于 fscanf 函数每执行一次都会返回成功读取数据的个数，因此，如果函数返回了一个与预想中读取的数据个数不一样的值，那么就说明已经遍历到了文件末尾，或者文件中的数据出错。所以，如果一个数据文件中都是有效信息，那么调用 fscanf 函数就能够确定何时遍历到文件的末尾。现在考虑下面这段语句：

```
while ((fscanf(data,"%lf",&x)) == 1)
{
   count++;
   sum += x;
}
ave = sum/count;
```

fscanf 函数试图从一个数据文件中读取一个值，然后将值赋给变量 x。数据读取之后，函数返回值为 1，随后循环体中的语句将被执行。如果已经遍历到了文件末尾，便没有数据可供读取了；如此一来函数返回值便不为 1，并继续执行 while 循环体后面的语句。

现在假设数据文件 sensor3.txt 中包含的信息与文件 sensor2.txt 完全相同，只是在 sensor3.txt 中不包含尾标记。在下面的伪代码及程序当中，将读取并收集该文件中的数据信息，直到遍历到文件末尾：

```
[最终版的伪代码]
主函数：将sum置为零
        将数据点的个数置为零
        if 文件无法打开
           打印错误信息
        else
```

```
while 没有读取到文件末尾
  读取 time, motion
  将数据点的个数加 1
  if k=1
    将 max 置为 motion
    将 min 置为 motion
  将 motion 加到 sum 中
  if motion > max
    将 max 置为 motion
  if motion < min
    将 min 置为 motion
将 average 置为 sum/数据点的个数
打印 average, max, min
```

124

```
/*-----------------------------------------------------------------*/
/*  程序 chapter3_8                                                */
/*                                                                 */
/*  本程序的目的是根据一份数据文件生成一个统计报告，该文件缺没有   */
/*  头记录也没有尾记录                                             */

#include <stdio.h>
#define FILENAME "sensor3.txt"

int main(void)
{
   /*  声明和初始化变量  */
   int num_data_pts=0;
   double time, motion, sum=0, max, min;
   FILE *sensor;

   /*  打开文件  */
   sensor = fopen(FILENAME,"r");
   if (sensor == NULL)
      printf("Error opening input file.  \n");
   else
   {
      /*  没有读取到文件末尾  */
      /*  读取并收集数据信息  */
      while ((fscanf(sensor,"%lf %lf",&time,&motion)) == 2)
      {
         num_data_pts++;

         /*  使用第一组数据点来初始化变量  */
         if (num_data_pts == 1)
            max = min = motion;

         /*  更新统计数据  */
         sum += motion;
         if (motion > max)
            max = motion;
         if (motion < min)
            min = motion;
      }
      /*  打印统计信息  */
      printf("Number of sensor readings: %d \n",
             num_data_pts);
      printf("Average reading:           %.2f \n",
             sum/num_data_pts);
      printf("Maximum reading:           %.2f \n",max);
```

```
        printf("Minimum reading:          %.2f \n",min);
        /*  关闭文件  */
        fclose(sensor);
    }
    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
```

125

程序执行后，根据文件 sensor3.txt 打印出的统计报告同前面根据文件 sensor1.txt 和 sensor2.txt 打印出的报告是完全一致的。

上面介绍的这三种文件结构在工程问题及科研应用中广泛使用。因此，在工作中要事先了解到正在使用的数据文件是哪种结构类型。如果搞错了文件类型，程序很可能会得出一个错误的结果而不是错误信息。有些时候只能通过打印文件的开头几行或者末尾几行的内容来确认文件结构。

修改

在本章设计的两个程序中，其循环体包含了一个条件检查语句，在第一次执行循环体时测试该条件，当条件为真时，将 max 和 min 初始化为第一个 motion 值。现在设想一下，如果程序使用的数据文件非常长，那么这个条件检查语句的执行时间就会相当可观。一个避免执行条件检查的方法就是，在进入循环体之前先读取第一组数据点并初始化变量。这种改动也可能会引起程序其他地方的变动。

1. 修改程序 chapter3_5，删除循环体第一次执行时检测的条件语句。
2. 修改程序 chapter3_7，删除循环体第一次执行时检测的条件语句。

3.7.3 生成数据文件

生成一个数据文件与显示一条提示信息类似；但是不同之处在于，后者是在终端屏幕上显示信息，而前者是将数据信息写入文件中。在生成数据文件之前，必须确定要使用哪种文件结构。在前面的讨论中，介绍了三种最常见的文件结构——第一种，文件在开头给出了有效数据条数的初始记录；第二种，文件在末尾用尾标记或哨兵标记来表示有效数据的结尾；第三种，文件只包含有效数据，在开头或结尾都没有特殊标记。

在上面讨论过的三种文件结构各有优缺点。如果要在文件开头给出真实数据行数的初始记录，在生成文件之前就必须要知道文件中有多少行数据。而这个数字往往不是太容易计算的。所以带有尾标记的文件使用起来会非常简单，但是在报尾标记值的选择上必须要非常谨慎，要避免选择的标记值出现在文件的有效数据中。因此，最简单的还是生成那种只包含有效信息的数据文件，文件的开头和结尾不做任何特殊标记。如果输出的文件信息需要用作曲线绘制等软件工具的输入，最好使用第三种文件结构，即只包含有效数据信息的文件。

下面对本章早些时候给出的程序来做第四次程序修改。先前的那段程序是确定两组波浪的波长，然后计算在相位差为 0 的情况下两组波浪叠加之后的最大波峰值。在下面这段程序里，会将波浪的相关信息写入数据文件里，包括两组波及相互叠加之后的周期和振幅。观察下面的程序，并同 3.6 节中的程序作比较。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter3_9                                                */
/*                                                                 */
/*  本程序计算两组插走波浪的组合波的最大振幅                        */
```

126

```
#include <stdio.h>
#include <math.h>
#define PI 3.141593
#define FILENAME "waves1.txt"

int main(void)
{
   /*  声明变量  */
   int k;
   double A1, A2, freq1, freq2, height1, height2, length1, length2;
   double T1, T2, w1, w2, sum, new_period, new_height, time_incr, t;
   double maxwave=0;
   FILE *waves;

   /*  打开输出文件  */
   waves = fopen(FILENAME,"w");

   /*  通过键盘输入数据  */
   printf("Enter integer wave period (s) and wave height (ft) \n");
   printf("for wave 1: \n");
   scanf("%lf %lf",&T1,&height1);
   printf("Enter integer wave period (s) and wave height (ft) \n");
   printf("for wave 2: \n");
   scanf("%lf %lf",&T2,&height2);

   /*  计算并打印波长  */
   length1 = 5.13*T1*T1;
   length2 = 5.13*T2*T2;
   printf("wavelengths (in ft) are: %.2f %.2f \n",length1,length2);

   /*  计算组合波的周期  */
   new_period = T1*T2;

   /*  计算在指定的周期中组合波的200个数据点，并找出最大高度  */
   time_incr = new_period/200;
   A1 = height1/2;
   A2 = height2/2;
   freq1 = 1/T1;
   freq2 = 1/T2;
   for (k=0; k<=199; k++)
   {
      t = k*time_incr;
      w1 = A1*sin(2*PI*freq1*t);
      w2 = A2*sin(2*PI*freq2*t);
      sum = w1 + w2;
      fprintf(waves,"%.4f %.4f %.4f %.4f \n",t,w1,w2,sum);
      if (sum > maxwave)
         maxwave = sum;
   }
   new_height = maxwave*2;
   /*  打印组合波的最大高度  */
   printf("Maximum combined wave height is %.2f ft \n",new_height);

   /*  关闭文件并退出程序  */
   fclose(waves);
   return 0;
}
/*-----------------------------------------------------------------*/
```

127

将示例中的数值作为程序的输入，生成数据文件的前几行信息如下所示：

```
0.0000    0.0000    0.0000    0.0000
0.2000    0.0773    0.0627    0.1399
0.4000    0.1469    0.1243    0.2713
0.6000    0.2023    0.1841    0.3863
```

该文件的数据格式可以用 MATLAB 软件（将在附录 C 中进行介绍）绘制出相应的曲线，得出类似图 3-11 所示的数据分布图。

修改

1. 修改程序 chapter3_8，使其生成的文件中最后一行的 4 个数据为负值。
2. 修改程序 chapter3_8，使其生成的文件中第一行为全部有效数据的行数值。

*3.8 数值方法：线性建模

线性建模（linear modeling）就是要用一条直线描述一组数据点的分布情况，确定一个线性方程，使得该线性方程的直线与这组数据点的距离的平方和达到最小，这个过程就叫线性建模（也叫作线性回归（linear regression））。在 2.6 节中由新引擎的汽缸盖上得到了一组时间－温度值，为了更好地理解线性建模，下面首先考虑这组数据：

时间，s	温度，℉
0.0	0.0
1.0	20.0
2.0	60.0
3.0	68.0
4.0	77.0
5.0	110.0

128

如果画出这些数据点的分布情况，就会发现点的分布非常接近一条直线。实际上，如果计算出相应的斜率和到 y 轴的截距，就能画出一条刚好穿越所有数据点的直线。图 3-12 展示了这些数据点（x 轴为时间，y 轴为温度）和对应直线的分布情况。而该直线的方程为

$$y = 20x$$

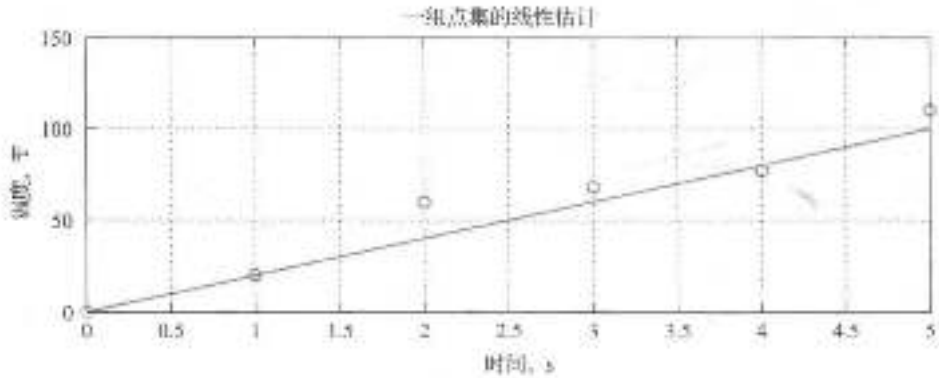


图 3-12 对一组点集的线性建模

为了检验对上述线性估计的准确度，首先要确定各点到线性估计之间的垂直距离；这

些距离如图 3-13 所示。从图中可以看到，前两个点准确地落在了直线上，所以 d_1 和 d_2 都为 0，d_3 的值为 20.0，即（60.0−40.0），而余下的垂直距离可以同理计算。如果要计算这些距离之和，有些正值和负值会因为互相抵消掉，从而使得到的总和比原本应该得的值要小。为了避免出现这种问题，通常都取绝对值或平方值来相加；线性回归一般使用平方值。因此，对线性估计准确度的检验应该要计算数据点到估算出的直线之间距离的平方和。很容易就可以计算出该平方和为 573。

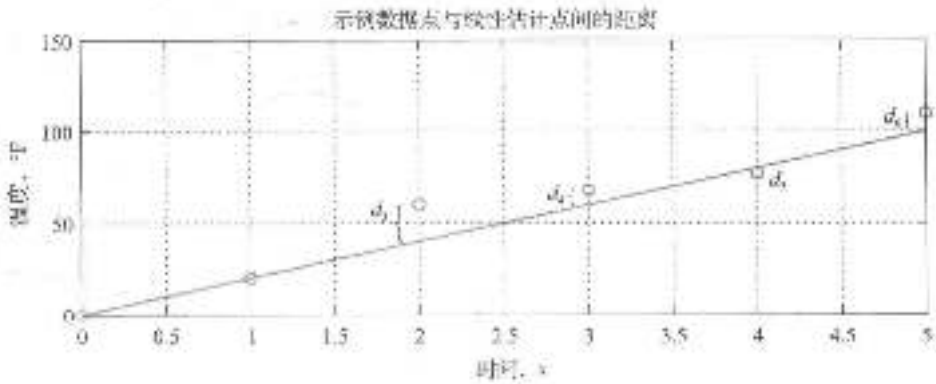


图 3-13 真实点与线性估计点间的距离

如果穿过这些点再画出一条直线，同样也可以计算这些点到这条直线距离的平方和。两条直线中具有更小平方和的一条显然就更加贴近数据点。为了找到这条具有最小距离平方和的直线，我们从一个直线方程的一般表达式展开分析。

$$y = mx + b$$

据此便可以得出所给数据点到该一般表达式之间距离的平方和。使用微积分的方法，便可以计算出该方程关于 m 和 b 的导数方程。令导数方程为零，便可以求得 m 和 b 的值，于是，具有最小距离平方和（也称作最小二乘距离）的直线也就确定了。在给出关于 m 和 b 的方程之前，首先要定义求和符号（summation notation）。

在本节开头给出的这组数据可以用下面的点集来表示 $(x_1, y_1), (x_2, y_2), \cdots, (x_6, y_6)$。符号∑代表求和；因此，$x$ 坐标之和就可以用下面的符号表示：

$$\sum_{k=1}^{6} x_k$$

129

这个求和是计算 k 从 1 变到 6 时，x_k 的累加总和。示例数据点集的求和计算结果是（0+1+2+3+4+5），即 15。其他的求和要使用下面的示例数据点集来计算：

$$\sum_{k=1}^{6} y_k = 0 + 20 + 60 + 68 + 77 + 110 = 335$$

$$\sum_{k=1}^{6} y_k^2 = 0^2 + 20^2 + 60^2 + 68^2 + 77^2 + 110^2 = 26\,653$$

$$\sum_{k=1}^{6} x_k y_k = 0 \times 0 + 1 \times 20 + 2 \times 60 + 3 \times 68 + 4 \times 77 + 5 \times 110 = 1\,202$$

我们现在回到为一组点进行最佳线性拟合的问题，运用上述步骤和微积分计算的结果，

用最小二乘法为 n 个数据点集找到最佳线性拟合的斜率和 y 轴截距，如下所示：

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2} \tag{3.1}$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2} \tag{3.2}$$

对于以上示例点数据集的拟合，m 的最佳值是 20.83，b 的最佳值是 3.76。示例数据点集和它的最佳线性拟合方程如图 3-14 所示。和图 3-13 中用直线法计算出的距离平方和 573 相比较，用最小二乘法计算的最佳拟合的距离平方和是 356.82。

130

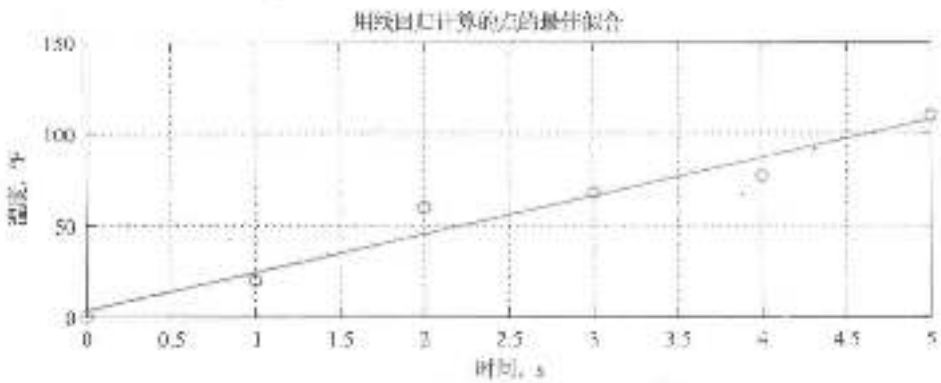


图 3-14 最小二乘线性回归模型

对于一组呈现线性规律的数据点集进行线性回归估算，就可以在没有数据时利用模型的线性性质来估计或者预测新的数据。例如，在气缸盖温度的例子中，假设我们想要估计在第 3.3 秒时气缸盖的温度。通过利用线性回归模拟的方程式，估计的温度是：

$$\begin{aligned} y &= mx + b \\ &= 20.83 \times 3.3 + 3.76 \\ &= 72.5 \end{aligned}$$

利用线性回归得到的方程模型，可以推测出线性插值不能计算的值。例如，利用线性回归模型可以估算第 8 秒的温度，但是不能利用线性插值计算第 8 秒的温度估计值，因为没有时间超过 8 秒的数据（要得到 8 秒之外的温度值，就是推断法（extrapolation）了，而不是插值法）。

需要注意的是，不是所有的数据都符合线性规律，因此不是都适合使用线性回归模型进行拟合。所以，在使用它推测新的数据点之前，首先要确定对于这些数据，线性模型是否是一个合适的模型。

在下一节中，我们将对卫星收集到的传感器数据进行最佳拟合，然后利用拟合的模型来估算或预测其他的传感器数据。

*3.9 解决应用问题：臭氧测量

卫星传感器可用于测量许多不同种类的信息，来帮助我们更多地了解地球周围大气层的分层结构。从地球表面开始，这些层分别是对流层、平流层、中间层、热层和散逸层。如图 3-15 所示。大气层的每一层有不同的温度特征。对流层（troposphere）是大气的最内层，大约在距离两极 5km，距离赤道 18km 处。绝大多数的云都是在这一层形成的。在对流层中随着高度的增加，温度以固定的速率下降。平流层（stratosphere）的特点是，不同的高度温度相对均衡，它的位置是从对流层延伸到地球上空大约 50km（约 31 英里）处。污染物可以在对流层中随着大气的变化稀释和消除，但是飘移到平流层的污染物在回到对流层之前会在平流层停留很多年。中间层（mesosphere）大约在地球上空 50 – 85km（约 53 英里）处，在这一层空气的对流运动很强。中间层之上是热层（thermosphere），位于地球上空 85 – 140km（约 87 英里）处，由于氧原子吸收了太阳能而使得这一层温度升高。电离层（ionosphere）是热层中带电粒子相对集中的区域，有些通信方式就是利用电离层对无线电波的反射来实现的。散逸层（exosphere）是大气层中最高的区域。散逸层中大气密度极低，大气分子向上运动时很容易逃离大气层而不会和其他分子相撞。

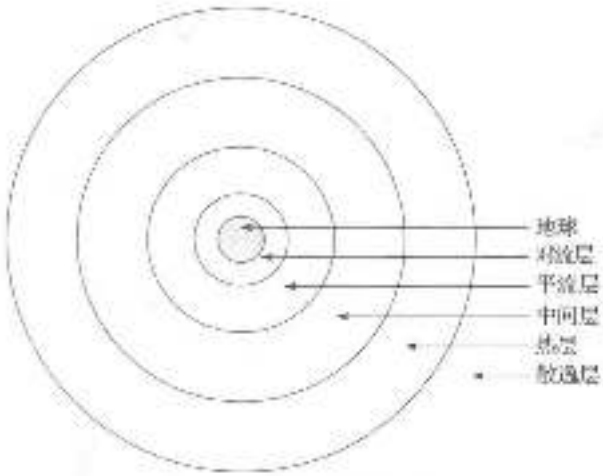


图 3-15 地球表面的大气层

[131]

下面考虑用一组测量数据描述每百万体积（ppmv）中的臭氧混合比例。在一个小范围中，这些数据近似线性，因此可以使用线性模型来估计某个高度的臭氧混合比例，而不用逐点去测定具体数值。编写一个程序，从文件 zone1.txt 中读取数据，这些数据是包括以 km 为单位的高度和对应高度的每百万体积臭氧混合比例，并且可以确定这些数据分布在线性模型区域中。zone1.txt 中仅包含有效数据，没有特殊的标题行或尾标记。运用上节提到的最小二乘法确定该模型并输出。除此之外，输出模型开始和结束的高度来指示该模型可以准确适用的区域。

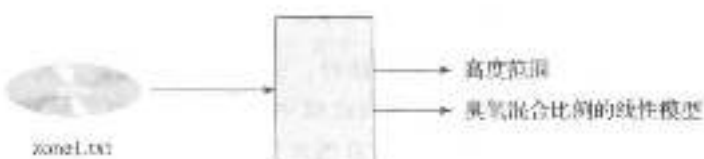
[132]

1. 问题陈述

运用最小二乘法确定一个线性模型，用来估算指定高度的臭氧混合比例。

2. 输入输出描述

下面的图表示程序的输入是数据文件 zone1.txt，输出是高度范围和线性模型：



3. 手动演算示例

假设有4组数据，如下表所示：

高度（km）	臭氧混合比例（ppmv）
20	3
24	4
26	5
28	6

现在需要代入数据计算公式（3.1）和公式（3.2）中，为了方便，将公式重复一遍：

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}$$

使用手动演算示例中的数据计算这两个公式，需要计算下面的这些累加和：

133

$$\sum_{k=1}^{4} x_k = 20 + 24 + 26 + 28 = 98$$

$$\sum_{k=1}^{4} y_k = 3 + 4 + 5 + 6 = 18$$

$$\sum_{k=1}^{4} x_k^2 = 20^2 + 24^2 + 26^2 + 28^2 = 2436$$

$$\sum_{k=1}^{4} x_k y_k = 20 \times 3 + 24 \times 4 + 26 \times 5 + 28 \times 6 = 454$$

用这些累加和可以计算出 m 和 b 值：

$$m = 0.37, \quad b = -4.6$$

4. 算法设计

首先根据问题列出分解提纲。

分解提纲

1）从数据文件中读取数值，并计算相应的累加值与范围。

2）计算斜率和 y 轴截距。

3）输出高度范围和线性模型。

在分解提纲步骤1）从文件中读取数据时，使用了一个循环，同时计算对应的累加值，为计算线性模型做好准备。读取文件的过程中也可以同时确定数据点集的数量。因为数据文件中没有头信息和尾信息，所以退出循环的条件是测试文件是否结束。由于要记录高度范围，所以需要保存第一个高度值和最后一个高度值。因此接下来的伪代码如下

所示：

```
[根据上面的伪代码]
主函数：将 count 设置为 0
        将 sumx, sumy, sumxy, sumx2 设置为 0
        if 数据文件不能打开
          输出错误提示信息
        else
          while 没有遇到文件末尾
            读取 x, y
            count 增加 1
            if count = 1
              将 x 的值赋给 first
            将 sumx 的值增加 x
            将 sumy 的值增加 y
            将 sumx2 的值增加 x2
            将 sumxy 的值增加 xy
          将 x 的值赋给 last
          计算斜率和 y 轴截距
          输出 first, last, 斜率和 y 轴截距
```

134

伪代码中的步骤足够详细，可以将它转换为 C 语言。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter3_10                                               */
/*                                                                 */
/*  这个程序计算一组高度和臭氧混合比例的线性模型                    */

#include <stdio.h>
#define FILENAME "zone1.txt"

int main(void)
{
   /*  声明和初始化变量  */
   int count=0;
   double x, y, first, last, sumx=0, sumy=0, sumx2=0,
          sumxy=0, denominator, m, b;
   FILE *zone;

   /*  打开输入文件  */
   zone = fopen(FILENAME,"r");
   if (zone == NULL)
      printf("Error opening input file.  \n");
   else
   {
      /*  当没有遍历到文件末尾时，读取并计算相应变量  */
      while ((fscanf(zone,"%lf %lf",&x,&y)) == 2)
      {
         ++count;
         if (count == 1)
            first = x;
         sumx += x;
```

```
            sumy += y;
            sumx2 += x*x;
            sumxy += x*y;
        }
        last = x;

        /* 计算斜率和y轴截距 */
        denominator = sumx*sumx - count*sumx2;
        m = (sumx*sumy - count*sumxy)/denominator;
        b = (sumx*sumxy - sumx2*sumy)/denominator;
        /* 输出总结信息 */
        printf("Range of altitudes in km: \n");
        printf("%.2f to %.2f \n\n",first,last);
        printf("Linear model: \n");
        printf("ozone-mix-ratio = %.2f altitude + %.2f \n",
               m,b);
        /* 关闭文件 */
        fclose(zone);
    }
    /* 退出程序 */
    return 0;
}
/*-----------------------------------------------------------*/
```

135

5. 测试

将手动演算示例中的数据作为数据文件 zone1.txt 中的内容，得到如下程序输出：

```
Range of altitudes in km:
20.00 to 28.00

Linear model:
ozone-mix-ratio = 0.37 altitude + -4.60
```

这和手动演算示例的结果相一致。

修改

这些问题和本节主要讨论的问题相关。如果需要，可以参考转换公式 1 千米 = 0.621 英里。

1. 添加语句，使得程序可以输入以千米为单位的高度，使用模型估计相应的臭氧混合比例。
2. 修改问题 1 中的程序，使程序检查输入的高度是否符合模型的高度范围。
3. 修改问题 2 中的程序，使程序允许输入以英里为单位的高度（程序应该将英里转换为千米）。
4. 修改原始程序，使程序输出线性模型，并且此线性模型中的高度是以英里为单位而不是千米。假设数据文件中的高度依然是以千米为单位。

136

本章小结

在本章中，我们学习使用条件语句 if 来选择应该执行的语句，也学习了用循环来重复执行一组语句的方法。循环语句可以使用 while 或者 for 实现。条件结构和重复结构在几乎所有的程序中都会被用到。此外，我们介绍了从数据文件中读取信息的基本语句，以便在程序中使用这些语句读取信息。我们还学习了在程序中生成数据文件的语句。数据文件在解决工程问题时被大量使用，这在前面就已经提到过。在后面的许多问题解决中还会使用。最后，我们介绍了为一组数据点集创建线性模型的概念，并且运用最小二乘法确定拟合度最佳的方程。

关键术语

case label（case 标签）
case structure（选择结构）
compound statement（复合语句）
condition（条件）
conditional operator（条件运算符）
controlling expression（控制表达式）
data file（数据文件）
default label（default 标签）
divide and conquer（分治）
empty statement（空语句）
end-of-file indicator（文件结束指示符）
error condition（错误条件）
file open mode（文件打开方式）
file pointer（文件指针）
flowchart（流程图）
for loop（for 循环）
iteration（迭代）
least squares（最小二乘法）
linear modeling（线性模型）
linear regression（线性回归）
logical operator（逻辑运算符）
loop（循环）
loop control variable（循环控制变量）
program walkthrough（程序走查）
pseudocode（伪代码）
relational operator（关系运算符）
repetition（重复）
selection（选择）
sequence（顺序）
sentinel signal（哨兵标记）
stepwise refinement（逐步求精）
structured program（结构化程序）
summation notation（求和标记）
test data（测试数据）
top-down design（自上而下的设计）
trailer signal（尾标志）
validation and verification（验证和评价）
while loop（while 循环）

C 语句总结

文件指针的声明：

```
FILE *sensor;
```

if 语句：

```
if (temp > 100)
    printf("Temperature exceeds limit \n");
```

```
temp>100 ? printf("Caution \n"): printf("Normal \n");
```

if/else 语句：

```
if (d <= 30)
   velocity = 4.25 + 0.00175*d*d;
else
   velocity = 0.65 + 0.12*d - 0.0025*d*d;
```

[137]

switch 语句：

```
switch (op_code)
{
   case 'n': case 'r':
      printf("Normal operating range \n");
      break;
   case 'm':
      printf("Maintenance needed \n");
      break;
   default:
```

```
        printf("Error in code value \n");
        break;
}
```

while 循环：

```
while (degrees <= 360)
{
   radians = degrees*PI/180;
   printf("%6.0f %9.6f \n",degrees,radians);
   degrees += 10;
}
```

do/while 循环：

```
do
{
   radians = degrees*PI/180;
   printf("%6.0f %9.6f \n",degrees,radians);
   degrees += 10;
} while (degrees <= 360);
```

for 循环：

```
for (degrees=0; degrees<=360; degrees+=10)
{
   radians = degrees*PI/180;
   printf("%6.0f %9.6f \n",degrees,radians);
}
```

break 语句：

```
break;
```

continue 语句：

```
continue;
```

文件打开：

138

```
sensor = fopen("sensor1.txt","r");
waves = fopen(FILENAME,"w");
```

文件输入：

```
fscanf(sensor,"%lf %lf",&t,&motion);
```

文件输出：

```
fprintf(waves,"%.2f %.2f %.2f %.2f \n",t,w1,w2,sum);
```

文件关闭：

```
fclose(sensor);
```

注意事项

1. 简单条件中，在逻辑表达式中关系运算符周围使用空格；复杂条件中，在逻辑运算符周围使用空格，而关系运算符周围不使用空格。
2. 复合语句或循环语句内部需要添加缩进。如果循环或复合语句是嵌套的，要相对于前一条语句，按照嵌套的级别设置缩进。
3. 使用花括号分隔复杂语句的结构，即使程序逻辑没有错误，也需要通过这种分隔使程序清晰可读。

4. 在 switch 语句中，一定要使用 default 语句，用来强调没有 case 标签与控制语句匹配时程序所要执行的语句。
5. 花括号单独占一行，使得循环主体容易辨认。
6. 用预处理指令定义文件名称，使文件名修改更加容易。

调试注意事项

1. 当你在程序中发现错误并改正后，要重新对程序进行一次测试，并保证原来所有使用过的测试数据都能正常工作。
2. 在条件式中，一定要使用关系运算符“==”而不是“=”。
3. 在语句块周围使用花括号，并且花括号单独占一行，可以使程序更加清晰。
4. 浮点型数值不要使用等号运算符，而是测试它是否与期望值足够接近。
5. 改正语法错误时，要经常重新编译程序。改正一条错误，可能会消除许多错误信息。
6. 调试循环时使用 printf 语句打印内存快照，以获得主要变量的值。
7. 一定要知道当程序进入死循环时，你的系统强制退出程序用的特殊字符。编程时死循环出现的频率很高。
8. 调试从数据文件读取信息的程序时，一边读一边输出读取的信息。这可以帮助你检查读取信息时出现的错误。
9. 调试从数据文件读取信息的程序时，确认你的程序可以访问包含数据文件的目录。
10. 为了避免操作系统不区分大小写的问题，用小写字母给文件命名。 139

习题

简述题

判断题

判断下列语句的正（T）误（F）。

1. 如果一个条件的值是 0，则这个条件被认为是错的。 T F
2. 如果一个条件的值既不是 0 也不是 1，则它是无效条件。 T F
3. 表达式 a==2 用来确定 a 的值是否等于 2，表达式 a=2 将变量 a 的值赋值为 2。 T F
4. 逻辑运算符 && 和 || 优先级相同。 T F
5. 除非用花括号来划分块语句，否则关键字 else 总是和离它最近的 if 语句匹配。 T F
6. 调试循环时，可以在循环中运用 printf 语句来输出变量的内存快照。 T F

语法题

找出下列语句中的语法错误（假设所有的变量都已声明，而且类型是整数型）：

7. for (b=1, b=<25, b++)

8. while (k=1)

9. switch (sqrt(x))

```
{
   case 1:
      printf("Too low. \n");
      break;
   case 2:
      printf("Correct range. \n");
      break;
   case 3:
```

```
        printf("Too high. \n");
        break;
  }
```

多选题

选择每个问题的最佳答案

10. 若有语句：

```
int i=100, j=0;
```

则下面（　　）表述正确。

(a) i<3　　(b) !(j<1)

140 (c) (i>0) || (j>50)　　(d) (j<i) && (i<=10)

11. 若 a1 为 true，a2 为 false，则下面（　　）表达式的结果为 true。

(a) a1 && a2　　(b) a1 || a2

(c) !(a1 || a2)　　(d) !a1 && a2

12. 下面（　　）是单目运算符。

(a) ;　　(b) ||

(c) &&

13. 表达式 (!((3-4%3) < 5 && (6/4 >3))) 的结果是（　　）。

(a) true　　(b) false

(c) 无效　　(d) 以上都不对

问题 14 ~ 16 基于下面的语句来选择答案

```
int sum=0, count;
...
for (count=0; count<=4; count++)
   sum += count;
printf("sum= %i \n",sum);
```

14. 屏幕上的输出是（　　）。

(a) sum = 1　　(b) sum = 6

(c) sum = 10　　(d) 错误信息

15. 循环执行完之后 count 的值是（　　）。

(a) 0　　(b) 4

(c) 5　　(d) 未知整数

16. 循环内的语句执行了（　　）次。

(a) 0　　(b) 4

(c) 5　　(d) 6

内存快照题

给出下列每组语句执行后其对应的内存快照

17.
```
int a = 750;
...
if (a>0)
   if (a >= 1000)
      a = 0;
   else
      if  (a < 500)
          a *= 2;
```

```
        else
            a *= 10;
  else
    a += 3;
```

141

编程题

单位换算。下面的问题要求生成单位转换表，包括一个表头和列标题，并为输出值选择合适精度。

18. 生成弧度到角度的转换表。弧度列从 0.0 开始，每次增加 π/10，直到 2π。
19. 生成角度到弧度的转换表。第一行是 0°，最后一行是 360°。允许用户输入转换表各行之间的递增量。
20. 生成英寸到厘米的转换表。英寸列从 0.0 开始，每次增加 0.5 英寸。最后一行是 20.0 英寸。（1 英寸 =2.54 厘米）
21. 生成英里 / 小时到英尺 / 秒的转换表。英里 / 小时列从 0 开始，每次增加 5 英里 / 小时。最后一行是 65 英里 / 小时。（1 英里 =5280 英尺）
22. 生成英尺 / 秒到英里 / 小时的转换表。英尺 / 秒列从 0 开始，每次增加 5 英尺 / 秒。最后一行是 100 英尺 / 秒。（1 英里 =5280 英尺）

货币换算。下面的问题要求生成货币转换表，并为表格生成表头和列标题。假设汇率如下：

1 美元（$）= 0.737 938 欧元（Europe）

1 日元（¥）= 0.013 005 美元

1 美元（$）=0.632 293 英镑（£，UK）

23. 生成欧元到美元的转换表。欧元列从 5 欧元开始，每次增加 5 欧元。输出转换表的前 25 行。
24. 生成英镑到美元的转换表。英镑列从 1 英镑开始，每次增加 2 英镑。输出转换表的前 30 行。
25. 生成日元到英镑的转换表。日元列从 100 日元开始。输出 25 行，最后一行是 10 000 日元。
26. 生成美元到欧元、日元和英镑的转换表。从 1 美元开始，每次增加 1 美元，输出转换表的前 50 行。 142

温度换算。下列问题生成温度转换表。使用下列华氏温度（T_F）、摄氏温度（T_C）、开氏温度（T_K）和兰氏温度（T_R）间的转换关系。

$$T_F = T_R - 459.67°\text{R}$$
$$T_F = (9/5)T_C + 32°\text{F}$$
$$T_R = (9/5)T_K$$

27. 编写程序，生成华氏温度到摄氏温度的转换表，表中华氏温度范围是从 0°F – 100°F，每行增加 5°F。解决方案中使用 while 循环。
28. 编写程序，生成华氏温度到开氏温度的转换表，表中华氏温度范围是从 0°F – 200°F，允许用户输入行之间的华氏温度增量。解决方案中使用 do while 循环。
29. 编写程序，生成摄氏温度到兰氏温度的转换表。允许用户输入开始摄氏温度和行之间的增量，输出表中包含 25 行。解决方案中使用 for 循环。

探测火箭轨迹。探测火箭用来探测大气层中不同层的状况，并且收集信息（例如测大气中的臭氧含量）。除了携带科学设备用以收集外大气层中的数据外，探测火箭还装载了遥测系统向发射基地传输科学数据。与此同时探测火箭还会传输自身的性能遥测数据，以使安全人员监督火箭，并供工程师后期分析。这些性能数据包括高度、速度和加速度。假设这些性能信息存储在一个文件中，并且每一行包含 4 个数值——时间、高度、速度和加速度，单位分别是 s、m、m/s 和 m/s²。

30. 假设文件 rocket1.txt 的第一行包含数据的总行数。编写程序，读取这些数据，并且确定火箭开

始落回地面的时间。（提示：开始回落的时间即高度开始减少的时间）

31. 当火箭的速度增加到一定峰值然后开始减少，这是火箭的某一级耗尽后脱落造成的。通过观察这种速度变化出现的次数，就可以判断火箭的级数。编写程序，读取数据，并且确定火箭的级数。使用文件 rocket2.txt 中的数据。该文件名包含尾标记，尾标记行中的4个数据都是 -99。

32. 修改问题31中的程序，使其输出火箭每级发射对应的时间点。假设某级火箭发射对应的时间点就是速度开始增加的时间点。

33. 在每级火箭发射之后，加速度将开始增加，推进力减小后逐步恢复成竖直向下的重力加速度，即 $-9.8m/s^2$。找到火箭仅有重力加速度作用的飞行时间段。由于各种数据箱差，加速度值在理论值的65%范围内的时间都记录在时间段内。使用数据文件 rocket3.txt。该文件中不包含初始行信息和尾行信息。

143

缝合线封装。缝合线是用来在受伤或者手术后将活组织缝合在一起的线或者纤维。缝合线的包装在运送医院之前必须小心封好，以防止污染物进入其中。用来封装包装的物品被称为封装模具。封装模具需要由电热器加热后用于密封。要使封装过程成功，需要将封装模具维持在设定的温度，并且用预设压力使封装模具接触包裹并保持一个特定的时间。封装模具接触包裹的时间称为压合时间。假设一个合格封装的可接受参数范围如下：

温度：150℃ ~ 170℃

压力：60psi ~ 70psi

压合时间：2s ~ 2.5s

34. 数据文件 suture1.txt 包含了一周内检验不合格的一系列缝合线信息。数据文件的每一行都包含不合格批次的批号、温度、压力和压合时间。质检工程师必须分析这些信息，并且需要分别计算本批次中由于温度、压力、压合时间造成不合格的产品所占百分比。有的批次可能由于多种原因导致不合格，那么在每种原因的分类中都要对其计数。编写程序计算并输出这三个百分比。

35. 修改问题34中编写的程序，使它能够输出每种原因不合格批次的数目，和不合格批次的总数目。（注意每个不合格批次在总数中仅能出现一次，在不同原因的分类中可多次出现。）

36. 编写程序读取数据文件 suture1.txt，并确定其中的信息都是不合格批次的相关信息。如果数据文件中出现了不该出现在这里的批次信息，输出这些批次的相关信息并显示一个适当的提示语。

木材再生。木材管理中的一个问题是对于一片给定面积的区域，应该保留多少面积的禁伐林区，才能使得被砍伐区域在一个指定时间内能够再生为森林。假设植被再生依赖于气候和土壤条件，每年以一个已知比率再生。再生林每年的生长量与保留的木材面积和再生率直接相关。这个关系被称为再生林公式。例如，砍伐后保留的森林面积为100英亩，再生率为0.05，那么第一年年末将有 100+（0.05 × 100），即105英亩被森林覆盖，第二年年末绿化面积为 105+（0.05 × 105），即110.25英亩。

37. 假设林区总面积为14 000英亩，2500英亩保留未砍伐，再生率为0.02。输出一个表格，显示20年内每年年末的植被覆盖面积。

144

38. 修改问题37中编写的程序，使用户可以输入表格中显示的总年数。

39. 修改问题37中编写的程序，使用户输入要砍伐的英亩数，并确定这些土地需要多少年可以被完全重新造林。

关键路径分析。关键路径分析是用来为工程确定时间安排的技术。这个信息在工程开始之前的规划阶段是非常重要的，在工程部分完成时对于工程进度的评估也很重要。一种分析方法是将工程分为若干顺序执行的事件，然后将每个事件划分为多个任务。虽然一个事件完成后才能开始下一个事件，但是一个事件中的多个任务是可以同时进行的。因此，一个事件的完成时间取决于完成最长任务所需天数，

而完成整个工程的总时间就是每个事件完成的时间总和。

假设有一个重点建设工程的关键路径信息存储在数据文件中，数据文件的每一行包括事件编号、任务编号和完成任务所需的天数。数据文件中，事件2的所有任务数据跟在事件1的所有任务数据之后，以此类推。示例数据如以下表格所示：

事件	任务	天数
1	15	3
1	27	6
1	36	4
2	15	5
3	18	4
3	26	1
4	15	2
4	26	7
4	27	7
5	16	4

40. 编写程序读取关键路径信息，并输出完成时间表，列出每个事件编号，事件中任务的最大完成天数以及项目完成的总天数。

41. 编写程序读取关键路径信息，并输出一份报告，列出事件编号以及事件中完成时间超过5天的任务编号。

42. 编写程序读取关键路径信息，并输出一份报告，列出每个事件编号和事件中的任务数量。

探空气球。探空气球用来收集大气中不同海拔地区的温度和气压数据。气球内部充满氦气，由于氦气的密度小于气球外的空气密度，所以气球能够上升。当气球上升时，周围空气密度变小，因此上升速度减缓直到内外密度达到平衡状态。白天时，阳光加热气球内部的氦气导致其膨胀，并且密度变小，因此气球能够升得更高。而在晚上，气球内部氦气温度变低，密度变大，因此气球下降到较低的海拔。第二天，阳光再次加热氦气，气球升起。随着时间的推移，这个过程记录产生一组测量数据，而刻量点的海拔高度值近似于一条多项式曲线。假设用下面的多项式表示探空气球发射后的48小时期间内，海拔高度随时间变化的函数。其中高度单位为米： 145

$$alt(t) = -0.12t^4 + 12t^3 - 380t^2 + 4100t + 220$$

t的单位为小时。气球相对应的以米/小时为单位的速度多项式模型为：

$$v(t) = -0.48t^3 + 36t^2 - 760t + 4100$$

图3-16是48小时期间内海拔和气球速度随时间变化的关系图，从图形中可以看出气球上升和下降的时间区间。

43. 编写程序输出海拔和探空气球速度表，其中单位分别为米和米/秒。用户输入开始时间、表格中每行的时间增量和结束时间。注意所有的时间值都要小于48小时。编写程序生成一个表格，显示气球发射4小时后，两个小时内的探空气球信息，时间间隔为10分钟。

44. 修改问题43中的程序，使其能够同时输出最高海拔和对应的时间。

45. 修改问题43中的程序，使其确定结束时间大于初始时间。如果不是，要求用户重新输入整套报告信息。

46. 上述方程仅适用于时间从0 – 48小时，因此修改问题43中编写的程序，使其输出信息，提示用户

输入时间上限为48小时。同时，检查用户输入，以确定它在指定界限中。如果存在错误，提示用户重新输入整套报告信息。

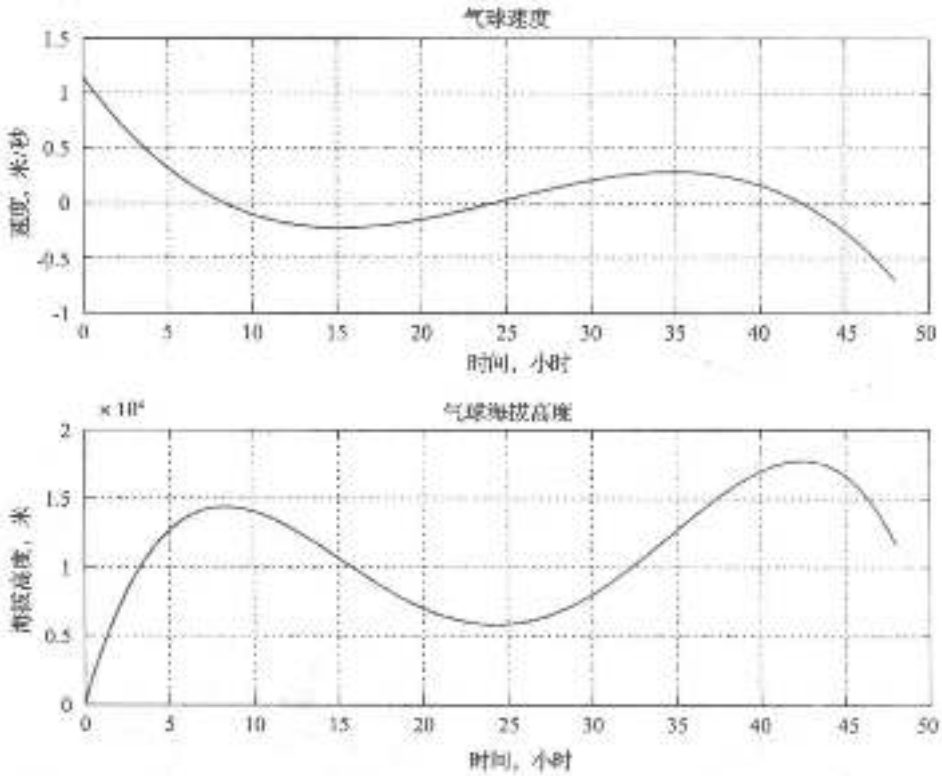


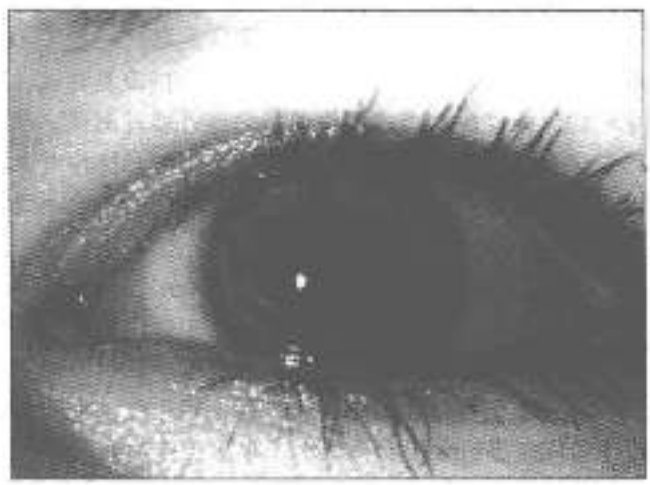
图3-16 探空气球的速度和海拔数据

146
~
147

47. 修改问题43中的程序，使其将时间、海拔和速度信息存储在数据文件 balloon1.txt 中。

用函数实现模块化程序设计

犯罪现场调查：虹膜识别



虹膜识别是基于虹膜的模式识别，其中虹膜是眼睛中带色的环形部分。虹膜识别处理的是由红外线照相机拍摄的眼球照片。红外线图像是黑白的，不包含颜色信息。但是，红外线图像可以捕获深褐色虹膜中的图案，这些是在可见光图像中无法捕获的。由于虹膜的结构复杂，虹膜识别成为最准确的生物测定方法之一。虹膜中的图案被称为条纹，是婴儿在子宫内发育过程中由胚胎撕裂而形成的。左虹膜和右虹膜完全不同，同卵双胞胎的虹膜也是不一样的。虹膜也是少数几个不随年龄而变化的生物特征之一。例如，脸部会随时间而变化，在孩子长大成人的过程中皆能会变化。虹膜识别已现在很多电影中，但是影视作品中的描述并不准确。例如，大多数虹膜识别系统都有活性测试，这意味着测试眼球是否属于活人。这些活性测试包括检查眼球抖动。所有眼睛都会有少量抖动，如果眼睛没有抖动，那么它就不是活的眼球。另一个活性测试通过改变采集系统的光照来进行。光照变化会导致瞳孔大小的变化，如果瞳孔大小没有改变，那么它就不是活的眼球。因此，像电影《少数派报告》中所做的简单眼球来欺骗虹膜识别系统，在现实生活中根本不会起作用。

[148]

学习目标

在本章，我们将要学到以下解决问题的方法：

- 标准 C 库中的模块。
- 自定义模块。
- 生成随机数的函数。
- 宏函数。
- 递归函数。
- 求解多项式实根的方法。

4.1 模块化

C 程序的执行从 main 函数中的语句开始，程序也会包含其他的函数，可能调用其他文件或者库中的函数。这些函数（function）或者模块（module），通常是能够执行某些操作或者计算某些数值的一组语句。例如，printf 函数能够在终端屏幕上输出一行信息，sqrt 函数能够计算一个值的平方根。

程序设计可以解决非常复杂的问题，代码也会写得很长，使用一个很长的 main 函数解决

问题会使得代码很难维护。为了保持解决方案的简洁性和可读性，我们使用main函数和附加函数组合的方法来开发程序。通过将解决方案分成一系列的模块，每个模块都更简洁和容易理解，这与第3章中介绍的结构化编程的基本方针保持一致。

149

在设计问题的解决方案时常常会用到“分治”的思想，就像第3章所介绍的分解提纲。所谓分解提纲就是一组连续的执行步骤，在编程时可以将步骤转化成函数。但实际上，分解提纲中的每个步骤并不一定都要对应main函数中的一个或者多个函数调用。

将一个解决方案分解成几个不同的模块有很多好处。由于每一个模块都要完成特定的功能，因此可以独立于方案的其他部分单独编写和测试。一个独立模块要比完整的解决方案小得多，所以测试起来也会容易得多。此外，一旦一个模块通过了认真的测试，那么该模块就可以用于新的问题解决方案之中而无需再重复测试。例如，假设现在设计了一个计算一组数据平均值的模块。当该模块编写完成并通过测试，那么当其他程序需要计算平均值时，就可以拿来直接使用。在设计大型软件系统时，这种可重用性（reusability）尤为重要，因为它可以大大节省开发时间。实际上，经常使用的模块通常都是以函数库的形式存在于计算机系统中，比如标准C库。

模块的使用（也叫作模块化（modularity））通常会减少程序的总长度，因为许多解决方案中都包含了在多个地方需要重复的步骤。将这些重复的步骤放在一个函数中，这些步骤就只需编写一次，在需要的时候只需一条单独的语句来引用。

将一个项目分成若干模块以后，每个单独模块都能被独立开发和测试，就可以允许多个程序员同时进行分工合作。由于很多工作都能同时进行，这就加快了项目的开发进度。

使用那些为了特定任务而编写的模块也符合抽象（abstract）的概念。模块中包含了这些任务的细节，而我们在使用这些模块时可以完全忽略这些细节。在开发解决方案时使用的I/O图就是抽象的一个例子。我们指出了输入和输出信息，而没有给出输出信息是如何确定的。类似的，可以将模块看作一个指定了输入信息，并计算特定信息的“黑盒”，我们可以利用这些模块开发解决方案。这样一来，就可以在更高的抽象层次上操作来解决问题。例如，标准C语言库中包含了一些计算对数的函数。编程中可以直接引用这些函数，而无须关注这些函数到底是使用了无穷级数还是通过查表来计算指定对数的。使用抽象的思想既可以显著缩短软件开发时间，又能提高工作质量。

作为总结，我们列出了在问题求解中使用模块的一些优势：

- 模块可以被独立编写和测试，因此在大型工程中的模块开发可以并行进行。
- 由于模块是整体方案的一小部分，所以进行独立测试相当方便。
- 一旦模块经过了认真的测试，应用到新的问题解决方案中时便无需重复测试。
- 使用模块可以减少程序的代码量，并增强可读性。
- 模块化思想进一步催生出抽象的概念，允许程序员将具体的实现细节隐藏在模块内部；进而允许我们从功能的角度使用模块，忽略具体的细节。

除此之外，模块化思想还有其他一些优势，我们将在本章后面的内容中逐一介绍。

150 ~ 151

结构图（structure chart），或者叫模块图（module chart），描述的是一个程序的整体模块结构。在main函数中会调用一些函数，而这些被调用的函数本身同时会调用其他函数。图4-1是一些程序的结构图，这些程序会在本章和下一章的“解决应用问题”环节中设计和实现。需要注意的是，结构图中的操作流程并不等同于分解提纲中包含的执行步骤。结构图的主要意义在于展示将一个程序任务分解成不同的独立模块，以及各模块间的相互引用关

系。因此，不论是分解表格还是结构图，都是从不同角度提供有用的解题视角。同时还需要注意的是，结构图中并不包含引用标准C库的模块，因为这些模块被频繁地使用，同时它们也属于C语言环境整体的一部分（而无须被列出）。

图 4-1 结构图示例

当解决大规模的复杂问题时，程序代码量也会相应增大。因此，下面这三个步骤将帮助我们调试大型程序。首先，使用不同的编译器来运行程序是有帮助的，因为不同的编译器会给出不同的错误信息；实际上，对于同样的程序错误，有些编译器会给出非常全面的错误信息，而其他编译器提示的某些报错信息就非常简短。在调试大型程序时，另一个有效步骤就是将某些代码片段临时注释掉（/* 和 */），这样便可以集中精力去调试其他部分的代码。当然，注释时也要格外小心；有时注释掉了变量声明相关的语句，可能会导致正在调试的代码段无法正常使用该变量。最后，为了验证复杂函数的正确性，要对单个函数进行独立测试。这里使用一个特殊程序（称为驱动程序（driver）），它的目的是为测试人员和被测函数提供接口。通常来讲，驱动程序告诉测试人员输入要传递给函数的参数，由它调用被测试函数后打印出函数返回值。随着后续章节的介绍，驱动程序的重要性将愈加明显。

4.2 自定义函数

一个程序的执行总是从main函数开始。当程序运行至函数名时，就会开始调用（invoke）相关的函数。这些被调用的函数，或者在与main函数同一个文件中定义，或者在一个可用文件中单独定义，再或者定义在库文件里。（如果函数定义在一个系统库文件中，比如函数sqrt，那么该函数就称为库函数（library function）；除此之外的其他函数都称为

自定义函数（programmer-written function 或 programmer-defined function）。）在函数中的语句执行完毕后，主程序将继续执行函数调用处的后续语句。

4.2.1 函数示例

函数 sinc(x)，其分布如图 4-2 所示，在工程应用中被广泛使用。下面给出了 sinc(x) 函数最常见的一种定义：

$$f(x) = \mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

（函数 sinc(x) 偶尔也会被定义成 sin(x)/x。）该函数的值很容易计算，除了 sinc(0)，此时形成了不确定形式 0/0。根据微积分学的洛必达法则可以证明 sinc(0) = 1。

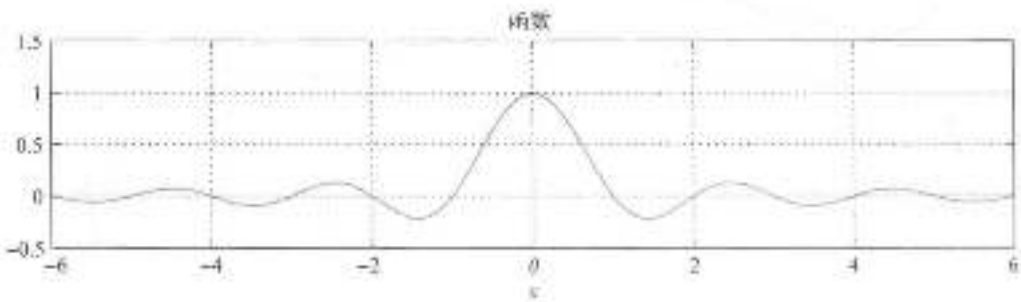


图 4-2 函数 sinc 在区间 [−20，20] 上的取值分布

假设现在要设计一个程序，允许用户输入一个区间上下限，a 和 b。随后程序应该计算出 x 在 a、b 之间均匀分布的 21 个函数值 sinc(x)。因此，第一个 x 取值应该是 a。接下来应该对 x 附加一个增量以获取下一个 x 值，并以此类推，直到获取 x 的第 21 个取值，也就是 b。因此，x 附加的增量应该等于

[152]

$$x\text{的常加增量} = \frac{\text{间隔宽度}}{20} = \frac{b-a}{20}$$

接下来选择 a 和 b 的取值，并且要事先确认，以 a 为第 1 个取值，以该增量为增量，确保第 21 个取值是 b。

由于函数 sinc(x) 并不是标准 C 函数库中提供的数学函数，因此我们使用两种方法实现问题解决方案。方案 1 是将函数 sinc(x) 计算的实现语句放在 main 函数中；方案 2 则是专门编写一个自定义函数来计算 sinc(x)，随后在主函数中需要进行相应计算的地方引用该函数。现在将这两种方法展示如下，读者可以自行比较。

方案 1

```
/*-------------------------------------------------------------*/
/*   程序 chapter4_1                                           */
/*                                                             */
/*   该程序打印函数 sinc 在区间 [a,b] 上的 21 个函数值。       */
/*   计算过程在主函数中实现                                    */

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

```
#define PI 3.141593

int main(void)
{
   /*  声明变量  */
   int k;
   double a, b, x_incr, new_x, sinc_x;

   /*  令用户输入区间端点  */
   printf("Enter endpoints a and b (a<b): \n");
   scanf("%lf %lf",&a,&b);
   x_incr = (b - a)/20;

   /*  计算并打印sinc(x)函数值表格  */
   printf("x and sinc(x) \n");
   for (k=0; k<=20; k++)
   {
      new_x = a + k*x_incr;
      if (fabs(new_x) < 0.0001)
         sinc_x = 1.0;
      else
         sinc_x = sin(PI*new_x)/(PI*new_x);
      printf("%f %f \n",new_x,sinc_x);
   }

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

153

方案 2

```
/*-----------------------------------------------------------------*/
/*  程序chapter4_2                                                 */
/*                                                                 */
/*  该程序打印sinc函数的21个值，使用自定义函数来实现计算过程        */

#include <stdio.h>
#include <math.h>
#define PI 3.141593

int main(void)
{
   /*  声明变量  */
   int k;
   double a, b, x_incr, new_x;
   double sinc(double x);

   /*  令用户输入区间端点  */
   printf("Enter endpoints a and b (a<b): \n");
   scanf("%lf %lf",&a,&b);
   x_incr = (b - a)/20;

   /*  计算并打印sinc(x)函数值表格  */
   printf("x and sinc(x) \n");
   for (k=0; k<=20; k++)
   {
      new_x = a + k*x_incr;
```

```
        printf("%f %f \n",new_x,sinc(new_x));
    }
    /*  退出程序  */
    return 0;
}
/^-----------------------------------------------------------------*/
/^  本程序估计 sinc 函数的值                                        ^/

double sinc(double x)
{
    if (fabs(x) < 0.0001)
        return 1.0;
    else
        return sin(PI^x)/(PI*x);
}
/*-----------------------------------------------------------------*/
```

154

下面是这两个程序的输出结果样例：

```
Enter endpoints a and b (a<b):
-2 2
x and sinc(x)
-2.000000 0.000000
-1.800000 -0.103943
-1.600000 -0.189207
-1.400000 -0.216236
-1.200000 -0.155915
-1.000000 0.000000
-0.800000 0.233872
-0.600000 0.504551
-0.400000 0.756827
-0.200000 0.935489
0.000000 1.000000
0.200000 0.935489
0.400000 0.756827
0.600000 0.504551
0.800000 0.233872
1.000000 0.000000
1.200000 -0.155915
1.400000 -0.216236
1.600000 -0.189207
1.800000 -0.103943
2.000000 0.000000
```

在图 4-3 中展示了对 4 个不同区间 $[a, b]$ 计算得出的 21 个值的分布情况。由于程序只计算了 21 个值，所以结果的分布依赖于区间的大小——区间越小，结果的分布就越平滑，效果也更好。在这里，方案 2 的 main 函数要比方案 1 中的 main 函数更易读，因为方案 2 中使用了函数使得程序更加简短。上面的示例中我们已经使用了一个自定义函数，下面将对函数语句进行更加一般化的讨论。

155

4.2.2 函数定义

函数包含一个定义语句，然后是声明和语句。函数定义语句的第一个部分是定义该函数得出的计算结果的类型（在上面的例子中是 double 类型）；如果函数不产生计算结果，那么类型为 void。在返回值类型之后，便是函数名和参数列表。因此函数的一般形式如下：

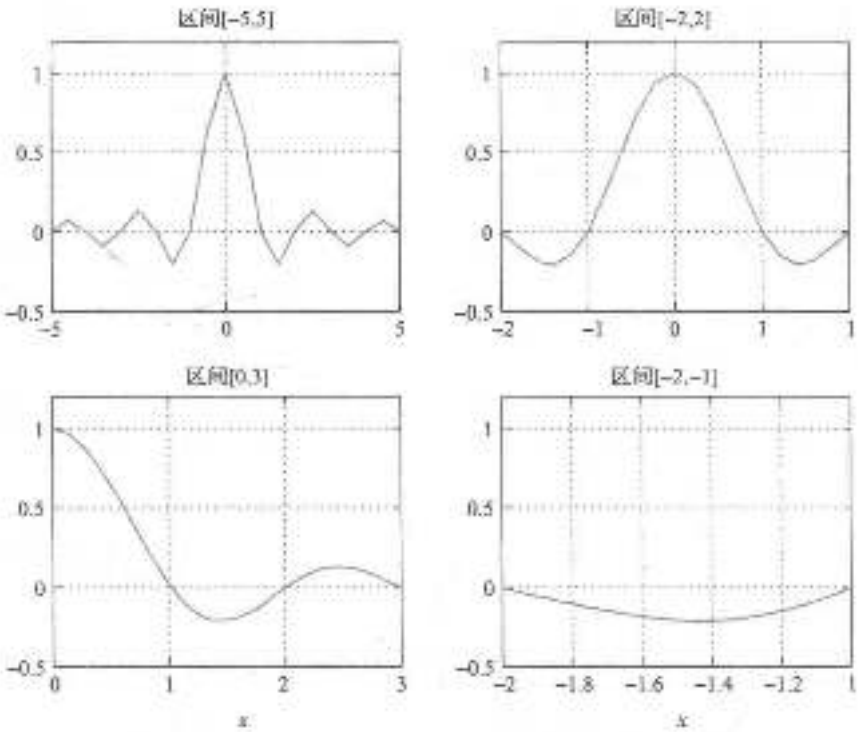


图 4-3 4 个不同区间的程序输出结果

```
返回值类型 函数名[一个或多个参数声明]
{
    声明;
    语句;
}
```

其中参数声明用于表示外界向函数传递的信息；如果函数没有输入参数（简称为参数），那么参数声明部分应该写作 void。函数中的附加变量是在声明中定义的。一个函数的声明部分和语句部分由一对大括号包裹，形成一个函数体。函数名应该有具体的含义，一般用于描述函数的功能和用途。在函数体中应该添加注释，以便清晰地描述函数功能并记录执行步骤。另外，还应该在自定义函数和 main 函数之间及其他自定义函数之间使用连字线构成的一个注释行进行区分。 156

所有函数都应包含一个 return 语句，它的一般形式为：

```
return 表达式;
```

其中，表达式就是返回给该函数的调用语句的值，而表达式类型则应该与函数定义的返回值类型相符以避免潜在错误。需要的时候，可以用类型转换操作符（在第 2 章讨论过）强制指定表达式类型。如果一个函数不具有返回值，那么其函数定义的一般形式如下：

```
void 函数名(参数声明)
```

在返回值是 void 的函数中的 return 语句不包含表达式，一般写成如下形式：

```
return;
```

函数定义可以在main函数之前，也可以在main函数之后。（main函数以右大括号结尾。）但是要注意，必须在一个函数的定义结束之后才能开始另一个函数；也就是说，函数定义是不能相互嵌套的。在上面的程序里可以看到，我们首先编写了main函数，然后再按照程序中的调用次序依次编写自定义函数。

下面将深入分析函数调用语句和函数自身之间的关系。

4.2.3 函数原型

在程序chapter4_2中main函数前的声明段中包含了如下语句：

```
double sinc(double x);
```

这条语句叫作函数原型（function prototype）语句。它的作用是通知编译器，main函数将要调用一个名叫sinc的函数，并且这个sinc函数需要一个double类型的参数，函数最后返回一个double类型的值。其中，标识符x并不是作为一个变量被定义；它只是用来说明sinc函数需要一个数来作为参数。实际上，在函数原型中只说明参数的类型也是合法的表示方式：

```
double sinc(double);
```

上面这两种原型语句传递给编译器的信息是相同的。我们推荐在原型语句中使用参数标识符，因为参数标识符可以更好地帮助我们记住参数定义和先后顺序。

一个函数原型可以是一个独立的语句，也可以通过预处理命令嵌入程序中，又或者，由于函数原型实际上是在定义其返回值的数据类型，所以一个函数原型也可以包含在其他变量声明里。例如，程序chapter4_2中的声明语句如下：

```
/*  声明变量和函数原型  */
int k;
double a, b, x_incr, new_x;
double sinc(double x);
```

157

上面的语句也可以写成如下形式：

```
/*  声明变量和函数原型  */
int k;
double a, b, x_incr, new_x, sinc(double x);
```

为了能够清晰地标识函数原型，一般在程序中将每个函数原型用一条单独的语句进行声明。

每一个程序中使用到的函数，都需要首先引入它的函数原型语句。stdio.h和math.h这类常用的头文件中已经包含了标准C库中许多函数的原型语句，在程序中直接嵌入就可以使用；否则，程序里使用的每一个系统函数（如printf和sqrt函数）都需要单独声明原型语句才能使用。如果一位程序员在他定义的函数调用了另一位程序员定义的函数，那么也需要把相应的函数原型语句添加到程序中。

如果一个程序调用了大量自定义函数，那么将所有函数原型声明语句逐个加到源代码中会变得异常繁杂。这种情况下，就需要使用自定义头文件来包含这些函数原型和相关的符号常量。头文件的文件名后缀是.h。头文件定义好后通过include语句嵌入源代码中，其中文件名用双引号包含。在第5章中我们设计了一组函数，用来实现为给定数据集计算常用的统计值。假设有一个包含了所有这些函数原型的头文件，将其命名为stat_lib.h，那么下面的预处理语句可以把这些原型全部包含在程序中：

```
#include "stat_lib.h"
```

程序员之间共享程序时，通常也会同时传递一个头文件，用以描述共享的函数原型。

4.2.4 参数列表

一个函数的定义语句包含了函数参数的定义：在这里参数叫作形式参数（formal parameter，简称形参）。而在程序里，任何调用函数的语句中必须为参数赋予对应的值；这些值叫作实际参数（actual parameter，简称实参）。例如，在本节早些时候设计的 sinc 函数，它的定义语句为：

```
double sinc(double x)
```

而主程序的 main 函数中调用该函数的语句为：

```
printf("%f %f \n",new_x,sinc(new_x));
```

其中，变量 x 是形参，而变量 new_x 是实参。当 printf 语句中的 sinc 函数调用执行时，实参中的值会被复制到形参中，即 x 被赋予一个新值，然后便开始执行函数 sinc 中的语句。函数执行后，其返回值会被打印出来。在这里有一个重要的地方需要注意，那就是当函数执行完毕时，形参中的值并不会退回给实参。为了说明这个过程，下面通过一个内存快照来展示实参与形参中的值是如何传递的。这里假设 new_x 的值为 5.0： 158

实参中的值复制到形参之后，sinc 函数也就开始执行了。在进行调试时，可以在函数调用前通过 printf 语句输出实参的内存快照，同时在函数开始执行时再输出形参的内存快照，能够更加清晰地看到程序的运行状态。

除了上面的基本形式，sinc 函数合法的调用方式还有很多，实参表中可以包括表达式或对其他函数的调用，下面给出了一些范例：

```
printf("%f \n",sinc(x+2.5));

scanf("%lf",&y);
printf("%f \n",sinc(y));

z = x*x + sinc(2*x);

m = sinc(fabs(y));
```

在上面的例子中，形参始终是 x，但实参分别是 x+2.5、y、2*x 和 fabs(y)，实参的选择因实际需求而异。

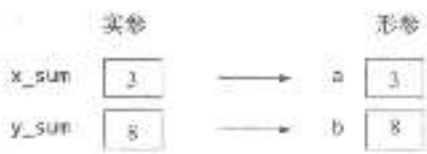
如果一个函数拥有多个参数，那么形参和实参就必须在数量、类型和顺序上严格保持一致。如果形参和实参的个数不匹配，编译器可以通过函数原型语句检测到错误。如果实参的数据类型与对应形参的类型不匹配，那么实参的值就会被转化为相应的数据类型；这种转换也称作参数类型的强制转换（coercion of argument），这个过程可能会引发错误，也可能不会。强制转换在第 2 章介绍过，它是将一个类型的变量值转移到另一个类型的变量中。将数据值转换到更高精度的类型时（例如从 float 转换成 double）通常不会出错；而将数据值转换到较低精度的类型时（例如从 float 转换成 int）都会有产生错误的风险。

我们通过下面的代码来说明参数类型强制转换的过程。这个函数能返回两个数之中的较大者：

```
/*-----------------------------------------------------------*/
/*  本程序返回两个整型值之中的较大者                          */

int max(int a,int b)
{
   if (a > b)
      return a;
   else
      return b;
}
/*-----------------------------------------------------------*/
```

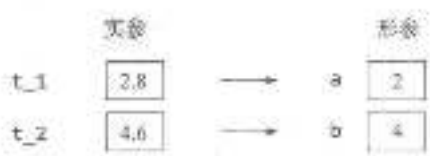
159

假设现有一个该函数的调用为 max(x_sum, y_sum)，并且 x_sum 和 y_sum 都是整型，值分别是 3 和 8。下面的内存快照展示了当函数 max(x_sum, y_sum) 被调用时，从实参到形参的值的传递过程：



最后函数语句将返回数值 8 作为函数调用 max(x_sum, y_sum) 的值。

现在假设一个 max 函数调用的参数为浮点型变量 t_1 和 t_2。其中 t_1 和 t_2 的值分别为 2.8 和 4.6。当函数调用 max(t_1, t_2) 执行时，便会发生如下参数传递：



当函数调用执行后，函数返回值为 4。显然，该函数返回了一个错误值。然而，出错的原因并不在函数本身；而在于函数调用时实参的数据类型错了。

除了上面所说的参数类型，如果实参的顺序不对也会引发错误。这种错误编译器往往检测不到，所以很难定位到哪里出错；因此在匹配形参和实参顺序的时候一定要多加小心。

上面 sinc 函数调用的例子通常叫作传值调用（call-by-value，或者 reference by value）。当进行函数调用时，实参中的值被传递给函数，作为相应的形参的值。一般来说，C 函数不会改变实参的值。但是也有例外，比如当实参为数组（将在第 5 章讨论）或指针（将在第 6 章讨论）时，情况会有所不同；这些例外情况的函数调用一般称为引用调用（call-by-reference）或地址引用（reference-by-address），在后面的章节中会依次介绍。

练习

思考下面的函数：

```
/*-----------------------------------------------------------*/
/*  该函数计算所有参数中正数的个数                            */

int positive(double a,double b,double c)
{
```

```
    int count;

    count = 0;
    if (a >= 0)
       count++;
    if (b >= 0)
       count++;
    if (c >= 0)
       count++;
    return count;
}
/*-------------------------------------------------------------*/
```

假设由下列语句进行函数调用：

```
x = 25;
total = positive(x,sqrt(x),x-30);
```

1. 画出实参和形参的内存快照。
2. 最后的 total 值是多少？

4.2.5 存储类型和作用域

在前面给出的示例程序里，变量的声明往往是在 main 函数和用户自定义的函数中进行的。当然，在 main 函数之前定义变量也是合法的。在编程时非常重要的一个环节就是要明确函数和变量的作用域（scope）。这里的作用域是指，在程序的哪些部分可以合法引用函数或变量；有时会根据函数或变量在程序的那部分是可见或可用的来定义其作用域。由于一个变量的定义域与它的存储类型（storage class）有密切的关联，接下来再分别介绍 4 种存储类型——自动（auto）、外部（extern）、静态（static）和寄存器（register）。 [160]

首先，对局部变量和全局变量的定义进行区分。局部变量（local variable）是在函数中进行定义，即函数的形参和其他在函数中进行声明的变量都是局部变量。除了定义局部变量的函数本身之外，外部是无法访问该局部变量的。此外，当函数开始执行时，其中的局部变量被赋值，而在函数执行结束后，变量的空间随即被释放，它的值也无法保留。全局变量（global variable）是在 main 函数或自定义函数之外被定义。由于是在所有函数的外部定义，所以程序中的任意函数均可访问到全局变量。在引用全局变量时，有些编译器要求函数在引用该全局变量之前，需要在类型指派前面加上一个关键字 extern 对其进行声明，以此来通知计算机在函数外寻找到该全局变量。自动存储类型（automatic storage class）表示的是局部变量；它是默认存储类型，当然也可以在类型指派前加上关键字 auto 来特别说明。外部存储类型（external storage class）表示的是全局变量；在函数中引用时必须要加上 extern 指派。而在初始定义全局变量时则没有硬性要求。 [161]

现在分析下面的程序：

```
#include <stdio.h>
int count=0;
...
int main(void)
{
   int x, y, z;
   ...
}
```

```
int calc(int a,int b)
{
   int x;
   extern int count;
   ...
}
void check(int sum)
{
   extern int count;
   ...
}
```

上面的程序里，变量 count 是一个全局变量，可以被函数 calc 和函数 check 引用。变量 x、y 和 z 是局部变量，只能在 main 函数中被引用；类似的，变量 a、b 和 x 也是只能在函数 calc 中被引用的局部变量；变量 sum 是只能在函数 check 中被引用的局部变量。注意到上面有两个局部变量 x，它们是完全不同的两个变量，分别具有各自不同的作用域。

在程序中声明的全局变量和用 extern 修饰的外部变量，在程序的整个执行期间都可以被访问到。因此可以通过使用适当的声明语句，在函数中很方便地使用全局变量进行数据传递，但是我们不提倡大量使用全局变量。一般来讲，参数是外界向函数传递信息的首选，这是因为所有的参数都可以在函数原型中体现出来，然而全局变量却不具备这个特质。要尽可能地避免使用全局变量。

函数的名称同样具有外部存储类型，因此也可以被其他函数所引用。在所有函数之外嵌入的函数原型声明同样是具有外部存储属性的，可以被程序中的任何函数所引用；这就解释了为什么我们只在代码的头部引入一次头文件 math.h，而不需要在每个函数中都引入它，就可以调用数学函数。但是，函数原型里的参数变量只在原型语句中可见。

静态（static）存储类型是用来说明一个局部变量所占用的内存空间会一直保留至整个程序执行完毕。因此，如果在一个局部变量的类型说明符前引入关键字 static，使其具有静态存储类型，那么即使该变量所在的函数执行完毕，变量的空间也不会被释放掉，变量的值也能够被一直保留。静态变量通常可以用来记录函数调用的次数，因为在函数被调用时静态变量能够保持着上一次函数调用时的值。

在变量类型指派之前引入关键字 register，是用来说明该变量被存储在寄存器（register）中，而不是在内存中。寄存器的存取速度远比内存的存取要快，所以一般在需要频繁获取数值的时候会用到寄存器存储类型。因为一个计算机可用的寄存器数量因系统而异，并且内存的存取速度不断被优化，所以这种存储类型很少会用到。

162

练习

使用 4.6 节的程序 chapter4_7，来给出下列信息（得出下列信息并不需要理解程序代码）：

1. 列出程序中的外部标识符。
2. 列出程序中的局部变量并判断其作用域。

使用 4.8 节的程序 chapter4_8，来给出下列信息（得出下列信息并不需要理解程序代码）：

3. 列出程序中的外部标识符。
4. 列出程序中的局部变量并判断其作用域。

4.3 解决应用问题：计算虹膜边界

在本节中，将会使用本章新介绍的编程方法去解决有关虹膜识别的问题。大多数虹膜识

别技术首先会采取一种分割操作。这种操作能够识别虹膜与瞳孔的边界，同时还能区分虹膜和巩膜（眼白部分）之间的边界，如图 4-4 所示。这对我们的肉眼来说很容易辨认，可是如果要使用计算机算法自动识别，这种分割操作是非常复杂且不易实现的。在一些研究分析中会采取手动分割的方法，也就是说首先在屏幕上提供一张人眼的图片，然后由实验者选取并点击瞳孔边界上的任意三个点，同时在瞳孔与眼白之间的边界上再选取三个点。计算机可以利用三点来确认并计算出在这三点上的圆方程。因此，便可以计算出由虹膜边界形成的圆方程。有了这些信息，就能提取出虹膜，并随即展开虹膜识别的下一步工作。

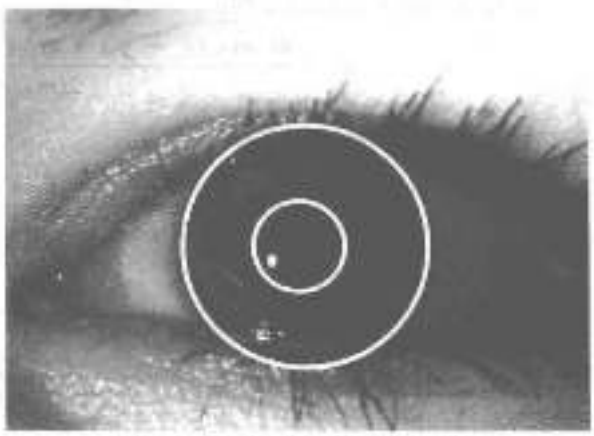


图 4-4 识别虹膜边界的人眼图片

本节要解决的问题是，在平面上取三个点，然后计算出通过这三个点的圆方程以及圆心的位置。根据给出的圆上三点来确定圆方程的问题，已经有一些现成的计算方法。本书中采用的方法是首先找出分别过点 P_1 和 P_2、点 P_2 和 P_3 的两条直线方程。假设这两条直线不平行，那么显然有：分别过两条线段（P_1P_2、P_2P_3）中点的中垂线一定相交于该圆的圆心（如图 4-5 所示）。由于相关的推导方程在网上都有详细的解析，故此处便不再给出详细的推导过程。下面给出的式子包括：直线 P_1P_2 和直线 P_2P_3 的方程，通过两条线段中点和圆心的中垂线方程，以及计算两条中垂线交点的方程。与此同时，这个交点就是初始的三个点所在圆的圆心。有了圆心坐标之后，便可以利用圆上的点来计算出圆半径。上面计算过程所需的方程如下所示，假设 P_1、P_2 和 P_3 的坐标分别为（x_1、y_1）、（x_2、y_2）和（x_3、y_3）。 163

直线 P_1P_2 的斜率：

$$m_{12} = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.1}$$

直线 P_2P_3 的斜率：

$$m_{23} = \frac{y_3 - y_2}{x_3 - x_2} \tag{4.2}$$

直线 P_1P_2 的方程：

$$y_{12} = m_{12}(x - x_1) + y_1 \tag{4.3}$$

直线 P_2P_3 的方程：

$$y_{23} = m_{23}(x - x_2) + y_2 \tag{4.4}$$

将线段 P_1P_2 等分的中垂线方程：

$$y_{p12} = -\frac{1}{m_{12}}\left(x - \frac{x_1 + x_2}{2}\right) + \left(\frac{y_1 + y_2}{2}\right) \tag{4.5}$$

将线段 P_2P_3 等分的中垂线方程：

164

$$y_{p23} = -\frac{1}{m_{23}}\left(x - \frac{x_2 + x_3}{2}\right) + \left(\frac{y_2 + y_3}{2}\right) \tag{4.6}$$

计算圆心的 x 坐标的方程：

$$x_c = \frac{m_{12}m_{23}(y_1 - y_3) + m_{23}(x_1 + x_2) - m_{12}(x_2 + x_3)}{2(m_{23} - m_{12})} \tag{4.7}$$

计算圆心的 y 坐标的方程：

$$y_c = -\frac{1}{m_{12}}\left(x_c - \frac{x_1 + x_2}{2}\right) + \left(\frac{y_1 + y_2}{2}\right) \tag{4.8}$$

计算圆半径的方程：

$$r = \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2} \tag{4.9}$$

有了详细的计算过程和所需的方程式，下面就可以用C语言来设计解决方案了。

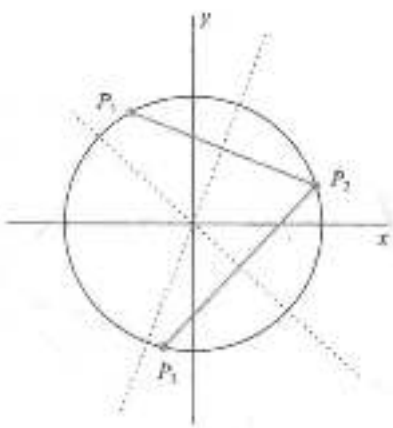


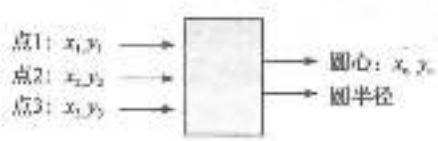
图4-5 一个圆上的点同圆心之间的几何关系

1. 问题陈述

在一个平面上给出三个点，确定这三点所在的圆的圆心坐标及半径。

2. 输入/输出描述

下图展示了，程序的输入是这三个点的 x、y 坐标，输出则是圆心的坐标以及圆的半径。



3. 手动演算示例

有了前面提供的方程式，现在就可以着手计算相应的圆心和半径了。现在假设已经给出了一个未知圆上的三个点：

$$P_1 = (-1, -1)$$
$$P_2 = (1, 1)$$

$$P_3 = (3, -1)$$

使用方程（4.1）和方程（4.2），可以计算出直线 P_1P_2 和 P_2P_3 的斜率 m_{12}、m_{23}：

$$m_{12} = 2/2 = 1$$
$$m_{23} = -2/2 = -1$$

165

使用方程（4.3）和方程（4.4），可以计算出直线 P_1P_2 和 P_2P_3 的方程式：

$$y_{12} = x$$
$$y_{23} = -x + 2$$

使用方程（4.5）和方程（4.6），可以计算出分别将线段 P_1P_2 和 P_2P_3 等分的垂直线方程：

$$y_{x12} = -x$$
$$y_{x23} = x - 2$$

使用方程（4.7）和方程（4.8），可以计算出圆心的坐标：

$$x_c = 1$$
$$y_c = -1$$

使用方程（4.9），可以计算出圆半径：

$$r = 2$$

至此，可以得出所求的圆方程为：

$$(x - 1)^2 + (y + 1)^2 = 4$$

（给出圆心坐标 (x_c, y_c) 和半径 r 的圆的方程式为：

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

其中，圆心坐标为 (x_c, y_c)，圆半径为 r。）

将这三个点的坐标代入该方程中进行验证，就可以确认这些点在圆上，得到的方程没有错误。

4. 算法设计

前面已经提出了一系列方程式用来解决计算圆心坐标的问题，这一系列计算可以设计成一个独立的函数。在前面的示例中，可以看到大部分方程式都是在计算为得到圆心的 x 坐标所需的中间值。现在将这些计算过程都放在一个函数里，这样主程序就会更加精炼也更易理解。下面，就开始设计主程序以及自定义函数的分解提纲。

程序的分解提纲

1）读取三个点的坐标值。

2）用函数来确定圆心的 x 坐标值。

3）计算圆心的 y 坐标值。

4）计算圆的半径。

5）打印圆心坐标及半径值。

166

自定义函数的分解提纲（假设函数的输入参数为三个给定点的坐标值，函数的输出为圆心的 x 坐标值）：

1）计算通过这三个给定点的两条直线的方程。

2）计算通过这三点的两条直线的中垂线方程。

3）计算圆心的 x 坐标值

由于主程序和函数的结构都比较简单，所以可以将上述分解提纲直接转化为C程序。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter4_3                                                */
/*                                                                 */
/*  该程序读取三个点的坐标值，通过调用函数来确定这三点所在的圆的圆心坐标 */
/*  以及圆的半径                                                   */

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  变量声明  */
   double x1, x2, x3, y1, y2, y3, m12, xc, yc, r;
   double circle_x_coord(double x1,double y1,double x2,double y2,
                         double x3,double y3);

   /*  用户从键盘输入数据  */
   printf("Enter x and y coordinates for first point: \n");
   scanf("%lf %lf",&x1,&y1);
   printf("Enter x and y coordinates for second point: \n");
   scanf("%lf %lf",&x2,&y2);
   printf("Enter x and y coordinates for third point: \n");
   scanf("%lf %lf",&x3,&y3);

   /*  通过函数来确定圆心的 x 坐标值  */
   xc = circle_x_coord(x1,y1,x2,y2,x3,y3);

   /*  计算圆心的 y 坐标值  */
   m12 = (y2 - y1)/(x2 - x1);
   yc = -(1/m12)*(xc - (x1 + x2)/2) + (y1 + y2)/2);

   /*  计算圆的半径  */
   r = sqrt((x1 - xc)*(x1 - xc) + (y1 - yc)*(y1 - yc));

   /*  打印圆的参数  */
   printf("\nCenter of Circle: (%.1f,%.1f) \n",xc,yc);
   printf("Radius of Circle: %.1f \n",r);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数根据一个圆上的三个点坐标，计算圆心的 x 坐标值                */

double circle_x_coord(double x1,double y1,double x2,double y2,
                      double x3,double y3)
{
   /*  变量声明  */
   double m12, m23, xc_num, xc_den, xc;
   /*  计算通过给出点的两条直线的斜率  */
   m12 = (y2 - y1)/(x2 - x1);
   m23 = (y3 - y2)/(x3 - x2);

   /*  计算圆心的 x 坐标值  */
   xc_num = m12*m23*(y1 - y3) + m23*(x1 + x2) - m12*(x2 + x3);
   xc_den = 2*(m23 - m12);
   xc = xc_num/xc_den;

   /*  返回圆心的 x 坐标值  */
   return xc;
```

167

```
}
/*-----------------------------------------------------------------*/
```

5. 测试

首先使用手动演算示例中的数据来测试程序。程序的运行结果如下：

```
Enter x and y coordinates for first point:
-1 -1
Enter x and y coordinates for second point:
1  1
Enter x and y coordinates for third point:
3 -1
Center of Circle: (1.0,-1.0)
Radius of Circle: 2.0
```

程序的执行结果与手动演算示例的相符，所以随后可以使用其他点来继续测试程序。 168

修改

本节主要讨论给出圆上的三个点，然后找出圆的圆心和半径的问题。而以下问题便是由此产生的：

1. 修改程序，使其输出圆的方程。
2. 如果有一条线是竖直的，其相应斜率为无穷大。确定这种情况是否发生，并且在退出程序之前输出错误信息。
3. 如果直线 P_1P_2 是竖直的，交换 P_1 和 P_3 的值，然后检查现在是否有两条不竖直的直线。是的话继续执行程序。
4. 如果直线 P_2P_3 是竖直的，交换 P_2 和 P_1 的值，然后检查现在是否有两条不竖直的直线。是的话继续执行程序。
5. 将上述问题 3 和问题 4 描述的情况结合起来，这个解决方案可以解决几乎所有的问题。除非三个点是一条竖直的直线，但是这表明这些输入点是错误的，因为这三个点不可能同时在一个圆上。

4.4 解决应用问题：冰山追踪

冰山是由卫星追踪的，它们的位置用经纬度来描述。确定冰山与它附近船舶间的距离是很重要的。在本节中，我们将编写程序来确定两个给定经纬度物体间的距离。在编写程序之前，我们需要简要讨论经纬度，并且找到用经纬度确定两点间距离的公式。

假设地球是一个半径为 3960 英里[⊖]的球体，然后我们就可以根据由测量的经纬度值确定的网格来描述地球表面的位置。为了理解这些测量值，我们先回顾一下球体表面上大圆（great circle）的定义——由球体和经过这个球体质心的平面交叉形成的圆。如果平面没有经过球心，将得不到球体表面的最大圆，因为形成的圆周长较小。本初子午线（prime meridian）是南北方向的最大圆，经过伦敦城外的格林威治和北极。赤道（equator）是东西方向的最大圆，从赤道到北极和南极的距离相等。我们定义一个以地球球心为原点的直角坐标系，z 轴从球心开始穿过北极，x

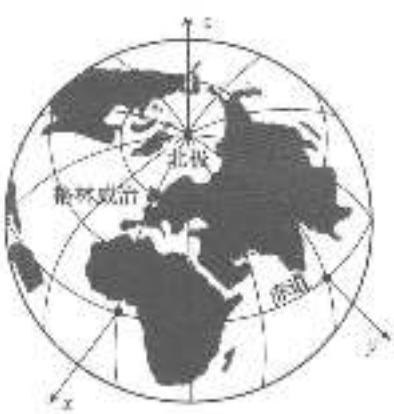


图 4-6 地球直角坐标系

⊖ 1 英里 = 1.609 344 千米

轴从球心开始穿过本初子午线和赤道的交点（如图 4-6 所示）。纬度（latitude）是从赤道开始向北或向南延伸的角度，（如 25°N 是指从赤道向北延伸 25°）。经度（longitude）是从本初子午线开始向西或向东延伸的角度（如 120°W 是指从本初子午线向西延伸 120°）。

全球定位系统（Global Positioning System，GPS）最初用于军事，在地球周围环绕 24 颗人造卫星来定位地球表面位置。这些人造卫星运行在距离地球 11000 英里的轨道上，每颗人造卫星都会广播无线电信号，信号中的编码指明了信息发送的时间和卫星在轨道上的位置。人造卫星上装载了精度极高的原子钟，每 70000 年的误差在 1 秒之内。GPS 接收器接收卫星信号，并且计算信号发送和接收之间的时间差以计算与卫星的距离。通过比较至少三颗卫星发送的信号，接收器可以确定它所在位置的纬度、经度和高度。

169

球面上两点间的最短距离是由这两点所确定的最大圆上的弧的长度。如果知道从球心出发分别到两点所形成的两个向量间的角度，我们就可以依据比例计算两点间的距离。举个例子，假设从球心出发的两个向量间的角度是 45°，那么这个角度的比值就是 45/360 或者说是整圆的 1/8。所以，两点间的距离是地球周长（$2\pi r$）的 1/8，或者 3110 英里。

计算两点间最短距离的最好方法是将经纬度进行一系列的坐标转换。回顾点 P 的球面坐标（ρ，ϕ，θ），在直角坐标系中，ρ（读作 rho）表示连接原点和点 P 的向量的距离，ϕ（读作 phi）表示 z 轴和向量间的角度，θ（读作 theta）表示 x 轴与向量在 xy 平面投影间的角度（如图 4-7 所示）。我们可以把球面坐标转换成标准的直角坐标，然后，用一个三角方程就可以计算直角坐标系中两点（或者向量）间的角度。一旦知道了两点间的角度，我们就可以用前面所描述的方法来计算两点间的球面距离。

为了解决上述问题，我们需要将经度和纬度转换成球面坐标的方程，将球面坐标转换为直角坐标的方程，和在直角坐标系中计算两个向量间角度的方程。我们使用图 4-7 中标出的符号作为变量来完成上面这些转换公式。

170

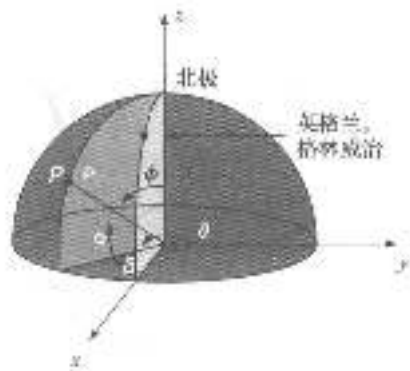


图 4-7 球坐标系

经度/纬度转换为球面坐标：

$$\alpha = 90° - \phi, \quad \beta = 360° - \theta$$

直角坐标转换为球面坐标：

$$x = \rho \sin\phi \cos\theta, \quad y = \rho \sin\phi \sin\theta, \quad z = \rho \cos\theta$$

两个向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 之间的角 γ：

$$\cos\gamma = \boldsymbol{a} \cdot \boldsymbol{b} / (|\boldsymbol{a}||\boldsymbol{b}|)$$

$\boldsymbol{a} \cdot \boldsymbol{b}$ 是向量 $\boldsymbol{a}$ 和向量 $\boldsymbol{b}$ 的点积，$|\boldsymbol{a}|$ 是向量 $\boldsymbol{a}$ 的长度。

直角坐标系中两个向量（x_a，y_a，z_a）和（x_b，y_b，z_b）的点积：

$$\boldsymbol{a} \cdot \boldsymbol{b} = x_a x_b + y_a y_b + z_a z_b$$

直角坐标系中向量（x_a，y_a，z_a）的长度：

$$|\boldsymbol{a}| = \sqrt{(x_a^2 + y_a^2 + z_a^2)}$$

最大圆上两点间的距离：

$$距离 = \frac{\gamma}{2\pi} \times 地球周长 = \frac{\gamma}{2\pi} \times \pi \times 2 \times 半径$$
$$= \gamma \times 3960$$

要特别注意这些公式中的角的单位。除非另有规定，否则假设这些角都是以弧度计量的（给角度乘以 $\pi/180$ 可以将其转换为弧度）。

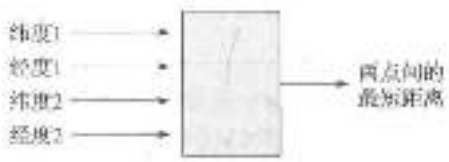
编写程序，要求用户输入北半球上两点的经度和纬度坐标，然后计算这两点间的最短距离。

1. 问题陈述

计算北半球上两点间的最短距离。

2. 输入 / 输出描述

对于这个程序，输入是北半球上两点的经度和纬度，输出是两点间的最短距离，如下图所示：



171

3. 手动演算示例

在手动演算示例中，我们将会计算纽约和伦敦之间的最大圆距离。纽约的纬度和经度分别是 45.75°N 和 74°W，伦敦的纬度和经分别是 51.5°N 和 0°W。

纽约的球面坐标是：

$$\phi = (90 - 40.75)^\circ = 49.25(\pi/180) = 0.8596$$
$$\theta = (360 - 74)^\circ = 286(\pi/180) = 4.9916$$
$$\rho = 3960$$

纽约的直角坐标（保留两位小数位）是：

$$x = \rho \sin\phi \cos\theta = 826.90$$
$$y = \rho \sin\phi \sin\theta = -2883.74$$
$$z = \rho \cos\theta = 2584.93$$

类似的，伦敦的直角坐标计算出来是：

$$x = 2465.16,\ y = 0,\ z = 3099.13$$

这两个向量之间角的余弦值等于两个向量的点积除以它们长度的乘积，即 0.6408。使用反余弦函数计算出来的 γ 值为 0.875。最后，纽约和伦敦之间的距离是：

$$0.875 \times 3960 = 3466 \text{ 英里}$$

4. 算法设计

为了开发算法，我们首先要将问题解决方案分解为一组顺序执行的步骤，如下所示：

分解提纲

1）读取两个位置的经度和纬度。

2）计算两个位置间的距离。

3）输出计算的距离。

完成手动演算示例后，就可以很容易地将分解步骤转换为伪代码。由于其他程序可能需要计算以经纬度表示的两点间的距离，我们将使用函数来开发计算过程。

[提炼后的伪代码]

主函数：获取两点的经纬度

　　使用函数 gc_distance 计算最大圆上两点间的距离，并且输出结果

gc_distance (lat1N, long1W, lat2N, long2W):

　　将经度和纬度转换为球面坐标

　　将球面坐标转换为直角坐标

　　计算两个向量的点乘

172

　　计算两个向量在最大圆上形成的弧的长度

伪代码中的步骤足够详细，可以将其转换为C程序。

```
/*-------------------------------------------------------------*/
/*  程序 chapter4_4                                            */
/*                                                             */
/*  这个程序计算北半球1度经纬度表示的两点间的距离              */
#include <stdio.h>
#include <math.h>
#define PI 3.141593

int main(void)
{
   /*  声明变量和函数原型  */
   double lat1, long1, lat2, long2;
   double gc_distance(double lat1,double long1,
                      double lat2,double long2);

   /*  得到两点的位置  */
   printf("Enter latitude north and longitude west ");
   printf("for location 1: \n");
   scanf("%lf %lf",&lat1,&long1);
   printf("Enter latitude north and longitude west ");
   printf("for location 2: \n");
   scanf("%lf %lf",&lat2,&long2);

   /*  输出最大圆距离  */
   printf("Great Circle Distance: %.0f miles \n",
           gc_distance(lat1,long1,lat2,long2));

   /*  退出程序  */
   return 0;
}
/*-------------------------------------------------------------*/
/*  这个函数使用最大圆距离计算两点间的距离                     */
double gc_distance(double lat1,double long1,
                   double lat2,double long2)
{
   /*  声明变量  */
   double rho, phi, theta, gamma, dot, dist1, dist2,
          x1, y1, z1, x2, y2, z2;

   /*  将经纬度转换为直角坐标  */
   rho = 3960;
```

```
    phi = (90 - lat1)*(PI/180.0);
    theta = (360 - long1)*(PI/180.0);
    x1 = rho*sin(phi)*cos(theta);
    y1 = rho*sin(phi)*sin(theta);
    z1 = rho*cos(phi);
    phi = (90 - lat2)*(PI/180.0);
    theta = (360 - long2)*(PI/180.0);
    x2 = rho*sin(phi)*cos(theta);
    y2 = rho*sin(phi)*sin(theta);
    z2 = rho*cos(phi);

    /*  计算两个向量的夹角  */
    dot = x1*x2 + y1*y2 + z1*z2;
    dist1 = sqrt(x1*x1 + y1*y1 + z1*z1);
    dist2 = sqrt(x2*x2 + y2*y2 + z2*z2);
    gamma = acos(dot/(dist1*dist2));

    /*  计算和返回最大圆距离  */
    return gamma*rho;
}
/*-----------------------------------------------------------------*/
```

173

5. 测试

用手动演算示例中的数据进行测试，得到下面的交互信息：

```
Enter latitude north and longitude west for location 1:
40.75 74
Enter latitude north and longitude west for location 2:
51.5 0
Great Circle Distance: 3466 miles
```

修改

上面给出了计算北半球上两点间最短距离的程序。试对上面的程序做适当修改，以回答下面的问题。

1. 编写一个函数，计算两个向量的夹角（以弧度表示），函数原型声明如下：

```
double angle(double x1, double y1, double z1,
             double x2, double y2, double z2)
```

2. 修改本节开发的程序，使其使用问题1中的函数。
3. 修改程序，使其允许用户输入北半球或者南半球上的纬度。在程序中，用户需要输入纬度值，然后通过输入N或者S来指定在北半球还是南半球。如果纬度是在南半球，则将纬度值前的符号换为负号，且仍然按照原来北半球的纬度计算规则即可。需要注意的是，虽然不需要改变gc_distance函数，但是需要对标识做修改，以确保在计算时使用的是北半球纬度，而在显示结果时能够正确显示南半球纬度。
4. 修改程序，使其允许用户输入东经度数或者西经度数。在程序中，用户需要输入经度值，然后通过输入W或者E来指定是东半球还是西半球。如果是东部符号，则用360减去这个经度值，然后按照西部经度进行计算。需要注意的是，虽然不需要改变gc_distance函数，但是需要对标识做出修改，以确保在计算时使用的是西半球经度，而在显示结果时能够正确显示东半球经度。 174
5. 修改程序，使其将问题3与问题4中的修改结合起来。也就是说，用户可以输入北半球或者南半球的纬度，西半球或者东半球的经度。同时，gc_distance函数还是不需要修改。

4.5 随机数

随机数（random number）的序列不是通过公式计算出来的，而是由某些分布的特征来

定义。这些特征包括最小值、最大值和平均值，还包括每个可能值出现的概率是否相同，或者是某些可能值的出现概率较高。随机数列可以通过实验产生，比如抛硬币、掷骰子或者选择编号球。此外随机数列也可以由计算机生成。

许多工程问题的解决过程中需要用到随机数列。在有些应用中，随机数用于实现对复杂问题的仿真。大量运行仿真实验以分析结果数据，其中每运行一次，就重新产生一组随机数。我们也可以用随机数来近似噪声序列。例如，我们在收音机里听到的静电干扰就是噪声序列。如果测试程序需要用到一个描述收音机信号的输入数据文件，那么我们可以生成噪声，并将其插入语音信号或者音乐信号中，来提供更加逼真的信号。

工程上经常要用到分布在指定范围内的随机数。例如，我们需要生成1 ~ 500之间的随机整数，或者-5 ~ 5之间的随机浮点数。现在讨论在两个指定值之间生成随机数。随机数中每个数据出现的概率是均等的，即如果随机数是1 ~ 5之间的整数，那么在集合{1, 2, 3, 4, 5}中的每个数是按照等概率出现的，也就是说每一个数字出现的概率都是20%。在指定集合内等概率出现的随机数称为统一随机数（uniform random number），或者均匀分布随机数。

4.5.1 整数序列

标准C库中有一个rand函数，可以生成0 ~ RAND_MAX之间的随机整数，RAND_MAX是定义在stdlib.h中依赖于系统的整数（RAND_MAX一般是32 767）。rand函数没有输入参数，调用时使用rand()即可引用。因此，要生成和输出两个随机数，可以使用下面的语句：

```
printf("random numbers: %i %i \n",rand(),rand());
```

运行程序后会发现，每次执行这条语句输出的两个值总是相同的，因为rand函数是按照某个指定序列生成整数的（因为这个序列最终会开始周期性的重复，所以它有时候被称为伪随机（pseudo-random）序列，而不是真正的随机序列）。然而，如果我们在同一个程序中继续生成其他随机数，则会得到不同的随机数序列。因此，下面的一组语句生成4个随机数：

175

```
printf("random numbers: %i %i \n",rand(),rand());
printf("random numbers: %i %i \n",rand(),rand());
```

在一次程序的运行中，每次rand函数被调用，就会生成一个新值。但是，程序每次产生的数字序列是相同的。

为了使程序每次执行后都生成新的随机数列，我们需要给随机数生成器一个新的随机数种子（random-number seed）。srand函数（函数原型定义在stdlib.h中）可以为随机数发生器指定种子，对于不同的种子值，由rand可以产生不同的随机序列。srand函数的参数是无符号整型，用于初始化随机数序列的计算，但是种子的值并不是序列的第一个数。如果在rand函数引用之前没有使用srand函数，计算机默认种子值是1。因此，如果指定种子值为1，rand函数产生的序列和没有指定种子时是一样的。

在下面的程序中，用户可以输入一个种子值，然后程序生成10个随机数。如果用户每次执行程序都输入相同的种子值，则每次生成的随机整数都相同。如果每次输入不同的种子值，则每次生成的随机整数不同。rand和srand的函数原型都包含在stdlib.h中。下面是程序代码：

```
/*-----------------------------------------------------------*/
/*  程序 chapter4_5                                          */
/*                                                           */
/*  本程序生成和输出 1 到 RAND_MAX 之间的 10 个随机整数       */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   /*  声明变量  */
   unsigned int seed;
   int k;

   /*  读取用户输入的种子值  */
   printf("Enter a positive integer seed value: \n");
   scanf("%u",&seed);
   srand(seed);

   /*  生成并输出 10 个随机数  */
   printf("Random Numbers: \n");
   for (k=1; k<=10; k++)
      printf("%i ",rand());
   printf("\n");

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------*/
```

176

使用 Microsoft Visual C++ 2010 编辑器的一个示例输出如下所示：

```
Enter a positive integer seed value:
123
Random Numbers:
440 19053 23075 13104 32363 3265 30749 32678 9760 28064
```

在自己的计算机上测试这个程序，可以看到使用相同的种子生成相同的数列，使用不同的种子生成不同的数列。

因为 stdlib.h 中包含 rand 和 srand 原型，所以不需要在程序中单独声明。但还是需要单独分析一下这些函数的原型。rand 函数返回一个整数，没有输入参数，它的原型声明如下：

```
int rand(void);
```

srand 函数没有返回值，并且有一个无符号整数作为输入参数，所以函数原型如下：

```
void srand(unsigned int);
```

用 rand 函数很容易生成指定范围内的随机数。例如，我们用下面的语句可以生成 0 ~ 7 之间的随机数。在这行语句中，首先生成 0 ~ RAND_MAX 之间的随机数，然后使用模运算符计算随机数与 8 的模：

```
x = rand()%8;
```

模运算的结果是 rand() 除以 8 的余数，所以 x 的值可以假设为是 0 ~ 7 之间的整数。

假设我们想要生成 -25 ~ 25 之间的随机数，可能出现的整数有 51 个，可以用下面的语句生成这个范围内的随机数：

```
y = rand()%51 - 25;
```

这条语句首先生成0～50之间的一个随机数，然后用这个随机数减去25，就得到了-25～25之间的值。

按照这个思路我们可以编写函数来生成两个指定数字 a 和 b 之间的随机数。首先计算包含在 a 和 b 之内的整数个数 n，可以简单算出 n 就是 $b-a+1$。然后使用模运算符和rand()生成0到 $n-1$ 之间的一个整数。最后，加上下限 a，使这个数字在 a 与 b 之间。这3步可以合并在函数的return语句中的一个表达式中。

```
/*-----------------------------------------------------------------*/
/*  该函数生成指定界限a与b之间的一个随机整数(a<b)                  */
int rand_int(int a,int b)
{
   return rand()%(b-a+1) + a;
}
/*-----------------------------------------------------------------*/
```

177

为了说明这个函数的用法，下面这段代码生成并打印了用户指定界限间的10个随机整数（随机序列的生成从用户输入种子开始）：

```
/*-----------------------------------------------------------------*/
/*  程序chapter4_6                                                 */
/*                                                                 */
/*  该程序生成并输出用户指定界限间的10个随机整数                    */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   /*  声明变量和函数原型  */
   unsigned int seed;
   int a, b, k;
   int rand_int(int a,int b);

   /*  读取种子值和区间界限  */
   printf("Enter a positive integer seed value: \n");
   scanf("%u",&seed);
   srand(seed);
   printf("Enter integer limits a and b (a<b): \n");
   scanf("%i %i",&a,&b);

   /*  生成并输出10个随机数  */
   printf("Random Numbers: \n");
   for (k=1; k<=10; k++)
      printf("%i ",rand_int(a,b));
   printf("\n");

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数生成指定界限a与b之间的一个随机整数(a<b)                  */
int rand_int(int a,int b)
{
   return rand()%(b-a+1) + a;
}
/*-----------------------------------------------------------------*/
```

该程序的一组输出示例如下所示：

```
Enter a positive integer seed value:
13
Enter integer limits a and b (a<b):
-5 5
Random Numbers:
-1 3 1 4 -2 -3 5 0 -2 4
```

178

需要说明的是，生成的随机数列的值是系统相关的，不同的编译器会得到不同的值。

修改

修改上述程序，使用不同的种子值，对于下列每组范围都生成一组随机整数。

1. 0 ~ 500　　2. -10 ~ 200　　3. -50 ~ -10　　4. -5 ~ 5

4.5.2 浮点数序列

在许多工程问题中，需要生成指定区间 $[a, b]$ 内的随机浮点数。将 0 ~ RAND_MAX 之间的整数转换为 $[a, b]$ 内的浮点数有 3 个步骤。将 rand 函数的返回值除以 RAND_MAX，得到一个 0 ~ 1 之间的随机数；然后将这个 0 ~ 1 之间的数乘上（$b-a$）（区间 $[0, b-a]$ 的宽度）就得到 0 ~ $b-a$ 之间的数，最后将这个 0 ~ $b-a$ 之间的数字加上 a，以使其在 a 与 b 之间。在下面的函数中，这 3 个步骤被合并到一个 return 语句中：

```
/*                                                          */
/*  该函数生成 a 和 b 之间的随机双精度数                      */

double rand_float(double a,double b)
{
   return ((double)rand()/RAND_MAX)*(b-a) + a;
}
/*----------------------------------------------------------*/
```

注意强制转换符是很必要的，它把 rand() 生成的整数转换为 double 型，以使除法运算的结果也是 double 型。

可以对本节前面的程序进行修改，使其生成和输出浮点型随机数。修改后的一组示例值如下所示：

```
Enter a positive integer seed value:
82
Enter limits a and b (a<b):
-5 5
Random Numbers:
-4.906613 -3.671834 -1.478164 -2.086093 -4.181494 -3.135624
-4.559923 1.599628 1.382031 0.490280
```

179

修改

修改生成随机整数的程序，使得程序生成用户指定范围内的 10 个随机浮点数，然后使用不同的种子值，在下列每组范围内都生成若干随机数：

1. 0.0 ~ 1.0　　2. -0.1 ~ 1.0　　3. -5.0 ~ -4.5　　4. 5.1 ~ 5.1

4.6 解决应用问题：仪器可靠性

可靠性（reliability）是指系统中的元件正常工作的时间占总使用时间的比例，通常通过对概率与统计数据的研究得到用于分析仪器可靠性的公式。因此，如果一个元件的可靠性为0.8，则该元件80%的使用时间可以正常工作。多个元件可以组合形成元件系统，而求解系统可靠性的前提是系统中单个元件的可靠性已知。考虑图4-8中的关系图，在串联设计中，为了使信息从 a 点流到 b 点，三个元件都必须正常工作，而在并行设计中，仅需三个元件中的一个正常工作，信息就能从 a 点流到 b 点。如果我们知道了单个元件的可靠性，那么特定组合的可靠性就可以由下面两种方法确定：分析计算方法，即依据概率与统计的定理和结论分析计算得到解析可靠性；计算机仿真，即通过计算机程序模拟仿真给出可靠性的估计值。

180

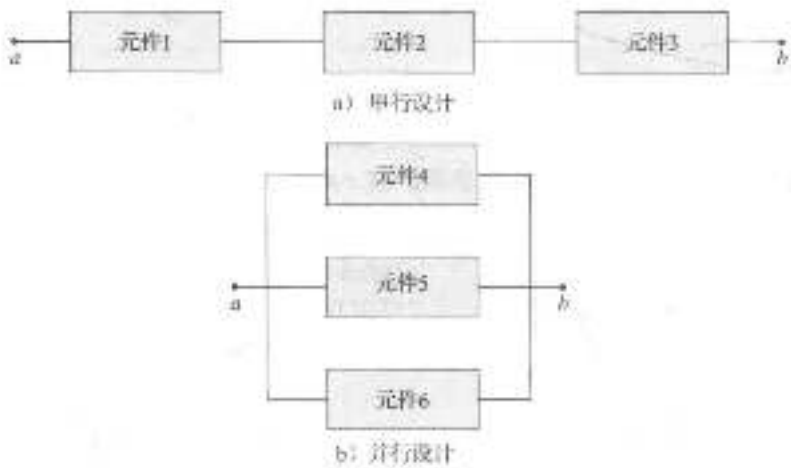


图4-8 串行和并行结构

考虑图4-8a中的串行结构，如果 r 是一个元件的可靠性，并且三个元件的可靠性相同，那么串行结构的可靠性可以表示为 r^3。因此，如果每个元件的可靠性都是0.8（一个元件80%的时间正常工作），串行结构的解析可靠性就是 0.8^3，即0.512。因此，串行结构51.2%的时间正常工作。

考虑图4-8b中的并行结构，如果 r 是一个元件的可靠性，并且三个元件的可靠性相同。那么并行结构的可靠性可以表示为 $3r-3r^2+r^3$。因此，如果每个元件的可靠性都是0.8，并行结构的解析可靠性就是 $3\times0.8-3\times0.8^2+0.8^3=0.992$，即并行结构99.2%的时间正常工作。直觉上，并行结构更可靠，因为只要三个元件中的一个正常工作，整个结构就能正常工作。反之，只有三个元件都正常工作，串行结构才能正常工作。

我们也可以使用计算机仿真（computer simulation）产生的随机数据估算两种设计方式的可靠性。首先，我们需要仿真单个元件的性能。如果一个元件的可靠性是0.8，那么它80%的时间正常工作。为了仿真这项工作，我们需要生成0～1之间的随机数。如果随机数在0～0.8之间，我们假设元件正常工作，否则，没有正常工作（我们也可以使用0～0.2表示非正常工作，0.2～1.0表示正常工作）。为了仿真具有三个元件的串行结构，我们将随机生成3个0～1之间的浮点数。如果这三个随机数都小于等于0.8，则认为本次仿真运行工作正常；如果三个数中有一个大于0.8，则本次仿真运行无法正常工作。如果重复进行成

百上千次仿真实验，就可以计算出整个系统正常工作次数所占的比例。这种仿真估计是对于使用解析法计算可靠性的一种近似。

为了估计单个元件可靠性是0.8的并行结构的可靠性，我们再一次生成3个0～1之间的随机浮点数。如果3个随机数中任何一个小于等于0.8，则本次仿真运行工作正常。如果3个数都大于0.8，则本次仿真运行无法正常工作。为了通过仿真估计可靠性，可以用正常工作的实验次数除以总的实验次数。

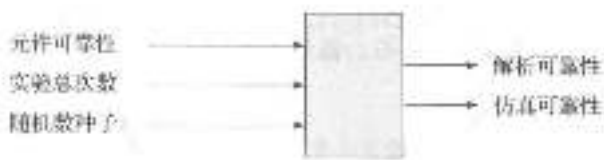
随着实验次数的增加，通过计算机仿真得到的可靠性应该近似于解析计算得到的可靠性。因此，如前面讨论的，我们可以用计算机仿真来验证解析计算值的正确性。在某些情况下，通过解析计算得出仪器的可靠性是非常困难的，而在这些情况下，可以使用计算机仿真来得出可靠性的估计值。

编写程序，利用计算机仿真的方法估算串行结构与并行结构系统的可靠性，并比较解析计算与计算机仿真的结果。允许用户输入单个元件的可靠性和仿真实验的次数。 181

1. 问题陈述

比较有三个元件的串行结构和并行结构的解析可靠性与仿真可靠性，假设三个元件的可靠性相同。

2. 输入/输出描述



从I/O图中可以看到，输入信息是元件的可靠性、实验的次数和随机数序列的随机数种子。输出信息是串行结构和并行结构的解析可靠性和仿真可靠性。

3. 手动演算示例

手动演算示例中，我们假设元件的可靠性为0.8，并进行三次仿真实验。每次实验需要使用三个随机数，假设下面是实验用的前9个随机数（由rand_float函数生成在0－1间的随机数，种子值是47）：

第一组中的三个随机数：

0.005860 0.652303 0.271187

第二组中的三个随机数：

0.589007 0.064699 0.992248

第三组中的三个随机数：

0.719565 0.786615 0.001923

通过每组中的三个随机数，我们可以确定串行结构和并行结构是否正常工作。对于第一组中的三个随机数，三个数都小于0.8，所以串行结构和并行结构都正常工作。第二组中有一个数大于0.8，所以只有并行结构正常工作。第三组随机数表明两个结构都正常工作。所以三次实验的解析结果（本节前面计算过）和仿真结果如下所示：

```
Analytical Reliability:
Series: 0.512  Parallel: 0.992
Simulation for 3 Trials
Series: 0.667  Parallel: 1.000
```

[182]

当增加实验次数，仿真结果应该更接近解析结果。如果改变随机数种子，即使只有三次实验，仿真结果也会改变。

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解为几个连续的解决步骤：

分解提纲

1）读取元件可靠性、实验次数和随机数种子。

2）计算解析可靠性。

3）计算仿真可靠性。

4）打印出两种可靠性的对比结果。

步骤1）提示用户输入程序必需的信息，然后读取相应数据。步骤2）使用前面给出的公式来计算解析可靠性。因为计算过程相对简单，所以直接在main函数中实现。步骤3）包含一个循环体来生成随机数，并且确定在每次试验中的元件配置是否工作正常。其中函数rand_float负责在循环体中生成随机数。最后在步骤4）中打印计算结果。图4-1中已经给出了解决方案的结构图。提炼后的伪代码如下所示：

```
[提炼后的伪代码]
主函数：读取元件可靠性数组、实验的次数，
        以及随机数种子
        计算解析可靠性
        设置 series_success 为 0
        设置 parallel_success 为 0
        设置 k 为 1
        while k <= 实验次数
          生成三个 0 - 1 之间的随机数
          if 每个随机数之元件可靠性
            series_success 自增 1
          if 存在任何一个随机数之元件可靠性
            parallel_success 自增 1
          k 自增 1
        打印解析可靠性
        打印仿真可靠性
```

伪代码中的操作步骤已经足够详细，可以直接转化为C程序，在程序中还使用了前面已经实现过的函数rand_float：

```
/*-----------------------------------------------------------------*/
/*   程序 chapter4_7                                               */
/*                                                                 */
```

183

```
/*  该程序利用计算机仿真估测在串行结构和并行结构下的可靠性值                    */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
   /*  声明变量和函数原型  */
   unsigned int seed;
   int n, k;
   double component_reliability, a_series, a_parallel,
          series_success=0, parallel_success=0,
          num1, num2, num3;
   double rand_float(double a,double b);

   /*  获取计算机仿真所需的数据  */
   printf("Enter individual component reliability: \n");
   scanf("%lf",&component_reliability);
   printf("Enter number of trials: \n");
   scanf("%i",&n);
   printf("Enter unsigned integer seed: \n");
   scanf("%u",&seed);
   srand(seed);
   printf("\n");

   /*  计算解析可靠性  */
   a_series = pow(component_reliability,3);
   a_parallel = 3*component_reliability
                - 3*pow(component_reliability,2)
                + pow(component_reliability,3);

   /*  确定仿真可靠性估计值  */
   for (k=1; k<=n; k++)
   {
      num1 = rand_float(0,1);
      num2 = rand_float(0,1);
      num3 = rand_float(0,1);
      if (((num1<=component_reliability) &&
          (num2<=component_reliability)) &&
          (num3<=component_reliability))
             series_success++;
      if (((num1<=component_reliability) ||
          (num2<=component_reliability)) ||
          (num3<=component_reliability))
             parallel_success++;
   }
   /*  打印结果  */
   printf("Analytical Reliability \n");
   printf("Series: %.3f Parallel: %.3f \n",
           a_series,a_parallel);
   printf("Simulation Reliability, %i trials \n",n);
   printf("Series: %.3f Parallel: %.3f \n",
           (double)series_success/n,
           (double)parallel_success/n);
```

184

```
   /*  退出程序  */
   return 0;
}
/*-------------------------------------------------------------------*/
/*  该函数生成一个在 a, b [a<b] 之间的 double 型随机数                */
double rand_float(double a,double b)
{
   return ((double)rand()/RAND_MAX)*(b-a) + a;
```

```
}
/*-----------------------------------------------------------------------*/
```

5. 测试

将手动演算示例中的数据作为程序输入，可以得到如下输出结果，同前面手算的结果完全符合：

```
Enter individual component reliability:
0.8
Enter number of trials:
3
Enter unsigned integer seed:
47

Analytical Reliability
Series: 0.512 Parallel: 0.992
Simulation Reliability, 3 trials
Series: 0.667 Parallel: 1.000
```

下面是额外增加的两次仿真结果，说明了当实验次数增加时，仿真结果会逐渐趋近于解析结果：

```
Enter individual component reliability:
0.8
Enter number of trials:
100
Enter unsigned integer seed:
123

Analytical Reliability
Series: 0.512 Parallel: 0.992
Simulation Reliability, 100 trials
Series: 0.470 Parallel: 1.000

Enter individual component reliability:
0.8
Enter number of trials:
1000
Enter unsigned integer seed:
3535

Analytical Reliability
Series: 0.512 Parallel: 0.992
Simulation Reliability, 1000 trials
Series: 0.530 Parallel: 0.990
```

185

修改

本节的主要内容是比较解析可靠性和仿真可靠性，而以下这些问题便是由此产生的：

1. 使用该程序来分别计算试验次数为10、100、1000和10 000的仿真结果，假设元件可靠性为0.85。
2. 使用该程序来计算1000次试验的仿真结果，分别采用5个不同的随机数种子。假设元件可靠性为0.75。
3. 要使一个串行结构的可靠性为0.7，需要可靠性为多少的元件？（提示：可以使用解析可靠性方程。）使用程序来验证你的答案。
4. 要使一个并行结构的可靠性为0.9，需要可靠性为多少的元件？在这里使用解析可靠性方程并不会将问题简化。如果无法算出多项式方程的根，可以用程序来验证结果，直到找到一个仿真可靠性与0.9足够接近的值。

*4.7 数值方法：求多项式的根

多项式就是关于一个变量的函数，它可以表示为下面的一般形式：

$$f(x) = a_0x^N + a_1x^{N-1} + a_2x^{N-2} + \cdots + a_{N-2}x^2 + a_{N-1}x + a_N \qquad (4.10)$$

其中，变量为 x，变量的系数则表示为 a_0，a_1，…，a_N。而多项式的次数等于最大非零指数。因此，对于一个三次多项式（次数为 3）的一般形式为

$$g(x) = a_0x^3 + a_1x^2 + a_2x + a_3$$

而下面就是一个典型的三次多项式

$$h(x) = x^3 - 2x^2 + 0.5x - 6.5$$

对式（4.10）进行分析可以发现，对于多项式一般形式方程中的每一项，系数下标与变量指数之和正好等于该多项式的次数。

4.7.1 多项式的根

在求解很多工程问题时，需要寻找一个如下形式的方程的根：

$$y = f(x)$$

所谓方程的根（root）实际上就是当 y 等于 0 时，对应的 x 值。那些需要人们求解方程的根的工程应用有很多，比如为一个机械臂设计控制系统，设计车用弹簧和减震器，分析发动机的机械反馈以及研究滤波器的稳定性等。

如果函数 $f(x)$ 是一个 N 次多项式，则 $f(x)$ 应该正好具有 N 个根。这 N 个根可能包含实根也可能包含复根，下面的例子会展示这些不同的情况。假设多项式的系数（a_0，a_1，…，a_N）均为实数，那么该多项式的根常常会出现共轭复数（前面介绍过，一个复数可以表示为 $a+ib$，其中 $i=\sqrt{-1}$，而 $a+ib$ 的共轭为 $a-ib$。） 186

如果一个多项式被因式分解成若干个一次项的乘积，那么只需解出每一个一次项为 0 时的根，便可轻松求得多项式的根。例如，下面的方程：

$$\begin{aligned} f(x) &= x^2 + x - 6 \\ &= (x-2)(x+3) \end{aligned}$$

如果令 $f(x)$ 等于 0，则有：

$$(x-2)(x+3) = 0$$

当 $f(x)=0$ 时，该方程的根（也就是 x 的值）为 $x=2$ 和 $x=-3$。这些根还对应着多项式与 x 轴的交点，如图 4-9 所示。

如果一个二次方程（多项式的次数是 2）无法被因式分解，还可以用二次公式来确定方程的两根。一个二次方程的一般式可以表示为

$$y = ax^2 + bx + c$$

则该方程的根可以通过如下公式计算得出：

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

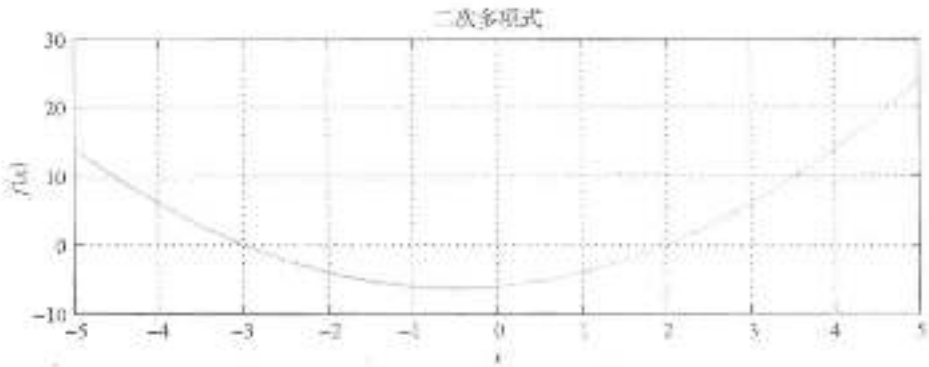


图 4-9 具有两个实根的多项式

于是，对于二次方程

$$f(x) = x^2 + 3x + 3$$

它的两个根分别为

[187]

$$x_1 = \frac{-3 + \sqrt{-3}}{2} = -1.5 + 0.87\sqrt{-1}$$

和

$$x_2 = \frac{-3 - \sqrt{3}}{2} = -1.5 - 0.87\sqrt{-1}$$

因为一个三次多项式的次数是 3，也就是拥有三个根。如果假设多项式的系数都是实数，那么对于多项式的根有以下 4 种可能：

- 三个不同的实根（不等根）。
- 三个相同的实根（三重根）。
- 两个相同的实根（二重根）和一个不同的实根。
- 一个实根和一对共轭复数根。

下列函数是对这 4 种情况分别进行举例说明：

$$\begin{aligned}
f_1(x) &= (x-3)(x+1)(x-1) \\
&= x^3 - 3x^2 - x + 3, \\
f_2(x) &= (x-2)^3 \\
&= x^3 - 6x^2 + 12x - 8, \\
f_3(x) &= (x+4)(x-2)^2 \\
&= x^3 - 12x + 16, \\
f_4(x) &= (x+2)[x-(2+i)][x-(2-i)] \\
&= x^3 - 2x^2 - 3x + 10
\end{aligned}$$

这些函数的图像分布如图 4-10 所示。再次看到方程的实根对应着函数与 x 轴的交点。

要确定一次或二次多项式的根相对简单，但是对于三次以上的高阶多项式来说，求解多项式的根就变得非常困难。当然，目前已经有很多数值技术专门用来求解高阶多项式的根。比如增量搜索、二分法以及错误定位法。最后一种方法是通过搜索函数符号变化区间来确定实根，因为发生符号变化就表明了函数正在穿过 x 轴。除此之外，还有其他的方法诸如牛

顿－拉夫逊方法等，可以用来求解多项式的复数根。

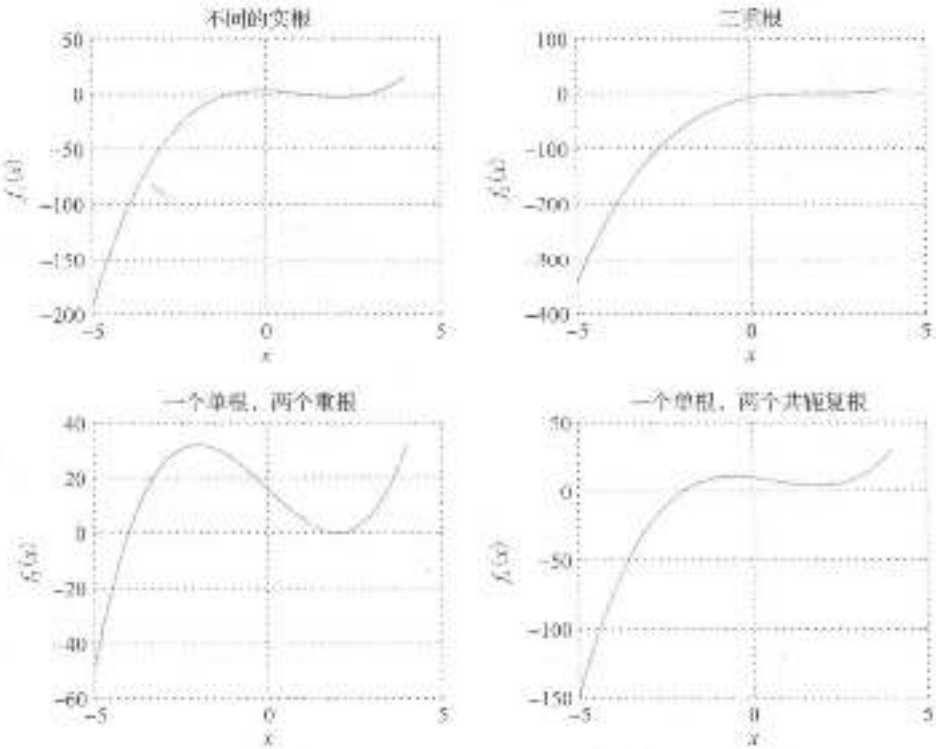


图 4-10 三次多项式

4.7.2 增量搜索技术

增量搜索（incremental-search）技术通常被用来确定一个方程在区间 $[a, b]$ 内的实根。这个方法实际上就是要寻找一个子区间 $[a_k, b_k]$，使得方程在子区间的一端为正，在另一端为负。此时可以确定在该区间内至少有一个根。

关于增量搜索技术有很多变体。这里讨论的方法是，首先选择一个步长，并以此将原始区间划分成一组较小的子区间，如图 4-11 所示。对于每个子区间，要判定函数在两个区间端点处的取值情况。

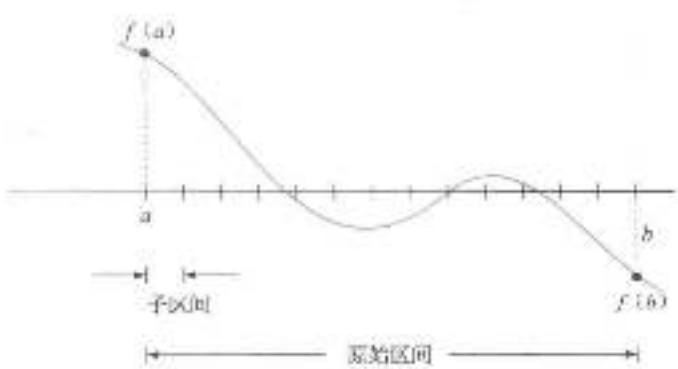


图 4-11 增量搜索

如果两个函数值的乘积为负，则说明在此区间上存在一个根。（乘积为负表明一个函数值为正，另一个函数值为负；因此，该函数在这个子区间内必定穿过 x 轴。）此时，可以估计根就是这段小区间的中点，如图 4-12a 所示。当然，也有可能子区间的端点就是根，或者距离根非常接近，如图 4-12b 所示。要注意的是，一个浮点值不可能完全等于 0，所以在检验区间端点是否为实根的时候应该将函数值与一个非常小的数相比较，而不是同 0 比较。

188 ~ 189

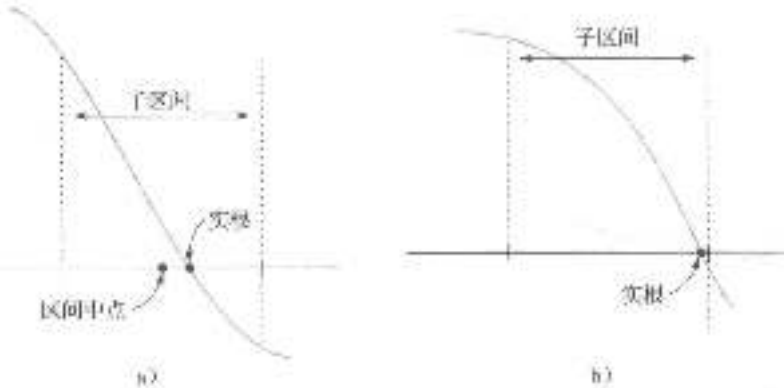


图 4-12 子区间分析

当然增量搜索技术也不是万能的，还有很多情况下使用增量搜索无法求解多项式的根。例如，在一个子区间内存在两个根的情况，此时由于区间端点处的两个函数值符号必然相同，它们的乘积也一定是正数，故算法会选择跳过并开始检验下一个子区间。另一个例子是，假设一个子区间内包含三个根。此时由于区间端点处的函数值符号不同，算法会估计根值为子区间的中点。接下来开始继续检验其中一个子区间，于是另一区间的另外两个根就被漏判了。列举这些例子是为了说明增量搜索技术实际上是有缺陷的，有时会出错误。当然，大多数情况下它都能快速准确地找出多项式的根。如果需要性能更佳的技术，那么就应该研究其他求根方法。

*4.8 解决应用问题：系统稳定性

系统（system）是一个常见术语，经常用来表示通过指定输入来产生特定输出和动作的仪器或设备。工程应用中包含各种各样的系统，比如连接到流水线支架上的制冷设备，制造工厂中使用的机械臂和“子弹头”快速列车等。这里给出稳定系统（stable system）的一个定义：如果在一个系统中，合理的输入能产生合理的输出，那么该系统就是一个稳定系统。现在以机械臂的控制系统为例，一个合理的系统输入会指定机械手臂沿着一个合法方向移动。如果该输入引起机械臂操作不稳或者沿非法方向移动，那么这个系统就是不稳定的。系统设计的稳定性分析需要涉及系统的若干个动态属性，包括对于系统功能或者类型的讨论，这些已经超出本文的研究范围，不再阐述。系统分析的一个重要组成部分就是要确定多项式的根。通常来讲，实根和复根都需要求出，但是求解复根的方法涉及多项式的求导问题，而这就变成了更为复杂的数学问题。因此，在这里缩小问题范围，只讨论在给定搜索区间的情况下求解多项式的实根。这里只对三次多项式进行求解，但求解的方法可以轻松扩展到处理更高阶多项式的问题。

190

下面要设计程序来确定一个三次多项式的实根。首先令用户输入多项式的系数，待搜索

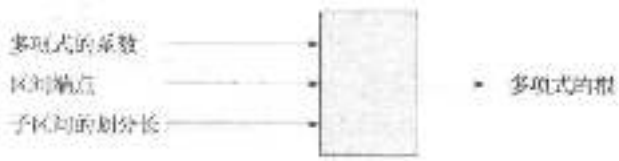
的区间以及子区间划分的步长。

1. 问题陈述

确定一个三次多项式的实根。

2. 输入/输出描述

如下面的I/O图所示，程序的输入有多项式的系数、区间端点以及子区间的划分步长；同时程序的输出是一个指定区间内的根：



3. 手动演算示例

在这里，使用下面的方程

$$y = 2x - 4$$

这个函数可以被描述成一个三次多项式，系数分别为 $a_0=0$，$a_1=0$，$a_2=2$，$a_3=-4$。如果将多项式的值置为0，便很容易发现它的根为2。为了检验增量搜索技术，首先使用一个区间步长，以使得根正好落在子区间的一个区间端点处；接下来改变区间步长，使得根没有落在子区间的区间端点。如果根落在区间端点，则很容易就能找到，因为此时多项式的值会非常接近于0。如果根落在一个子区间内部，两个区间端点处函数取值的乘积为负，则可以估计根值为该区间中点。

首先，考虑区间[1，3]，区间增量为0.5，则子区间和相应的区间信息如下所示：

子区间1：[1.0，1.5]

$f(1.0) \cdot f(1.5) = (-2) \times (-1) = 2$

该区间内无根

子区间2：[1.5，2.0]

当检验区间端点时，检查得出根为 $x=2.0$ 191

子区间3：[2.0，2.5]

当检验区间端点时，再次检查根为 $x=2.0$。一定要注意确保在程序中不会将此根重复认定两次

子区间4：[2.5，3.0]

$f(2.5) \cdot f(3.0) = 1 \times 2 = 2$

该区间内无根

现在考虑区间[1，3]，区间增量为0.3，则子区间和相应的区间信息如下所示：

子区间1：[1.0，1.3]

$f(1.0) \cdot f(1.3) = (-2) \times (-1.4) = 2.8$

该区间内无根

子区间2：[1.3，1.6]

$f(1.3) \cdot f(1.6) = (-1.4) \times (-0.8) = 1.12$

该区间内无根

子区间3： [1.6，1.9]

$f(1.6)\cdot f(1.9)=(-0.8)\times(-0.2)=1.6$

该区间内无根

子区间4： [1.9，2.2]

$f(1.9)\cdot f(2.2)=(-0.2)\times 0.4=-0.08$

此区间内有根，并估计根出现在区间中点处：

$x=\frac{1.9+2.2}{2}=2.05$

子区间5： [2.2，2.5]

$f(2.2)\cdot f(2.5)=0.4\times 1.0=0.4$

该区间内无根

子区间6： [2.5，2.8]

$f(2.5)\cdot f(2.8)=1.0\times 1.6=1.6$

该区间内无根

子区间7： [2.8，3.1]

注意到区间右端点超出了原始区间。在程序中，可以修改该子区间端点，使之可以结束在原始区间的右侧端点处

$f(2.8)\cdot f(3.0)=1.6\times 2.0=3.2$

该区间内无根

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤：

分解提纲

1）读取多项式系数，目的区间和区间步长。

192 2）利用子区间确定根落入的位置。

步骤1）提示用户输入必需的信息，然后程序读取数据。

步骤2）中需要使用循环体来计算子区间的端点，并确定根是落在区间端点还是在子区间内部。一旦确定了根落入的位置，便打印相应的区间信息。由于步骤2）中包含了很多操作，应该考虑使用函数来完成主要操作步骤，以保证main函数简短精炼。因为需要在程序中的几个地方计算三次多项式，所以可以用一个函数实现这部分功能。在每个子区间内都需要搜索根，因此搜索过程也非常适合通过函数来完成。图4-1中展示了解决方案的结构图。下面是提炼后的伪代码：

```
[提炼后的伪代码]
主函数：读取系数，区间端点a和b，
        以及区间增量
        计算子区间的个数，记为n
        将set置为0
        while k <= n-1
```

```
            计算子区间的左端点
            计算子区间的右端点
            check_roots (left, right, coefficients)
            k 自增 1
        check_roots (b, h, coefficients)
check_roots (left, right, coefficients):
        将 f_left 置为 poly (left, coefficients)
        将 f_right 置为 poly (right, coefficients)
        if f_left 接近于 0
            打印在左端点处的根值
        else
            if f_left · f_right < 0
            打印在区间中点处的根值
        return
poly (x, a3, a1, a2, a3):
        return a0x3 + a1x2 + a2x + a3
```

这里要注意的是，在伪代码中的 check_roots 函数中，需要检查子区间的左端点是否为根，但是右端点并不需要检查。这是因为在程序中必须要避免将同一个根重复认定两次——第一次检查的区间右端点在下一次就成了下一子区间的左端点。因为每次只检查子区间的左端点，所以在最后一个子区间中要记得检查区间右端点，因为它不是任何子区间的左端点。

伪代码中的执行步骤足够详细，可直接转化为 C 程序：

```
/*-----------------------------------------------------------*/
/*  程序 chapter4_8                                          */
/*                                                           */
/*  本程序通过增量搜索来估计一个多项式函数的实根              */
#include <stdio.h>
#include <math.h>
int main(void)
{
    /*  声明变量和函数原型  */
    int n, k;
    double a0, a1, a2, a3, a, b, step, left, right;
    void check_roots(double left,double right,double a0,
                     double a1,double a2,double a3);
    /*  用户输入  */
    printf("Enter coefficients a0, a1, a2, a3: \n");
    scanf("%lf %lf %lf %lf",&a0,&a1,&a2,&a3);
    printf("Enter interval limits a, b (a<b): \n");
    scanf("%lf %lf",&a,&b);
    printf("Enter step size: \n");
    scanf("%lf",&step);
    /*  检查根的区间  */
    n = ceil((b - a)/step);
    for (k=0; k<=n-1; k++)
```

193

```
    {
       left = a + k*step;
       if (k == n-1)
          right = b;
       else
          right = left + step;
       check_roots(left,right,a0,a1,a2,a3);
    }

    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数检查根的区间                                             */
void check_roots(double left,double right,double a0,
                 double a1,double a2,double a3)
{
    /*  声明实参和函数原型  */
    double f_left, f_right;
    double poly(double x,double a0,double a1,
                double a2,double a3);
    /*  估计区间端点并计算根的区间  */
    f_left = poly(left,a0,a1,a2,a3);
    f_right = poly(right,a0,a1,a2,a3);
    if (fabs(f_left) < 0.1e-04)
       printf("Root detected at %.3f \n",left);
    else
       if (fabs(f_right) < 0.1e-04)
          ;
       else
          if (f_left*f_right < 0)
             printf("Root detected at %.3f \n",(left+right)/2);
    return;
}
/*-----------------------------------------------------------------*/

/*  该函数计算一个三次多项式的值                                   */
double poly(double x,double a0,double a1,double a2,double a3)
{
    return a0*x*x*x + a1*x*x + a2*x + a3;
}
/*-----------------------------------------------------------------*/
```

194

5. 测试

使用手动演算示例中的数据测试程序，可以得到如下的执行结果。得到的根与之前手算出的结果相符：

```
Enter coefficients a0, a1, a2, a3:
0 0 2 -4
Enter interval limits a, b (a<b):
1 3
Enter step size:
0.5
Root detected at 2.000

Enter coefficients a0, a1, a2, a3:
0 0 2 -4
Enter interval limits a, b (a<b):
1 3
Enter step size:
```

```
0.3
Root detected at 2.050
```

使用4.7.2节上方给出的多项式来检验程序。改变区间和步长变量的值，以确保根不会总是落在区间端点处。

修改

1. 如果根没有落在区间端点处，区间长度就会影响对根值的估计。使用手动演算示例中的多项式，假设区间为[0.5，3]，用几个不同的步长来试验结果，步长分别是：1.1，0.75，0.5，0.3和0.14。
2. 使用$f_1(x)=x^3-3x^2-x+3$来测试程序，假设测试的区间[a, b]由用户输入，其a和b就是这个多项式的实根。
3. 使用$f_1(x)=x^3-3x^2-x+3$，找到一个合适的步长，使得程序在给定初始区间[-10，10]内会遗漏一些根。解释为什么程序会遗漏这些根。是程序设计出错了吗？
4. 修改程序，使其可以计算并判断一个四次多项式的实根的数值。
5. 用该程序去解决4.6节"修改"中的问题4。

[195]

*4.9 宏

在程序进入编译阶段之前，预处理器会通过相应的预处理命令来完成一些指定的操作，比如在程序中加入一些头文件，或者进行常量的符号定义等。除此之外，有一种简单的操作也同样可以通过一个叫作宏（macro）的预处理命令来指定，宏的一般形式如下：

```
#define macro_name(parameters) macro_text
```

其中的macro_text会在程序中替代所有引用了macro_name的地方。如果一个宏没有参数，它就是一个简单的常量符号。如果一个宏带参数，那么它可以替代一个简单函数。当宏的定义语句超过了一行时，则需要在除了最后一行之外的每一行末尾加上一个反斜线（\），来表示该行定义还未结束，将在下一行继续定义。

使用宏来代替函数的一个好处就是宏不是一个单独模块，这样一来程序的编译，链接/加载过程就被简化，同时执行时间也就减少了。在预处理过程中，宏的每一处引用都会被宏内容所替代。

为了说明这个过程，思考下面的程序，是将华氏温度转换为摄氏温度：

```
/*-----------------------------------------------------------*/
/*  程序chapter4_9                                           */
/*                                                           */
/*  本程序将华氏温度转换为摄氏温度                            */

#include <stdio.h>
#define degrees_C(x) (((x) - 32)*(5.0/9.0))

int main(void)
{
   /*  声明变量  */
   double temp;

   /*  输入一个华氏温度  */
   printf("Enter temperature in degrees Fahrenheit: \n");
   scanf("%lf",&temp);

   /*  转换成摄氏温度并打印结果  */
   printf("%f degrees Centigrade \n",degrees_C(temp));
```

```
    /* 退出程序 */
    return 0;
}
/*-----------------------------------------------------------*/
```

当程序中的 printf 语句被编译之后，宏内容就会代替宏指令引用。此时就相当于执行了如下语句：

196

```
printf("%f degrees Centigrade \n",(((temp) - 32)*(5.0/9.0)));
```

当这条语句执行时，temp 中的值就会严格按照宏命令中的公式由华氏温度转换成摄氏温度，而后打印在屏幕上。

需要注意的是，宏定义中的每一个独立参数以及 macro_text 都要用圆括号括住。这样当引用一个表达式作为实参的时候，宏指令才会准确地识别宏定义的各部分，并正常工作。为了说明这个问题，思考下面的宏指令，是将摄氏温度转换为华氏温度：

```
#define degrees1_F(x) ((x)*(9.0/5.0) + 32)
#define degrees2_F(x) x*(9.0/5.0) + 32
```

当这些宏指令的实参是一个变量时，两条语句都正常工作。

比如，下面的两条语句

```
max_temp1 = degrees1_F(temp);
max_temp2 = degrees2_F(temp);
```

就会被正确编译，其等价于这样的两个等式计算：

```
max_temp1 = ((temp)*(9.0/5.0) + 32);
max_temp2 = temp*(9.0/5.0) + 32;
```

但是，下面的这两条语句：

```
max_temp1 = degrees1_F(temp+10);
max_temp2 = degrees2_F(temp+10);
```

在编译的时候就会相当于执行下面的语句，而且并没有得到相同的结果：

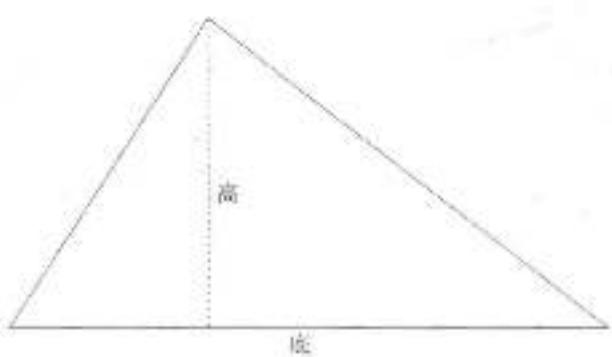
```
max_temp1 = ((temp+10)*(9.0/5.0) + 32);
max_temp2 = temp+10*(9.0/5.0) + 32;
```

因此，宏参数和 macro_text 都必须要加上括号，才能保证准确地计算出结果。

下面的宏指令用来计算指定底边和高的三角形的面积，如图 4-13 所示：

```
#define area_tri(base,height) (0.5*(base)*(height))
```

注意到在这条宏定义中，每一个参数都加上了圆括号，同时整个 macro_text 也加上了圆括号。



197

图 4-13 三角形

练习

给出相应的宏来计算下列值，并对每个宏都给出引用举例。

1. 计算正方形的面积：

$$A = 边^2$$

2. 计算长方形的面积：

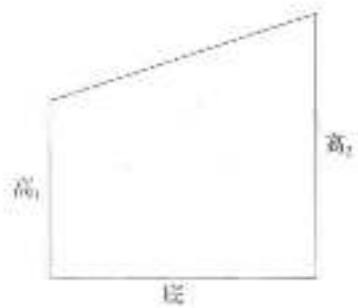
$$A = 边_1 \times 边_2$$

3. 计算平行四边形的面积：

$$A = 底 \times 高$$

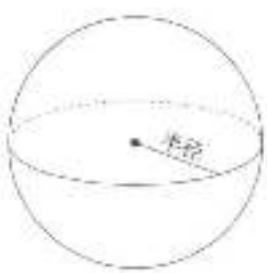
4. 计算梯形的面积：

$$A = 1/2\ 底 \times (高_1 + 高_2)$$



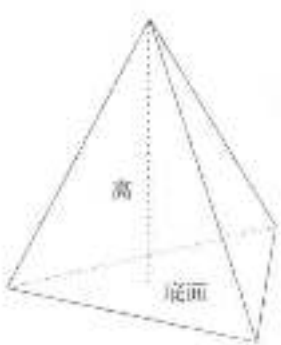
5. 计算球体的体积：

$$V = 4/3\pi \times 半径^3$$



6. 计算锥体的体积：

$$V = 1/3 \times 底面面积 \times 高$$

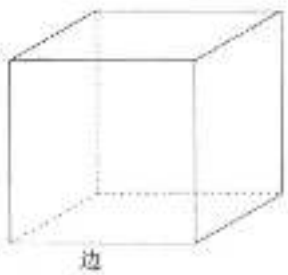


7. 计算正圆锥体的体积

$V = 1/3\pi \times 半径^2 \times 高$

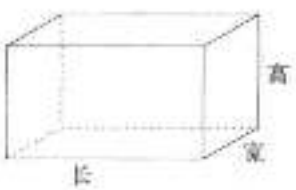
8. 计算立方体的体积：

$V = 边^3$



9. 计算一个长方体的体积：

$V = 长 \times 宽 \times 高$



*4.10 递归

如果一个函数在它的函数体中调用了它自身，则这样的函数叫作递归函数（recursive function）。对于有些问题，可以将其分解成类似的但规模较小的子问题，而对于每个子问题又可以将其分解成类似的但规模更小的子问题。将大问题划分成为相似的子问题，由子问题出发设计解决方案从而解决原始问题，这样的方法称为递归。将原始问题不断分解，直到子问题具有唯一的解，然后通过该唯一解来决定原始问题的解。由于递归函数需要重复地调用自身来将问题不断分解成子问题，有时会由于系统的原因对递归调用有次数限制，但是这些限制通常不会造成什么麻烦。

199

在下面的两个例子中，将会说明如何利用递归算法来解决问题。在例子中需要注意递归

方法主要分成两部分。第一，需要通过与原问题类似但规模更小的子问题来确定解决方案；第二，子问题必须要分解到具有一个唯一解时为止。

4.10.1 阶乘运算

递归方法的一个简单例子就是计算阶乘（factorial）。$k!$（读作 k 的阶乘）的定义如下：

$$k! = k(k-1)(k-2)\cdots 3\cdot 2\cdot 1$$

其中 k 是一个非负整数，并且 $0! = 1$。那么有

$$5! = 5\cdot 4\cdot 3\cdot 2\cdot 1 = 120$$

同样 5! 也可以由下面的步骤来计算：

$$\begin{aligned} 5! &= 5\cdot 4! \\ 4! &= 4\cdot 3! \\ 3! &= 3\cdot 2! \\ 2! &= 2\cdot 1! \\ 1! &= 1\cdot 0! \\ 0! &= 1 \end{aligned}$$

这样一来，就可将一个阶乘定义为包含了较小阶乘的式子。而较小的阶乘可以被持续分解直到 0!。在最后一个等式中将 0! 的值代入，然后由下至上依次将对应的值代入等式中，于是上面的阶乘变为：

$$\begin{aligned} 1! &= 1\cdot 1 \\ 2! &= 2\cdot 1! = 2 \\ 3! &= 3\cdot 2! = 6 \\ 4! &= 4\cdot 3! = 24 \\ 5! &= 5\cdot 4! = 120 \end{aligned}$$

这样，我们就为阶乘运算设计出了相应的递归算法。

下面的程序是计算一个阶乘的值，程序中包含了两个函数。第一个是非递归（迭代）函数，第二个是递归函数。由于阶乘的计算值会飞速增长，所以此处使用一个长整型变量来存储阶乘的值。显然这两个函数都是以类似的方式在 main 函数中被调用。

```
/*-----------------------------------------------------------*/
/*   程序 chapter4_10                                        */
/*                                                           */
/*   该程序通过阶乘计算来比较递归函数和非递归函数            */

#include <stdio.h>

int main(void)
{
   /*  声明变量和函数原型  */
   int n;
   long factorial(int k);
   long factorial_r(int k);

   /*  用户输入  */
   printf("Enter positive integer: \n");
   scanf("%i",&n);

   /*  计算阶乘并打印结果  */
   printf("Nonrecursive: %i! = %li \n",n,factorial(n));
   printf("Recursive: %i! = %li \n",n,factorial_r(n));
```

200

```
    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数通过循环体计算阶乘                                       */

long factorial(int k)
{
    /*  声明变量  */
    int j;
    long term;

    /*  用乘法计算阶乘  */
    term = 1;
    for (j=2; j<=k; j++)
        term *=j;

    /*  返回阶乘的值  */
    return term;
}
/*-----------------------------------------------------------------*/
/*  该函数使用递归方法计算阶乘                                     */

long factorial_r(int k)
{
    /*  使用递归调用，直到 k=0  */
    if (k == 0)
        return 1;
    else
        return k*factorial_r(k-1);
}
/*-----------------------------------------------------------------*/
```

判定条件 k == 0 是用来保证递归代码不会变成无限循环；该函数递归地调用自身，而伴随着每次调用，其参数都自减 1，一直到参数减为 0 为止。

当 k 值非常大时，阶乘 $k!$ 的值甚至会超过长整型的范围。此时一般应该使用 double 或 long double 类型来完成计算。另外在本章的末尾还会讨论关于 $k!$ 的近似计算方法。

[201]

4.10.2 斐波那契数列

斐波那契数列（Fibonacci sequence）是一组数列 $\{f_0, f_1, f_2, f_3, \cdots\}$，其中前两个数（$f_0$ 和 f_1）的值为 1，往后每一个后继元素的值都等于它前面两个元素之和。这样斐波那契数列的前几个数分别是：

$$1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad \cdots$$

斐波那契数列最早是在公元 1202 年提出的，现在已经被应用在众多领域，从生物学到电气工程学。例如，在研究兔子繁殖数量增长问题上就用到了斐波那契数列。

因为在数列中每一个新值的产生都依赖于它前面两个数值之和，因此计算斐波那契数列中的第 k 个值就是一个很典型的用递归方法解决的问题。下面的程序中分别以非递归和递归算法来计算一个斐波那契数：

```
/*-----------------------------------------------------------------*/
/*  程序 chapter4_11                                               */
/*                                                                 */
/*  该程序通过计算斐波那契数列比较递归函数和非递归函数             */
```

```
#include <stdio.h>

int main(void)
{
   /*  声明变量和函数原型  */
   int n;
   int fibonacci(int k);
   int fibonacci_r(int k);

   /*  用户输入  */
   printf("Enter positive integer: \n");
   scanf("%i",&n);

   /*  计算斐波那契数列并打印结果  */
   printf("Nonrecursive: Fibonacci number = %li \n",
           fibonacci(n));
   printf("Recursive:    Fibonacci number = %li \n",
           fibonacci_r(n));

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数使用非递归算法计算第 k 个斐波那契数                      */

int fibonacci(int k)
{
   /*  声明变量  */
   int term, prev1, prev2, n;
   /*  使用循环来计算第 k 个斐波那契数  */
   term = 1;
   if (k > 1)
   {
      prev1 = prev2 = 1;
      for (n=2; n<=k; n++)
      {
         term = prev1 + prev2;
         prev2 = prev1;
         prev1 = term;
      }
   }

   /*  返回第 k 个斐波那契数  */
   return term;
}
/*-----------------------------------------------------------------*/
/*  该函数使用递归算法计算第 k 个斐波那契数                        */

int fibonacci_r(int k)
{
   /*  声明变量  */
   int term;

   /*  递归地计算第 k 个斐波那契数，直到 k=1  */
   term = 1;
   if (k > 1)
      term = fibonacci_r(k-1) + fibonacci_r(k-2);

   /*  返回第 k 个斐波那契数  */
   return term;
}
/*-----------------------------------------------------------------*/
```

202

在递归函数中，条件 $k>1$ 可以防止函数进入死循环。

修改

1. 用程序 chapter4_12 计算下面序列的值：1!，2!，…，直到达到长整型数的极限。并观察，当计算的 $k!$ 值超过了系统极限时，会产生怎样的错误信息？
2. 修改程序 chapter4_12，用 double 类型代替整型来计算阶乘，就可以准确地计算出阶乘值。解释一下为什么使用的数据类型的精度位数决定了 $k!$ 的最大值？当使用系统中的 double 型去计算阶乘时所能支持的 $k!$ 的最大值是多少？

203

本章小结

大多数的 C 程序都得益于 C 语言丰富的库函数和自定义函数。通过函数可以实现软件的重用，并且抽象出一个一般性的解决方案，从而减少开发时间并提高软件质量。在本章已经提供了许多例子来说明如何使用自定义函数来解决问题，包括宏和递归函数的使用。同时，也给出了很多实战样例来解决实际问题，譬如生成随机数（整型或浮点型）和利用增量搜索技术找出多项式的实根。

关键术语

abstraction（抽象）
actual parameter（实际参数）
automatic class（自动类型）
call-by-reference（引用调用）
call-by-value（传值调用）
coercion of arguments（强制参数）
computer simulation（计算机仿真）
driver program（驱动程序）
external class（外部类型）
factorial（阶乘）
Fibonacci sequence（斐波那契数列）
formal parameter（形式参数）
function（函数）
function prototype（函数原型）
global variable（全局变量）
incremental search（增量搜索）
invoke（调用）
library function（库函数）
local variable（局部变量）
macro（宏）
modularity（模块化）
module（模块）
module chart（模块图）
programmer-defined function（自定义函数）
pseudorandom（伪随机）
random number（随机数）
random number seed（随机数种子）
recursion（递归）
register class（寄存器类型）
reusability（可重用性）
root（根）
scope（范围）
static class（静态类型）
storage class（存储类型）
structure chart（结构图）

C 语句总结

函数定义：

```
return_type function_name(parameter types)
{
    声明;
    语句;
}
```

返回语句：

```
return;
return (a + b)/2;
```

函数原型：

```
double sinc(double x);
double sinc(double);
void check_roots(double left,double right,double a0,
                 double a1,double a2,double a3)
```

宏：

```
#define degrees_C(x) (((x) - 32)*(5.0/9.0))
```

204

注意事项

1. 一个具有几个模块的程序要远比只有一整个长长的 main 函数的程序更具可读性、也更易理解。
2. 选择能直接反映出函数的功能用途的函数名。
3. 使用一个特别的行，如一行破折号，将自定义函数与 main 函数和其他自定义函数隔开。
4. 使用一致的函数顺序。如首先是 main 函数，随后按照函数被调用的顺序排列其他函数。
5. 在原型语句中使用参数标识符，以帮助说明参数的顺序和定义。
6. 在单独的行上列出函数原型，以方便区分。
7. 应该使用参数列表为函数传递信息，而不是使用外部变量。

调试注意事项

1. 如果编译器中提示的错误信息不好理解，那么可以试着在另一个编译器上编译一下，这样可以获得不同的错误信息。
2. 当调试一个很长的程序时，可以给某些部分的代码加上注释标识符（/* 和 */），这样就可以集中精力去调试另一部分代码。
3. 使用驱动程序来单独测试一个复杂函数。
4. 要确保使用返回语句返回的数据的类型与函数返回类型相匹配。如果有必要，可以用强制类型转换操作符将其转换成合适的类型。
5. 函数可以在 main 函数之前或之后定义，但不能定义在 main 函数里。
6. 要严格使用函数原型语句避免发生传参错误。
7. 在函数被调用前使用 printf 语句来生成实参的内存快照；同时在函数开始执行时用 printf 语句来生成形参的内存快照。
8. 使用函数时要仔细地匹配实参与形参的类型、顺序和实参的个数。
9. 在宏定义中，每个参数和整个宏定义内容都应该被相应的圆括号括住。
10. 在使用递归方法解决问题时偶尔可能会由于系统自身的限制而引发错误。

习题

简述题

判断题

判断下列陈述的正（T）误（F）。

1. 一个宏定义的函数体要用大括号包围。 T F

2. 参数列表包含了在函数中用到的所有变量。 T F

3. 在一次函数的传值调用中，函数不能改变实参的值。 T F

205 4. 静态变量在函数中声明，但是在每次函数调用中该静态变量的值始终会保留。 T F

多选题

选择每个问题的最佳答案。

5. 下列语句中是（　　）是合法的函数定义语句。

(a) function cube(double x)　　(b) double cube(double x)

(c) double cube(x)　　(d) cube(double x)

6. 在函数调用中，实参之间是用（　　）分隔。

(a) 逗号　　(b) 分号

(c) 冒号　　(d) 空格

7. 变量标识符可以被使用（或者被访问）的语句段称为（　　）。

(a) 全局的　　(b) 局部的

(c) 静态的　　(d) 作用域

程序分析题

分析下列函数，回答 8 – 11 题。

```
/*-----------------------------------------------------------*/
/*  该函数返回 0 或 1                                        */

int fives(int n)
{

   /*  计算并返回结果  */
   if ((n%5) == 0)
      return 1;
   else
      return 0;
}
/*-----------------------------------------------------------*/
```

8. fives(15) 的值是多少？

9. fives(26) 的值是多少？

10. fives(ceil(sqrt(62.5))) 的值是多少？

（提示：不需要计算器便可以计算出结果。）

11. 该函数对所有整数都是可用的吗？如果不是，它的计算极限是多少？

编程题

简单的仿真。在下面的这些问题中，使用函数 rand_int 和 rand_float 进行简单的仿真。

12. 编写程序来仿真抛硬币，允许用户输入抛硬币的次数。打印抛出正面的次数和抛出反面的次数。抛出正、反面的比例是多少？

206

13. 编写程序来仿真抛一个特制的硬币，即有一面被加重因而有 60% 的概率会出现正面。允许用户输入抛硬币的次数，打印抛出正面的次数和抛出反面的次数。

14. 编写程序来仿真掷骰子，其中 1 – 6 点所在的面都均匀分布。允许用户输入掷骰子的次数。然后分别打印出每个点出现的次数。观察在这 6 个点中的百分比分布是怎样的？

15. 编写程序来仿真同时掷两个骰子，其中 1 – 6 点所在的面均匀分布。允许用户输入仿真掷骰子的次

数。观察两个骰子点数之和等于8的次数所占的百分比是怎样的？

16. 编写程序来仿真使用记号从1～10的小球来抽奖。假设随机抽取三个球。允许用户输入仿真抽奖的次数。在仿真抽奖的结果中出现被抽取的三个球都是偶数的百分比是多少？在抽取的三个球中出现7号球的次数的百分比是多少？在仿真抽奖的结果中恰好出现1、2、3号球的次数百分比是多少？

元件可靠性。下面的问题用计算机仿真的方法评估元件配置的可靠性。使用在本章设计的函数rand_float来解决下列问题。

17. 编写程序，仿真图4-14所示的设计图，元件1的可靠性为0.8，元件2的可靠性为0.85，元件3的可靠性为0.95。使用5000次仿真，打印整个系统的可靠性评估结果。（该系统的解析可靠性为0.794。）

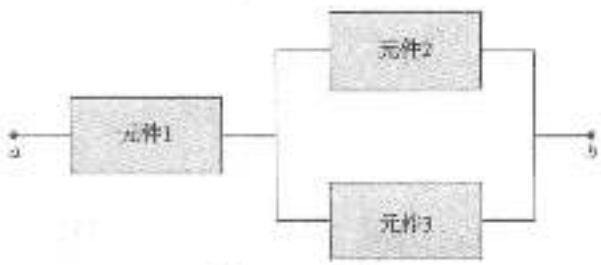


图4-14 配置1

18. 编写程序，仿真图4-15所示的设计图，元件1和元件2的可靠性为0.8，元件3和元件4的可靠性为0.95。使用5000次仿真，打印整个系统的可靠性评估结果。（该系统的解析可靠性为0.9649。）

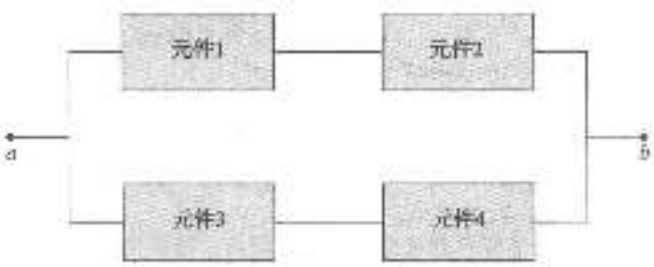


图4-15 配置2

19. 编写程序，仿真图4-16所示的设计图，全部由可靠性为0.95的元件组成。使用5000次仿真，打印整个系统的可靠性评估结果。（该系统的解析可靠性为0.99976。）

207

飞行模拟器的风速。下面是一组与飞行模拟器中的风速的计算机仿真相关的问题。假设一个特定区域的风速可以使用平均风速再加上一定范围的阵风值来模拟。例如，风速可能是10英里每小时，加上时速在-2～2英里的范围内随机变化的噪声数据（也就是阵风值），如图4-17所示。使用本章开发的rand_float函数。

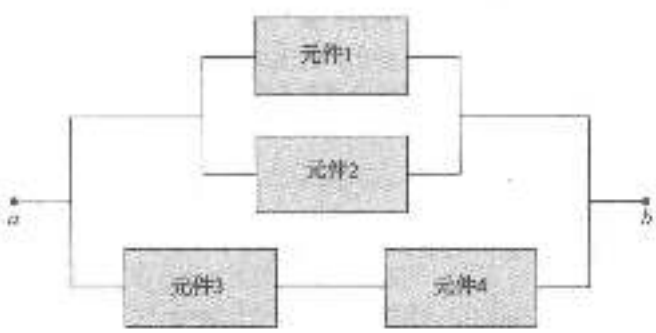


图4-16 配置3

20. 编写程序生成1小时内的风速仿真数据，并保存在文件wind1.dat中。文件中的每一行应该包括时间值（以秒为单位）和相应的风速值。从0秒开始计时，每隔10秒生成一组数据，并且数据文件最后一行应该对应3600秒。要求提示用户输入平均风速值和阵风值的范围。

21. 假设每次生成飞行模拟器的风速值时，都有0.5%的概率发生一次小型风暴。修改20题的解决方案，使其一旦遇到风暴，平均风速将增加10英里/小时，持续时间为5分钟。一个包含一次风暴的示例数据文件的图像如图4-18所示。

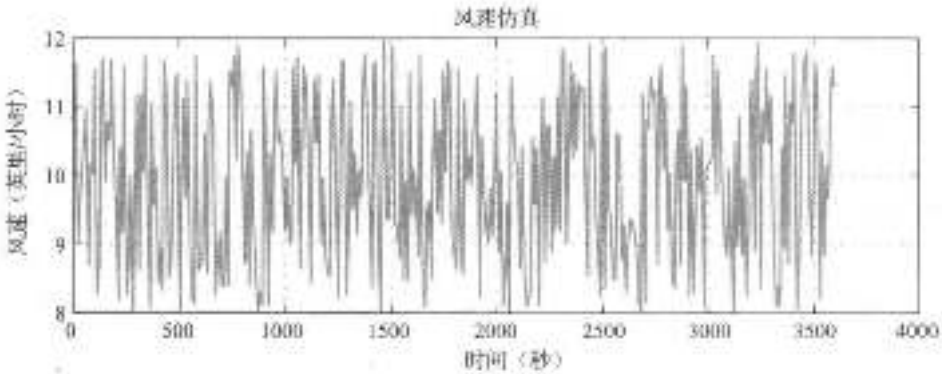


图 4-17 风速仿真

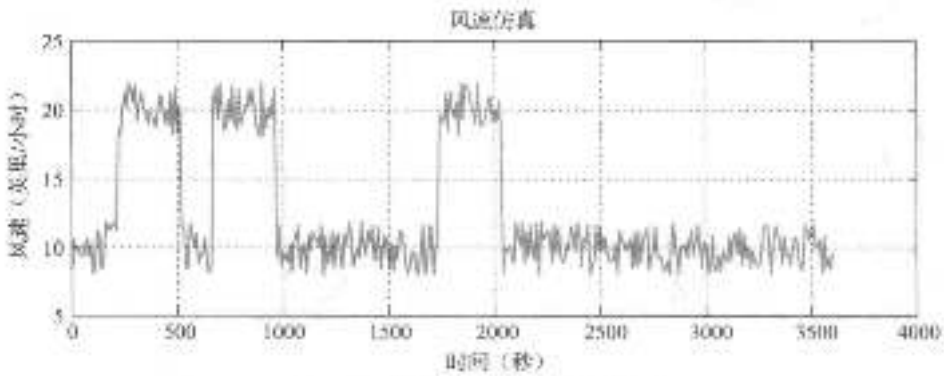


图 4-18 发生三次暴风的风速仿真

22. 假设在小型风暴发生的时间区间内，每次生成仿真数据时都有 1% 的可能性发生微爆气流。修改 21 题的解决方案，使其一旦发生微爆气流，平均风速在暴风值的基础上风速增加 50 英里/小时，持续时间为 1 分钟。图 4-19 展示了在暴风中发生微爆气流的示例数据文件图像。

208

23. 修改 21 题的程序，使用户输入发生风暴的概率。
24. 修改 21 题的程序，使用户输入风暴的持续时间（以分钟为单位）。
25. 修改 21 题的程序，使得风暴的持续时间在 3 – 5 分钟之间随机取值。

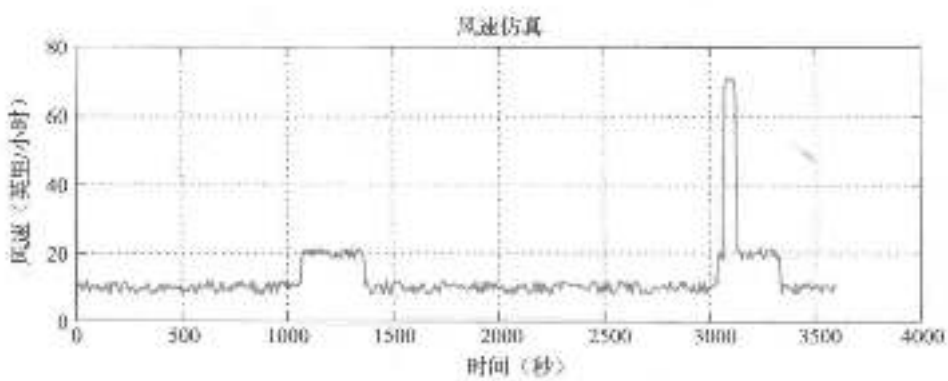


图 4-19 发生微爆气流时的风速仿真

函数实根。以下是一些关于求解函数实根的问题：

26. 编写程序，求解二次方程的实根，假设二方程式系数由用户输入。如果根为复根，则输出适当说明信息。

27. 修改 26 题中的程序，使得当方程具有复根时，程序计算并输出相应的实部和虚部。

28. 编写程序，求解下面数学函数的根。

$$f(x) = 0.1x^2 - x\ln x$$

209

假设对应的函数原型为

```
double f(double x);
```

修改 4.8 节中的程序，使其计算本题中函数 $f(x)$ 的根（而不再是求解多项式的根）。然后测试程序，求解函数在区间 [1，2] 中的根。

29. 修改 4.8 节中的程序，在用户指定的区间内求解下列函数的根：

$$f(x) = \mathrm{sinc}(x)$$

这里使用本章设计的 sinc 函数。

30. 在 4.8 节的程序中，我们首先寻找在区间端点处函数值异号的子区间；然后估计根在该子区间的中点上。现有一种更为精确的根估值方法，通常是采用通过函数值两点间连线 x 轴的交叉点，如图 4-20 所示。应用相似三角形，交点 c 可以用如下方程计算：

$$c = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$$

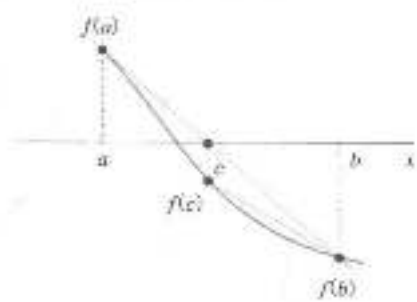


图 4-20 区间（a，b）中的直线交点

210

修改程序 chapter4_7，使用这种近似方式估计子区间上的根。

阶乘。下面是一些关于阶乘计算的问题。如果之前没有学习过有关递归的章节，可以阅读 4.10.1 节中关于阶乘定义的内容，然后学习使用非递归函数计算阶乘的方法。

31. 当 n 很大时，可以用斯特林公式近似计算阶乘 $n!$：

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

其中，e 是自然对数的基，取值近似为 2.718 282。设计一个整型函数计算此阶乘的近似值。假设相应的函数原型为

```
int n_fact(int n);
```

32. 假设现有 n 个不同物体。显然能够以多种不同的次序将这些物体排成一排。实际上，n 个对象可以得到 $n!$ 种次序，或称为排列方式（permutation）。如果在 n 个物体中选出 k 个，那么这 k 个物体就有 $n!/(n-k)!$ 种可能的排列方式。编写一个命名为 permute 的函数，函数输入为参数 n 和 k，随后计算从 n 个物体中一次取出 k 个共有多少种排列方式，并将此作为函数返回（举例来说，假设要从数集 {1，2，3} 中取出两个数字，可以得到的不同排列方式有 {1，2}，{2，1}，{1，3}，{3，1}，{2，3} 和 {3，2}）。假设相应的函数原型为

```
int permute(int n,int k);
```

33. 排列方式（32 题）考虑了排列次序，而组合却不考虑。因此，给定 n 个不同的物体，从中一次取 n 个只有一种组合，却有 $n!$ 种排列。从 n 个物体中一次取 k 个，共有 $n!/(k!)(n-k)!$ 种组合

（combination）。编写一个名为 combine 的函数，函数输入为参数 n 和 k，随后计算从 n 个对象中一次取 k 个共有多少种组合，并将此作为函数返回（例如，假设要从数集 {1，2，3} 中取出两个数字，可以得到的不同组合有 {1，2}，{1，3} 和 {2，3}）。假设相应的函数原型为

```
int combine(int n, int k);
```

34. 一个角的余弦值可以使用下列无穷级数计算：

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

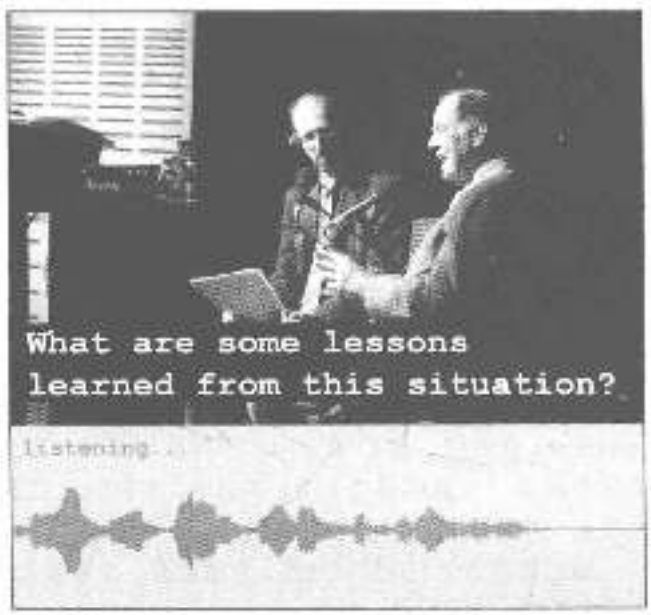
编写程序，从键盘读取角 x（以弧度表示），随后在函数中用该无穷级数的前 5 项计算角 x 的余弦值。最后分别输出使用该方法计算出的余弦值和使用 C 库函数计算得出的余弦值。

35. 修改 34 题中的程序，使用无穷级数中绝对值大于 0.000 1 的项来求取近似值。程序输出在级数近似过程中使用的级数项的个数。

211

数组和矩阵

犯罪现场调查：语音分析和语音识别



在关于生物特征识别的章节中，我们将语音识别作为一种生物特征进行了讨论。与此同时，还指出讲话者识别（通过语音信号进行身份识别）是一个非常具有挑战性的问题。该问题的部分难点在于，有很多因素可以导致语音发生变化。例如，当你感冒时，声音就会异于平常；当你变得疲倦或受到压力时，声音也会发生变化。目前在语音识别领域已经取得了相当可观的研究成果，商业系统中已经开发出了一些语音识别的成功实例（能够识别语音中的文字内容，但无法识别身份）。最新的第五代战斗机 Joint Strike Fighter 正在设计具备语音识别能力的系统。在起飞前的初始准备中，飞行员需要输入他的语音特征，随后在执行任务期间，该飞行员便可从自己的机口头下达指令信息。语音分析已经在商业系统和军事战场领域中得到了普遍应用。例如，语音分析技术可以通过分析音频信号来抑制信号中的噪声。同时它还可以分离出汽车引擎和飞机噪声之类的背景声音，从而确定信号的来源位置。如果有大量语音数据需要分析，那么可以设计一个自动化程序来执行各种工作任务，例如将电话信号转换为文本，确定语音所属语言，判断讲话者的性别，甚至确定语音中的地区方言等。本章将设计一个 C 程序来执行一些常见的语音分析任务。 [212]

学习目标

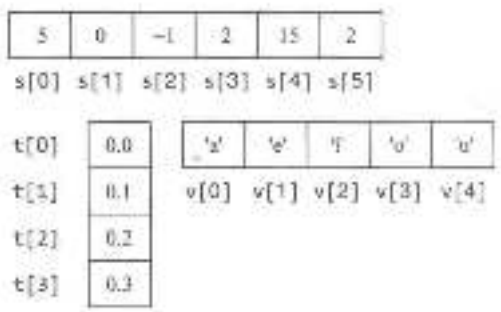
在本章，我们将学到以下解决问题的方法：

- 一维数组。
- 用于数据统计分析的自定义模块。
- 排序函数和搜索函数。
- 二维数组。
- 向量和矩阵计算。
- 求解联立方程组的方法。

5.1 一维数组

在解决工程问题时，对相关数据进行可视化处理非常重要。有时候数据只是单一数字，比如圆的半径。有时候数据是由一对数字组成的，比如平面上的坐标，其中一个表示 x 轴坐标，另一个表示 y 轴坐标。此外，有时需要处理一组相似的数据，但又不想为每个数据单独命名。例如，如果要用100个温度测量值进行计算工作，显然并不需要为每个测量值单独命名，所以要找到一种方法，实现只用一个标识符就能操作一组数据值。C语言中支持一种叫作数组（array）的数据结构，可以提供一种很好的解决方案。其中，一维数组（one-dimensional array）可以被形象化为排成一行或一列的一组数值，如下所示：

213



这里首先为数组分配一个标识符，然后用下标（subscript）来区分数组中的元素（element）或数值。在C语言中，下标从0开始，增量为1。因此，在上图所示的例子里，s数组中的第一个数值通过 s[0] 来引用，t 数组中的第三个数值通过 t[2] 来引用，v 数组中的最后一个数值通过 v[4] 来引用。

数组非常适合于存储和处理大批量数据，功能强大。因此有些人在很多不必要的情况下也倾向于在算法中使用数组。但是，数组使用起来要远比普通变量复杂，故而使程序代码更长，并且更难调试。因此，只有在必须将大量的数据集全部同时放入内存中的情况下才使用数组。

5.1.1 定义和初始化

数组是通过声明语句来定义的。标识符后面跟着一个包含在方括号内的整型表达式，该表达式指定了数组中的元素个数。要注意的是，数组中元素的数据类型是完全相同的。下面是声明语句的三个示例：

```
int s[6];
char v[5];
double t[4];
```

数组可以通过声明语句和程序语句两种方式来进行初始化。当使用声明语句初始化数组时，通过在花括号内用逗号分隔的形式来指定数组元素。对于示例数组 s，v 和 t，用下列语句进行定义和初始化：

```
int s[6]={5,0,-1,2,15,2};
char v[5]={'a','e','i','o','u'};
double t[4]={0.0,0.1,0.2,0.3};
```

如果初始化的数值序列小于数组长度，那么余下的值会被初始化为0。因此，如果要定义具有100个数值的整型数组，并且每个数值都被初始化为0，那么可以使用下列语句：

214

```
int s[100]={0};
```

如果一个数组没有指定大小，但是指定了初始化序列，则数组的大小就等于序列中的数值个数，如下所示：

```
int s[]={5,0,-1,2,15,2};
double t[]={0.0,0.1,0.2,0.3};
```

数组的大小必须是在声明语句中，或者通过方括号内的常数来指定，或者通过花括号内的初始化序列来指定。

数组也可以通过程序语句来初始化。例如，假设有一个 double 型数组 g，现在要使用序列 0.0，0.5，1.0，1.5，…，10.0 来对数组 g 进行初始化。因为共有 21 个数，将全部数值列在声明语句中就会显得过于冗长。因此，使用下列语句来实现该数组的定义和初始化：

```
/*  声明变量  */
int k;
double g[21];
...
/*  初始化数组g  */
for (k=0; k<=20; k++)
   g[k] = k*0.5;
```

这里需要特别注意的是，for 语句中的判定条件指定最后的下标是 20，而不是 21，因为数组元素是从 g[0] 到 g[20]。数组操作的一个常见错误就是指定的下标值大于数组最大有效下标值。一般这类错误很难被发现，因为它访问了数组之外的值。访问数组之外的值可能会产生执行错误，比如“段错误”或者“总线错误”。通常来讲，这类错误在程序执行时一般是检测不到的，但它会导致不可预知的结果，这是因为程序修改了数组之外的内存。所以说注意是否发生数组越界访问是非常重要的。在本书的示例程序中，对 for 循环的判定条件特别选用了最后一个值作为下标，以提醒自己设置判定条件时要注意避免错误。本例使用了条件 k<=20，而不是 k<21，虽然两个条件都可以正常运行，但是前者的警示效果更好。同时，对于一维数组的下标访问一般选用字母 k 来作为变量名。

数组经常用来存储从数据文件中读取的数据信息。例如，假设有一个名为 sensor3.txt 的数据文件，其中包含 10 组从地震仪中采集的时间和运动测量值。可以通过下列语句将这组数据分别读取到数组 time 和 motion 中：

```
/*  声明变量  */
int k;
double time[10], motion[10];
FILE *sensor;
...
/*  打开文件，并将数据读取到数组中  */
sensor = fopen("sensor3.txt","r");
for (k=0; k<=9; k++)
   fscanf(sensor,"%lf %lf",&time[k],&motion[k]);
```

[215]

练习

给出下列每组语句所定义的数组内容。

1. int x[10]={-5,4,3};

2. char letters[]={'a','b','c'};

3. double z[4];

```
...
z[1] = -5.5;
z[2] = z[3] = fabs(z[1]);
```

```
4. int k;
   double time[9];
   ...
   for (k=0; k<=8; k++)
      time[k] = (k-4)*0.1;
```

5.1.2 计算和输出

数组元素的计算和普通变量一样，但是必须使用下标来指明一个数组元素。例如，下面的程序从数据文件中连续读取100个浮点数存入数组y，计算数组的平均值，并将其存储在y_ave中。然后程序统计并输出数组y中大于平均值的元素个数。如果该程序是为了计算文件中数据的平均值，则没有必要使用数组，只需要在循环体中每次都将数值读取到相同变量里，并将其加到总和中。然而，为了计算数据中大于平均值的数值个数，每个值都要与平均值进行比较，因此需要使用数组来完成数值的多次访问。

```
/*-----------------------------------------------------------*/
/*  程序 chapter5_1                                          */
/*                                                           */
/*  本程序从数据文件中读取100个数并且确定其中大于平均值的元素个数  */

#include <stdio.h>
#define N 100
#define FILENAME "lab1.txt"

int main(void)
{
   /*  声明和初始化变量  */
   int k, count=0;
   double y[N], y_ave, sum=0;
   FILE *lab;
   /*  打开文件，将数据读取到一个数组中，并计算其总和  */
   lab = fopen(FILENAME,"r");
   if (lab == NULL)
      printf("Error opening input file. \n");
   else
   {
      /*  输入并处理数据  */
      for (k=0; k<=N-1; k++)
      {
         fscanf(lab,"%lf",&y[k]);
         sum += y[k];
      }

      /*  计算平均值并统计大于平均值的元素个数  */
      y_ave = sum/N;
      for (k=0; k<=N-1; k++)
         if (y[k] > y_ave)
            count++;

      /*  输出大于平均值的个数，并关闭文件  */
      printf("%d values greater than the average \n",count);
      fclose(lab);
   }
   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------*/
```

216

对于数组值的输出，是通过使用下标来指定想要输出的数组元素。例如，对于前一个例子，想要输出数组 y 的第一个元素和最后一个元素，可以使用如下语句：

```
printf("first and last array values: \n");
printf("%f %f \n",y[0],y[N-1]);
```

如果要在同一行输出数组 y 的全部 100 个元素，可以使用下面的循环来实现：

```
printf("y values: \n");
for (k=0; k<=N-1; k++)
   printf("%f \n",y[k]);
```

在处理较大的数组时，比如刚才程序中的数组 y，可能就需要在一行中输出几个数据然后再换行。这种情况可以使用取模运算符来判断是否要跳转到新的一行。如果要每组 5 个数值进行输出然后跳转到新的一行，可以通过下列语句来实现：

217

```
printf("y values: \n");
for (k=0; k<=N-1; k++)
   if (k%5 == 0)
      printf("\n %f ",y[k]);
   else
      printf("%f ",y[k]);
printf("\n");
```

也可以使用其他类似的语句向数据文件写入数组值。例如，使用文件指针 sensor 在数据文件的一行输出 y[k] 的值，可用如下语句实现：

```
fprintf(sensor,"%f \n",y[k]);
```

因为语句中包含了换行符，下一个值将会写在文件新的一行。

数组声明和循环访问数组元素都要用到数组元素的个数。如果要改变数组中元素的数目，那么程序的多个相应位置也需要同步修改。但是，如果使用符号常量来指定数组的大小，那么改变元素数目（即数组大小）所需的工作就会被大大简化。这种情况下，要改变数组大小只需修改相应的预处理命令即可。如果程序中包含许多模块，或者在几个程序员共同开发的编程环境中，大家统一使用这种编程风格十分重要。由符号常量定义数组大小的用法将会在后面的程序中加以说明。

表 5-1 给出了下标方括号的优先级顺序。方括号和圆括号的结合顺序在其他运算符之前。如果方括号和圆括号出现在同一条语句中，结合顺序是从左到右。如果两者嵌套，则最内部的优先运算。

表 5-1 运算符优先级

优先级	运算符	结合性
1	( )[ ]	从内向外
2	++--+-!(type)	从右至左（单目）
3	* / %	从左至右
4	+ -	从左至右
5	< <= > >=	从左至右
6	== !=	从左至右
7	&&	从左至右
8	||	从左至右
9	?:	从右至左

（续）

优先级	运算符	结合性
10	= += -= *= /= %=	从右至左
11		从左至右

练习

假设变量k和数组s由下列语句定义：

```
int k, s[]={3,8,15,21,30,41};
```

使用手动计算，确定下列每组语句的输出结果：

```
1. for (k=0; k<=5; k+=2)
     printf("%d %d \n",s[k],s[k+1]);
2. for (k=0, k<=5; k++)
     if (s[k]%2 == 0)
        printf("%d ",s[k]);
  printf("\n");
```

5.1.3 函数参数

要将数组信息传递到函数中，一般需要两个参数。一个参数指定数组，另一个参数指定需要的数组元素个数。可以根据需要来指定数组元素个数，这使得函数变得更加灵活。例如，如果函数参数规定了一个整型数组，则任何整型数组都可以用在该函数中；而这时有一个专门的参数指定了数组元素数目，以确保正确使用了数组长度。同时，函数每次所使用的数组元素个数可能不同。例如，将文件中的数据读入数组元素时，该数组的元素个数就取决于程序运行时指定的数据文件。通常来讲，数组一定要声明为可能出现的数据长度的最大值，而实际用到的元素个数则应该小于或等于最大值。

218

下面的程序从数据文件中读取数组，然后调用一个函数来确定数组中元素的最大值。变量npts用来指定数组元素的数目；npts的值应该小于或等于定义的数组大小100。调用的函数包含两个参数——数组名和数组元素个数，如函数原型所示。

本程序假设文件中数据的个数小于100，否则该程序将无法正常运行。所以必须要确保程序指定的数组长度不小于从文件中读入数据条数的最大值。

程序chapter5_2是为了展示数组如何作为函数参数来使用的。如果仅仅是为了确定文件中数据的最大值，则没有必要使用数组，因为在读取数据的过程中就完全可以找出最大数据值。

```
/*-----------------------------------------------------------*/
/*  程序 chapter5_2                                          */
/*                                                           */
/*  本程序从数据文件中读取数值，并用函数确定其中的最大值     */

#include <stdio.h>
#define N 100
#define FILENAME "lab2.txt"

int main(void)
{
   /*  声明变量和函数原型  */
```

```
   int k=0, npts;
   double y[N];
   FILE *lab;
   double max(double x[],int n);
   /*  打开文件，将数据读取到数组中  */
   lab = fopen(FILENAME,"r");
   if (lab == NULL)
      printf("Error opening input file. \n");
   else
   {
      while ((fscanf(lab,"%lf",&y[k])) == 1)
         k++;
      npts = k;
      /*  找到并输出最大值  */
      printf("Maximum value: %f \n",max(y,npts));

      /*  关闭文件并退出程序  */
      fclose(lab);
   }
   /*  退出程序  */
   return 0;
}

/*-----------------------------------------------------------------*/
/*  该函数返回具有 n 个元素的数组 x 的最大值                        */

double max(double x[],int n)
{
   /*  声明变量  */
   int k;
   double max_x;
   /*  确定数组的最大值  */
   max_x = x[0];
   for (k=1; k<=n-1; k++)
      if (x[k] > max_x)
         max_x = x[k];
   /*  返回最大值  */
   return max_x;
}
/*-----------------------------------------------------------------*/
```

219

使用简单值作为函数参数和使用数组作为函数参数是有显著差异的。当使用简单变量作为参数时，变量中的数值被传递（复制）到函数的形参中，因此原始变量的值不会发生改变，这种方式称为传值调用（call-by-value）引用。而当使用数组作为参数时，数组的内存地址被传递到函数中，而不是整个数组的值。因此，函数引用的是原始数组中的值，这种方式叫作传址调用（call-by-address）引用。因为函数访问的是原始数组值，所以一定要注意避免在不经意间修改函数中的数组值。当然，有些时候也不免会需要对数组值进行修改。本章后面的例子将会对此加以说明。

220

练习

假设定义了如下变量：

```
int k=6;
double data[]={1.5,3.2,-6.1,9.8,8.7,5.2};
```

使用本节中的 max 函数，给出下列每个表达式的计算值。

```
1. max(data,6);
2. max(data,5);
3. max(data,k-3);
4. max(data,k%5);
```

5.2 解决应用问题：飓风等级

飓风（hurricane）是伴有强风和强降雨的热带风暴（在西北太平洋地区被称为台风，在印度洋被称为旋风）。这些热带风暴（或者旋风）本质上是低气压中心，通常在夏季或初秋形成。这些大型旋转气团在卫星图像中很容易被观测到。由于这些风暴对居民区的潜在危害极其巨大，气象专家一直小心翼翼地对其进行追踪和监测。如果暴风风速在每小时 38 – 74 英里[1]之间，就称其为热带风暴；如果风速超过每小时 74 英里，就变成了热带气旋，或者飓风。沙佛 – 辛普森等级（Saffir-Simpson scale）依据风速等级定义了飓风强度的类别。本节将会详细地对等级定义进行讨论分析，然后设计程序，读取包含有当前风暴记录及最大风速的数据文件，分析并找出那些风速强到足以归为飓风的风暴记录，将分析结果打印为一份相关的信息报告。

沙佛 – 辛普森等级是根据暴风袭击居民区时可能产生的损害程度来对飓风强度进行分类的。5 种等级的主要特征如下所示：

等级 1	风速 74 – 95 英里 / 小时 4 – 5 英尺[2]的风暴潮 少量财产损失
等级 2	风速 96 – 110 英里 / 小时 6 – 8 英尺的风暴潮 中等财产损失
等级 3	风速 111 – 130 英里 / 小时 9 – 12 英尺的风暴潮 大量财产损失
等级 4	风速 131 – 155 英里 / 小时 13 – 18 英尺的风暴潮 极大的财产损失
等级 5	风速 155 英里 / 小时以上 18 英尺以上的风暴潮 灾难性的财产损失

221

表 5-2 展示了 1950 – 2002 年这半个世纪间全美发生的 12 次强飓风记录。

表 5-2 全美 1950 – 2002 年间的强飓风记录

飓风名称	年份	等级
Hazel	1954	4
Audrey	1957	4

[1] 1 英里 = 1 609.344 米

[2] 1 英尺 = 0.304 8 米

（续）

飓风名称	年份	等级
Donna	1960	4
Carla	1961	4
Camille	1969	5
Celia	1970	3
Frederic	1979	3
Allen	1980	3
Gloria	1985	3
Hugo	1989	4
Andrew	1992	5
Opal	1995	3

美国有史以来最具毁灭性的飓风是发生在2005年8月的卡特琳娜飓风。这次飓风的风速超过每小时125英里。然而，强风并不是这次飓风最具毁灭性的部分。强风伴随的暴雨引起了新奥尔良附近Pontchartrain湖发生洪灾，大量堤坝被冲垮。新奥尔良城内80%被洪水淹没，遇难者超过1800人。

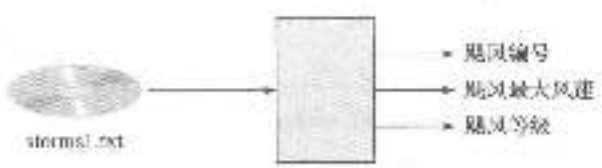
每年大约有100次风暴可能会转化为飓风。设计一个程序，首先从数据文件中读取当前的风暴信息。假设文件中的数据由风暴编号和对应的最高风速（英里/小时，即mph）构成。程序最后应该打印出一份报告，列出那些风速高到足以转化为飓风的风暴信息。除了风暴编号（编号是一个整数）以外，还要打印出最大风速以及相应的飓风强度等级。同时，在具有最大风速的飓风编号后面还要打印一个星号加以标注。

1. 问题陈述

利用包含当前风暴信息的数据文件，确定哪些风暴会转化为飓风。

2. 输入/输出描述

下面的I/O图展示了数据文件作为程序输入，相应的飓风信息作为输出。



3. 手动演算示例

假设数据文件中包含如下5条数据：

编号	最大风速
142	38
153	135
162	59
177	76
181	63

设计的程序应该输出如下报告：

足以转化为飓风的风暴信息

编号	最大风速 (mph)	等级
153*	135	4
177	76	1

其中编号后标星号的风暴具有最大风速。

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤。在打印可以转化为飓风的风暴信息时，并不需要使用数组。其实在从文件读取数据的时候就能直接确定打印信息，因为能否演变成飓风只取决于风速大小。然而，由于程序要求对具有最大风速的暴风编号标注星号，因此还是需要用数组来存储信息。在确定了最大风速后，就可以回溯数据，从而打印出标有星号的飓风信息。

分解提纲

1）将暴风数据读入数组，并确定最大风速。

2）计算飓风强度等级，并打印出可以转化为飓风的风暴信息，在具有最大风速的飓风编号后标出星号。

下面将确定飓风强度等级的操作步骤转化为函数：

```
[提炼后的伪代码]
主函数：if 数据文件无法打开
          打印错误信息
        else
          将数据读入数组，并确定最大风速和数组元素个数 npts
          将 k 置为 0
          while k <= npts-1
            if mph[k] > 74
              if mph[k] = max speed
                print id[k], *, mph[k], category (mph[k])
              else
                print id[k], mph[k], category (mph[k])
            k+1
        category (mph):
              category = 1;
              if mph >= 96
                category = 2
              if mph >= 111
                category = 3
              if mph >= 131
                category = 4
```

223

```
        if mph ≥ 155
          category = 5
```

现在，可以将伪代码中的步骤一一转化为C程序：

```
/*-----------------------------------------------------------------*/
/*  程序chapter5_3                                                 */
/*                                                                 */
/*  该程序从数据文件中读取暴风数据，并打印出一份飓风信息报告       */

#include <stdio.h>
#define N 500
#define FILENAME "storms1.txt"

int main(void)
{
   /*  声明变量并初始化  */
   int k=0, npts, id[N];
   double mph[N], max=0;
   FILE *storms;
   int category(double speed);

   /*  打开文件，将数字读入数组，并确定最大风速  */
   storms = fopen(FILENAME,"r");
   if (storms == NULL)
      printf("Error opening input file. \n");
   else
   {
      /*  读取数据并确定最大风速  */
      while ((fscanf(storms,"%d %lf",&id[k],&mph[k])) == 2)
      {
         if (mph[k] > max)
            max = mph[k];
         k++;
      }
      npts = k;
      /*  打印飓风信息报告  */
      if (max >= 74)
      {
         printf("Storms that Qualify as Hurricanes \n");
         printf("Identification    Peak Wind (mph)   Category \n");
      }
      else
         printf("No hurricanes in the file \n");
      for (k=0; k<=npts-1; k++)
         if (mph[k] >= 74)
            if (mph[k] == max)
               printf("%d*              %.0f                 %d \n",
                      id[k],mph[k],category(mph[k]));
            else
               printf("%d               %.0f                 %d \n",
                      id[k],mph[k],category(mph[k]));

      /*  关闭文件  */
      fclose(storms);
   }

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数确定飓风的级别                                           */
```

224

```
int category(double speed)
{
   /*  声明变量  */
   int intensity=1;

   /*  确定强度等级  */
   if (speed >= 96)
      intensity = 2;
   if (speed >= 111)
      intensity = 3;
   if (speed >= 131)
      intensity = 4;
   if (speed >= 155)
      intensity = 5;

   /*  返回飓风强度等级  */
   return intensity;
}
/*-----------------------------------------------------------------*/
```

225

5. 测试

下面将会使用手动演算示例中使用的数据来验证程序。程序的输出结果如下：

```
Storms that Qualify as Hurricanes
Identification   Peak Wind (mph)   Category
153*             135               4
177              76                1
```

修改

由前面的打印飓风强度报告的程序引出下列问题。

1. 修改程序，只打印出具有最大风速的飓风信息报告。
2. 修改程序，在原有的基础上，同时打印出数据文件中风暴的次数。
3. 修改程序，在原有的基础上，同时打印出数据文件中飓风的次数。
4. 修改程序，按照飓风的种类分别打印出飓风的次数。

5.3 解决应用问题：分子量

在很多科学系统或工程系统中，化学反应都发挥着重要作用。在石油和天然气的生产中，如果能充分了解并控制化学反应，石油工程师们就可以显著提高精炼效率。此外，在强磁场干扰的高温环境下，如果能充分理解完全电离气体的反应过程，就能够有效控制核聚变。在基因工程里，DNA 中氨基酸的鉴定对于新品种的合成技术具有关键作用。

对于包含化学反应的众多应用，计算化学式中的分子量是一项常见工作。现在要求编写一个程序，从键盘读入一个化学式，然后计算出相应的分子量。这里假设该程序会被用在基因工程实验室中，计算蛋白质中的氨基酸分子量。氨基酸包含 5 种元素，分别是氧（O）、碳（C）、氮（N）、硫（S）和氢（H）。例如，丙氨酸的化学式为 $O_2C_3NH_7$，也就是说，丙氨酸包含 2 个氧原子、3 个碳原子、1 个氮原子和 7 个氢原子。（表 5-3 中包含了各种氨基酸的原子组成。）该程序的输入为一组代表指定化学式的字符。输入原子名称时采用缩写形式，即 O、C、N、S 和 H，并且大小写均可。同时在每个元素后面还要跟上指定原子个数的数字。因此，丙氨酸的输入字符应该是 O2C3NH7 或 o2c3nh7。如果出现了这 5 种元素之外的元素，或者化学式以数字为开头，程序将会报错。另外，这 5 种元素的原子量如下所示：

氧（O） 15.999 4　　硫（S） 32.066
碳（C） 12.011　　氢（H） 1.007 94
氮（N） 14.006 74

226

表 5-3　氨基酸分子组成

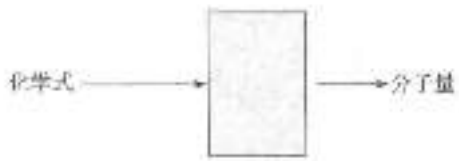
氨基酸	O	C	N	S	H
丙氨酸	2	3	1	0	7
精氨酸	2	6	4	0	15
天冬酰胺	3	4	2	0	8
天冬氨酸	4	4	1	0	6
半胱氨酸	2	3	1	1	7
谷氨酸	4	5	1	0	8
谷氨酸胺	3	5	2	0	10
甘氨酸	2	2	1	0	5
组氨酸	2	6	3	0	10
异亮氨酸	2	6	1	0	13
亮氨酸	2	6	1	0	13
赖氨酸	2	6	2	0	15
蛋氨酸	2	5	1	1	11
苯丙氨酸	2	9	0	1	11
脯氨酸	2	5	1	0	10
丝氨酸	3	3	1	0	7
苏氨酸	3	4	1	0	9
色氨酸	2	11	2	0	11
酪氨酸	3	9	1	0	11
缬氨酸	2	5	1	0	11

1. 问题陈述

计算一个氨基酸化学式的分子量。

2. 输入/输出描述

该程序的输入是一个化学式，由用户从键盘输入，输出是相应的分子量，结果展示在计算机屏幕上。



3. 手动演算示例

如果程序输入丙氨酸的化学式，即 O2C3NH7，那么对应输出的计算方式如下：

2 个氧原子：

$$2 \times 15.999\,4 = 31.998\,8$$

227

3 个碳原子：

$$3 \times 12.011 = 36.033$$

1个氮原子：

$$1 \times 14.006\ 74 = 14.006\ 74$$

7个氢原子：

$$7 \times 1.007\ 94 = 7.055\ 58$$

总分子量为89.094 12。

4. 算法设计

在设计具体算法之前，首先准备问题列出分解提纲，将问题分解成几个连续的解决步骤。

分解提纲

1）读取化学式。

2）计算分子量。

3）打印分子量。

在步骤1）中，程序从键盘读入字符，并将字符存储在一个整型数组中。在步骤2）中，首先检查字符（也叫文本解析（parsing））以确认每个元素的原子类型，然后再确定每个元素的原子个数。由于这个过程需要执行多个步骤，所以将其封装成一个函数。随后在主函数里，采用原子个数来以对应的原子量，再将这些值相加就可以得出化学式的分子量。要注意的是，这个过程需要先将原子个数从字符数的形式转换成相应的数值，以便进行下面的乘法操作。因为在ASCII码序列中的数字字符是连续的，所以通过将读入的数字字符减去一个'0'（字符0）就可以得到对应的数字的值。

步骤3）是打印最终的分子量。在这一步里，如果输入字符无法被正常解析，则打印一个错误信息。下面是提炼后的伪代码，以及对函数atomic_wt的设计：

```
[提炼后的伪代码]
主函数：向用户打印信息
        将k置为0
        while 字符未遍历完全
          读取formula[k]
          将k自增1
        将k置为0
        while 字符未遍历完全
          将当前字符转换为大写形式
          确定原子量
          如果原子后面跟着数字，确定该数值
          将原子量与数值相乘并加入总数
        打印分子量
```

[228]

现在将伪代码中的步骤转换成C程序。

```
/*-------------------------------------------------------------*/
/*  程序chapter5_4                                             */
/*                                                             */
/*  该程序通过氨基酸的化学式来计算其分子量                      */

#include <stdio.h>
```

```
#include <ctype.h>
#define NEWLINE '\n'

int main(void)
{
   /*  声明变量和函数原型  */
   int k=0, formula[20], n, current=0, done=0, d1, d2;
   double error=0, weight, total=0;
   double atomic_wt(int atom);

   /*  从键盘读入化学式  */
   printf("Enter chemical formula for amino acid: \n");
   while ((formula[k]=getchar()) != NEWLINE)
      k++;
   n = k;

   /*  确认元素并将对应的原子量求出分子量总和  */
   while (current<=(n-1) && done==0)
   {
      if (isalpha(formula[current]))
      {
         formula[current] = toupper(formula[current]);
         weight = atomic_wt(formula[current]);
         if (weight == 0)
            done = 1;
         else
         {
            if (current < n-1)
               d1 = isdigit(formula[current+1]);
            else
               d1 = 0;
            if (d1 && current<(n-2))
               d2 = isdigit(formula[current+2]);
            else
               d2 = 0;
            if (d1 && d2)
            {
               weight *= ((formula[current+1]-'0')*10 +
                          (formula[current+2]-'0'));
               current += 3;
            }
            else
               if (d1)
               {
                  weight *= (formula[current+1]-'0');
                  current += 2;
               }
               else
                  current++;
         }
         total += weight;
      }
      else
         done = 1;
   }
   /*  打印分子式和分子量  */
   printf("Formula: \n");
   for (k=0; k<=n-1; k++)
      putchar(formula[k]);
   printf("\n");
   if (done == 0)
      printf("Molecular weight: %f \n",total);
```

229

```
    else
       printf("Error in formula. \n");

    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数返回一个氨基酸分子中某个元素的分子量                      */

double atomic_wt(int atom)
{
    /*  声明并初始化变量  */
    int k=0, element[5]={'H','C','N','O','S'};
    double m_wt[5]={1.00794,12.011,14.00674,
           15.9994,32.066}, weight;

    /*  查找元素  */
    while (k<=4 && element[k]!=atom)
       k++;

    /*  返回相应的原子量  */
    if (k <= 4)
       weight = m_wt[k];
    else
       weight = 0;
    return weight;
}
/*-----------------------------------------------------------------*/
```

230

5. 测试

使用手动演算示例中的数据作为程序输入，得到对应的输出结果如下所示：

```
Enter chemical formula for amino acid:
O2C3NH7
Molecular weight:  89.094116
```

修改

本节主要讨论了关于计算一个氨基酸分子量的程序，而以下这些问题便是由此产生的。

1. 使用其他种类的氨基酸来验证程序。要求该氨基酸中的某种元素的原子数要超过9个。
2. 允许用户计算多种不同氨基酸的分子量，并且当输入一个句号时，程序停止。（要确保通知用户在计算完成时输入句号。）

5.4 统计测量

对工程试验中获得的实验数据进行分析是实验评估的重要部分。所谓分析可以是简单的数据计算，例如计算平均值，也可以是更复杂的数据分析。在数据计算和数据测量中大量使用统计计量方法，因为它们的数据集都具有明显的统计特征，并且在不同的数据集之间发生变化。例如，60°角的正弦值是一个精确值，不论何时计算结果都是相同的；但汽车上的加仑[⊖]汽油能行驶的里程数就是一个统计值，因为这个里程数依赖于很多因素，比如行车温度、行驶速度、路况以及行驶在山路上还是在沙漠中。

5.4.1 简单统计分析

在评估一组实验数据时，通常会计算最大值、最小值、平均值和中位数。本节将设计专

⊖ 1美制加仑=3.785 411 8升，1英制加仑=4.546 091 7升

[231]

们的函数，以数组作为函数输入，通过C语言代码计算这些统计数据。在后面的程序设计以及问题求解中，这些函数（存储在文件stat_lib.c中）将会非常有用。但是需要说明的是，这些函数均假设输入的数组至少具有一个有效数据。

1. 最大值和最小值

在前面的小节中已经介绍了查找一个数组中最大值的函数实现，所以现在可以编写一个类似的函数找到数组中的最小值。两个函数均假设数组是double型的，如果要指定计算整型值，只需对函数做些简单修改即可。

2. 平均值

平均值（mean value）通常用希腊符号μ来表示。它的计算等式中使用了求和符号，如下所示：

$$\mu = \frac{\sum_{k=0}^{n-1} x_k}{n} \tag{5.1}$$

其中

$$\sum_{k=0}^{n-1} x_k = x_0 + x_1 + x_2 + \cdots + x_{n-1}$$

即使一组数据中所有的数值都是整数，它们的平均值通常仍是一个浮点型值。为了计算一个具有n个元素的double型数组的平均值，可以使用下面的函数实现：

```
/*-------------------------------------------------------------------*/
/*  该函数返回一个具有n个元素的double型数组x的平均值                 */

double mean(double x[],int npts)
{
   /*  声明变量并初始化  */
   int k;
   double sum=0;
   /*  确定平均值  */
   for (k=0; k<=npts-1; k++)
      sum += x[k];
   /*  返回平均值  */
   return sum/n;
}
/*-------------------------------------------------------------------*/
```

要注意的是，变量sum在声明语句中被初始化为0。当然它也可以通过一个赋值语句被初始化为0。这两种情况下，都是在函数被调用时对sum进行的初始化。

3. 中位数

所谓中位数（median），就是在一组数值中居于中间位置的值，假设这些数值是有序的。如果数值个数为奇数，那么中位数就是位于中间的数值；如果数值个数为偶数，那么中位数就是位于中间位置的两个数的平均值。例如，对于{1，6，18，39，86}来说，中位数就是中间值18；对于{1，6，18，39，86，91}来说，中位数就是中间位置的两数平均值，即(18+39)/2，也就是28.5。假设一组有序数据存储在数组中，并且n为数组的元素个数。如果n为奇数，那么中间值的下标可以通过floor(n/2)来得到，即floor(5/2)，等于2。如果n为偶数，那么两个中间值的下标可以通过floor(n/2)-1和floor(n/2)来得

232 到，即 floor(6/2)-1 和 floor(6/2)，分别等于 2 和 3。

下列函数确定了一个数组中数值的中位数。假设该数组是有序的（升序或降序均可）。如果数组无序，那么可以在函数 median 中调用一个排序函数先将数组排序。排序函数的实现会在本章的后面部分介绍。

```
/*-----------------------------------------------------------------*/
/*  该函数返回一个有序数组 x 的中位数，数组元素个数为 npts          */

double median(double x[],int npts)
{
   /*  声明变量  */
   int k;
   double median_x;

   /*  确定中位数  */
   k = floor(npts/2);
   if (n%2 != 0)
      median_x = x[k];
   else
      median_x = (x[k-1] + x[k])/2;

   /*  返回中位数  */
   return median_x;
}
/*-----------------------------------------------------------------*/
```

使用前面讨论中给出的两组数据来手动演算该函数的执行过程。

5.4.2 方差和标准差

方差是最常用也是最重要的数据统计指标之一。在给出其数学定义之前，首先应该对方差的概念有一个直观理解。考虑两个数组 data1 和 data2，数值分布如图 5-1 所示。如果穿过数值分布的中间值画一条水平线，那么这条线大约是在 3.0 处。也就是说，两个数组的平均值是近似相等的。然而，这两个数组中的值很明显地表现出不同的分布特征。data2 中的值变化更剧烈，也更加偏离均值。所谓方差（variance），就是代表距离均值的平均偏离平方差；而标准差（standard deviation）就是方差的平方根。这样一来，数组 data2 的方差和标准差显然比数组 data1 中的要大。直观来看，方差（或标准差）越大，数值距离均值的波动范围也越大。

在数学上，方差用 σ^2 来表示，其中 σ 是希腊符号 sigma。那么在数组 x 中的这列数据的方差可以通过下面的等式来计算：

$$\sigma^2 = \frac{\sum_{i=0}^{n-1}(x_i - \mu)^2}{n-1} \tag{5.2}$$

这个等式乍看起来有点吓人，不过如果仔细观察，会发现其实式子的构成很简单。其中 233 $x_i-\mu$ 就是 x_i 同均值之差，或者说是 x_i 距离均值的偏离度。这个差值被取平方以便于得到一个正值。然后将所有数值计算出的平方差相加，再除以 $n-1$，取一个近似平均值。方差的定义有两种形式：分母是 $n-1$，即抽样方差；分母是 n，即总体方差。大多数的工程应用都采用抽样方差，如等式（5.2）所示。该等式计算的是数据相对于均值的平均平方差。而标准差定义为方差的平方根：

$$\sigma = \sqrt{\sigma^2} \tag{5.3}$$

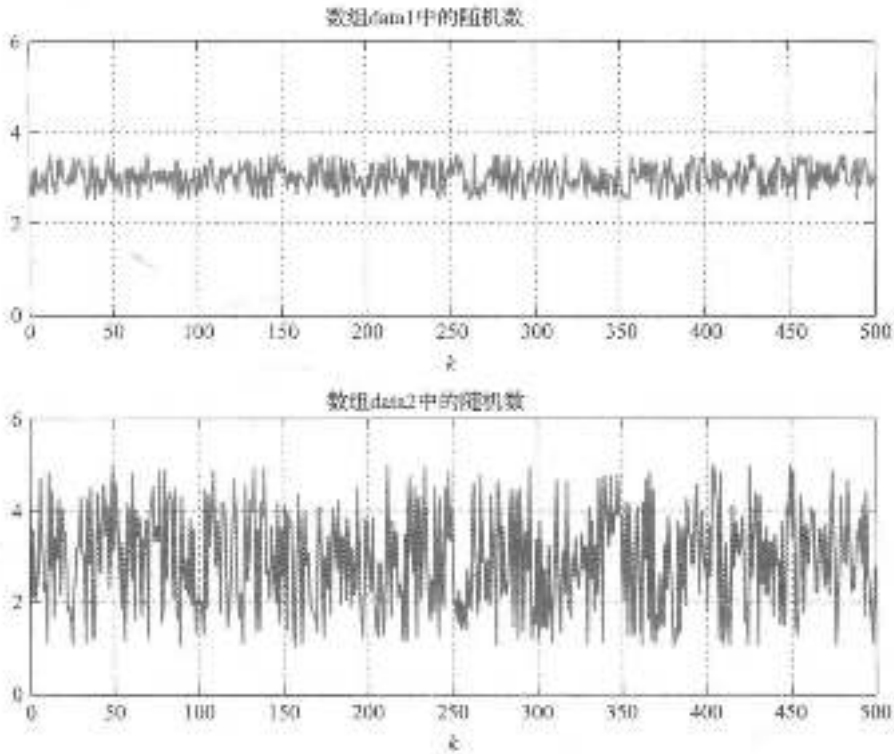


图 5-1 随机序列

方差和标准差在分析工程数据时常常会被用到，所以下面给出计算这两个统计值的函数。注意到，由于计算标准差的函数中会引用 variance 函数，而在 variance 函数中会引用 mean 函数，所以，这些函数必须包含相应的函数原型声明语句。另一方面，需要注意应保证数组中至少有两个元素，否则函数 variance 将会把零做除数。计算方差和标准差的函数如下：

```
/*-------------------------------------------------------------------*/
/*  该函数返回数组 x 的方差，其中数组元素个数为 npts                  */
double variance(double x[],int npts)
{
   /*  声明变量和函数原型  */
   int k;
   double sum=0, mu;
   double mean(double x[],int npts);

   /*  计算方差  */
   mu = mean(x,npts);
   for (k=0; k<=n-1; k++)
      sum += (x[k] - mu)*(x[k] - mu);

   /*  返回方差  */
   return sum/(npts-1);
}
/*-------------------------------------------------------------------*/
/*  该函数返回数组 x 的标准差，数组元素个数为 npts                    */

double std_dev(double x[],int npts)
```

234

```
{
   /* 声明函数原型 */
   double variance(double x[],int npts);

   /* 返回标准差 */
   return sqrt(variance(x,npts));
/*-----------------------------------------------------------------*/
```

5.4.3 自定义头文件

本小节设计的函数在解决工程问题中会经常用到。为了便于使用，可以创建一个自定义头文件，将这些函数原型语句全部包含其中。这样一来，便不需要在每一个 main 函数中都逐个包含所有的原型语句，可以使用预处理命令来包含该自定义头文件即可。

该自定义头文件命名为 stat_lib.h，它将包含下面的函数原型语句：

```
double max(double x[],int n);
double min(double x[],int n);
double mean(double x[],int n);
double median(double x[],int n);
double variance(double x[],int n);
double std_dev(double x[],int n);
```

接下来，只需要声明如下语句，即可在 main 函数中包含以上所有函数原型语句：

```
#include "stat_lib.h"
```

该自定义头文件的使用将会在下一节中说明。

除了使用 include 语句来获取自定义头文件之外，主程序编译时还必须加入文件 stat_lib.c，该文件中包含了全部统计学函数的实现。这个向程序编译中加入一个新文件的过程是系统相关的，有的需要在程序工程中进行配置，有些需要在控制编译和连接/加载操作的系统命令中加入该文件名。

[235]

练习

假设数组 x 的定义和初始化如下：

```
double x[]={2.5,5.5,6.0,6.25,9.0};
```

通过手动计算来确定下列函数的返回值：

1. max(x,5)
2. median(x,5)
3. variance(x,5)
4. std_dev(x,5)
5. min(x,4)
6. median(x,4)

5.5 解决应用问题：语音信号分析

语音信号是一种声音信号，可以用麦克风转换为电信号。随后电信号被转换成一串数字，来表示电信号的振幅水平。将这些数字存储在数据文件里，就可以通过计算机程序来分析对应的语音信息。例如，有些语音识别程序是对一些人类说话的声音信息数据进行处理，从中分析出 zero，one，…，nine 等这些单词的语音信号，从而正确地识别到语音中提到的数字。

图 5.2 是数字 0 对应的单词 zero 的语音信号分布。像这种复杂信号的分析工作，通常

是要先计算一些统计计量数据（上一小节中讨论的）。另外，关于语音信号还有一些其他的计量数据，如平均振幅（magnitude），又称为平均绝对值，其计算公式如下：

$$\text{平均振幅} = \frac{\sum_{k=0}^{n-1} |x_k|}{n} \tag{5.4}$$

其中，n 是数据的个数。

在语音分析中使用的另一个指标是信号的平均功率（power），也就是数据的平方取均值：

$$\text{平均功率} = \frac{\sum_{k=0}^{n-1} x_k^2}{n} \tag{5.5}$$

语音信号中过零点（zero crossing）的个数也是很有用的统计指标。这个过零点的值实际上就是语音信号从负值到正值，或者从正值到负值的转换次数。从非零值到零的转换并不算作过零点。 236

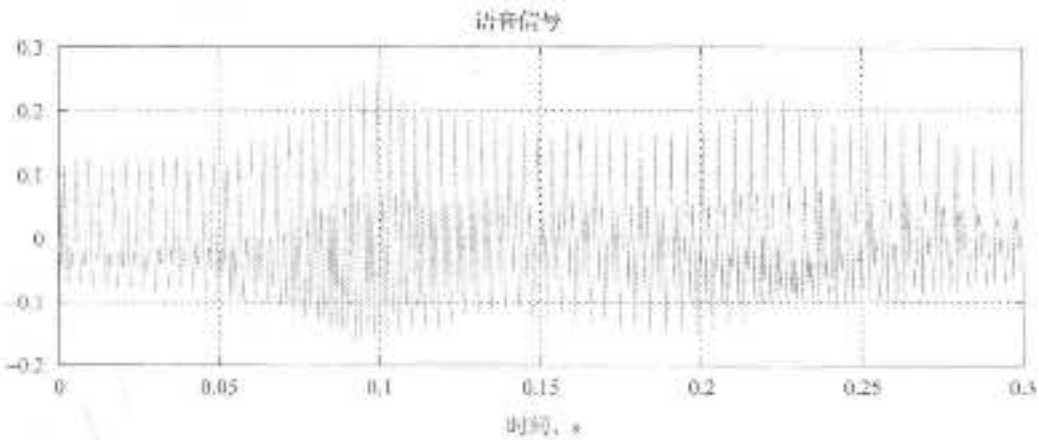


图 5-2 单词 zero 的语音信号分布

编写程序，从一个名为 zero1.txt 的数据文件中读取一段语音信号。该文件包含着能够表示语音 zero 的信号值。文件中的每一行都包含一个独立的值来表示从麦克风接收的语音信号的测量值，其中测量间隔为每 0.000 125 秒一次，所以每一秒语音数据可以由 8000 个测量值表示。该数据文件只包含有效数据，没有任何头、尾标记行；文件中最多包含 2500 个数据值。计算并打印出文件数据对应的统计计量值：均值、标准差、方差、平均功率、平均振幅和过零点的个数。

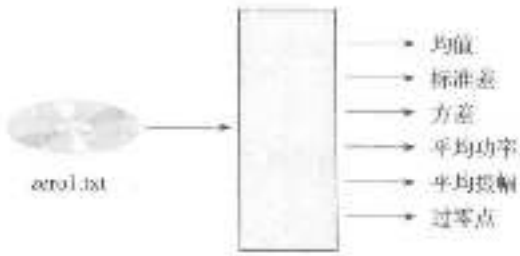
1. 问题陈述

对一段语音信号计算如下统计计量值：均值、标准差、方差、平均功率、平均振幅和过零点的个数。

2. 输入 / 输出描述

下列 I/O 图显示了数据文件作为程序输入，同时相应的统计计量值作为输出。

237



3. 手动演算示例

首先进行手动演算，现假设文件包含如下数据：

2.5 8.2 -1.1 -0.2 1.5

使用计算器，根据这些数据可以计算出如下值：

$$\text{均值} = \mu = \frac{2.5 + 8.2 - 1.1 - 0.2 + 1.5}{5} = 2.18$$

$$\text{方差} = [(2.5-\mu)^2 + (8.2-\mu)^2 + (-1.1-\mu)^2 + (-0.2-\mu)^2 + (1.5-\mu)^2]/4 = 13.307$$

$$\text{标准差} = \sqrt{13.307} = 3.648$$

$$\text{平均功率} = \frac{(2.5)^2 + (8.2)^2 + (-1.1)^2 + (-0.2)^2 + (1.5)^2}{5} = 15.398$$

$$\text{平均幅值} = \frac{|2.5| + |8.2| + |-1.1| + |-0.2| + |1.5|}{5} = 2.7$$

过零点的个数 = 2

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤：

分解提纲

1）读取语音信号并写入数组。

2）计算并打印相应的统计计量值。

步骤1）中包括读取数据文件，确定数据点的个数。步骤2）中包括计算并打印统计计量值；尽量使用已经开发好的函数帮助计算。在图4-1中展示的结构图已经对在main函数中引用自定义函数的例子做出说明。对main函数以及相应统计函数提炼后的伪代码如下所示：

[提示信息]

主函数：从数据文件中读取包括含噪声在内的数据点的个数（npts）

计算并打印数据均值

计算并打印标准差

计算并打印方差

计算并打印平均功率

计算并打印平均幅值

计算并打印过零点

统计函数：

ave_power (x, npts):

将 sum 置为 0

将 k 置为 0

while k <= npts-1

将 (x[k]^2) 加到 sum 中

k 自增 1

返回 sum/npts

ave_magn (x, npts):

将 sum 置为 0

将 k 置为 0

while k <= npts-1

将 |x[k]| 加到 sum 中

k 自增 1

返回 sum/npts

crossings (x, npts):

将 count 置为 0

将 k 置为 0

while k <= npts-2

if x[k] * x[k+1] < 0

count 自增 1

k 自增 1

返回 count

238

```
/*-----------------------------------------------------------------*/
/*  程序 chapter5_5                                                */
/*                                                                 */
/*  该程序计算一段语音信号的相关统计计量值                          */

#include <stdio.h>
#include <math.h>
#include "stat_lib.h"
#define MAXTNUM 2500
#define FILENAME "zero1.txt"

int main(void)
{
   /*  声明变量和函数原型  */
   int k=0, npts;
```

```
    double speech[MAXIMUM];
    FILE *file_in;
    double ave_power(double x[],int npts);
    double ave_magn(double x[],int npts);
    int crossings(double x[],int npts);
    /*  从数据文件中读取信息  */
    file_in = fopen(FILENAME,"r");
    if (file_in == NULL)
       printf ("Error opening input file. \n");
    else
    {
       while ((fscanf(file_in,"%lf",&speech[k])) == 1)
          k++;
       npts = k;

       /*  计算并打印统计值  */
       printf("speech statistics \n");
       printf("     mean: %f \n",mean(speech,npts));
       printf("     standard deviation: %f \n",
              std_dev(speech,npts));
       printf("     variance: %f \n",variance(speech,npts));
       printf("     average power: %f \n",
              ave_power(speech,npts));
       printf("     average magnitude: %f \n",
              ave_magn(speech,npts));
       printf("     zero crossings: %d \n",
              crossings(speech,npts));

       /*  关闭文件  */
       fclose(file_1);

    }
    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数返回数组x的平均功率，其中数组元素个数为npts              */

double ave_power(double x[],int npts)
{
    /*  声明并初始化变量  */
    int k;
    double sum=0;

    /*  确定平均功率  */
    for (k=0; k<=npts-1; k++)
       sum += x[k]*x[k];
    /*  返回平均功率  */
    return sum/npts;
}
/*-----------------------------------------------------------------*/
/*  该函数返回数组x的平均振幅值，其中数组元素个数为npts            */

double ave_magn(double x[],int npts)
{
    /*  声明并初始化变量  */
    int k;
    double sum=0;

    /*  确定平均功率  */
    for (k=0; k<=npts-1; k++)
       sum += fabs(x[k]);

    /*  返回平均振幅值  */
```

```
    return sum/npts;
}
/*-----------------------------------------------------------------*/
/*  该函数返回数组 x 中过零点的个数，其中数组元素个数为 npts          */

int crossings(double x[],int npts)
{
    /*  声明并初始化变量  */
    int count=0, k;

    /*  确定过零点的个数  */
    for (k=0; k<=npts-2; k++)
        if (x[k]*x[k+1] < 0)
            count++;

    /*  返回过零点的个数  */
    return count;
}
/*-----------------------------------------------------------------*/
```

要注意的是，个数为 n 的数据点可能出现的过零点最多为 n-1 个，这是因为每一个过零点是由两个数据点来确定的。因此，要检测的最后一组数值的下标应该为 n-2 和 n-1。

5. 测试

该程序的执行需要向程序中加入前面小节开发的头文件 stat_lib.h 和文件 stat_lib.c。下面的数据是根据语音"零"的采集数据文件 zero1.txt 计算得来的：

```
Speech Statistics
     mean: -0.000208
     standard deviation: 0.077035
     variance: 0.00534
     average power: 0.005932
     average magnitude: 0.060567
     zero crossings: 124
```

241

修改

现在大多数计算机都具有连接麦克风的端口。利用语音信号采集工具，同本章开发的程序配合使用（语音信号采集工具可以请老师或实验助理帮忙在互联网上搜索下载）。自己采集单词 zero 的三份语音文件，同时分别采集数字 one、two 和 three 的语音文件。

1. 将采集的单词 zero 的三个文件依次在程序中执行。要注意，在来自同一讲话者的不同语音信号之间，这些统计数据可能会产生显著变化。
2. 将采集的单词 one、two 和 three 的语音文件依次在程序中执行。这三个不同单词的统计数据之间的差别应该比同一数字的多次统计差别要大得多。
3. 在处理语音信号的时候，通常会计算出均值，然后将文件中的每个值都减去均值。这样一来就转换成了一列均值为 0 的数据。将这一特点加入程序中，以使得除了均值以外的统计指标都使用该均值为 0 的数据来计算。
4. 修改程序，在输出结果中增加一行，打印出该数据文件中数据点的个数。
5. 修改程序，在输出结果中增加一行，打印出该数据文件中数据点的最大值。

5.6 排序算法

在进行数据分析时，将一组数值进行排序（sorting）是一项常规操作。很多相关书籍中

已经详细介绍了多种不同的排序算法。之所以列出各种排序算法，一个重要原因就是没有一种算法可以称得上是“最佳”排序算法。当数据本身已经接近正确顺序的时候，有些算法会非常快，但是如果数据是随机散落的，或者是接近于逆序排列的，这些算法马上会变得极其低效。因此，为特定应用选择最佳排序算法时，通常需要对原始数据进行了解。本书并没有对全部的排序算法做详细讲解，而是专门介绍其中的两种算法。在本节将会介绍选择排序，该算法易于理解，并且代码实现简单。在第6章，将会对基于递归原理的快速排序算法进行详细讲解，届时也会用到第6章的相关知识。

选择排序（selection sort）算法的基本原理是，首先在数组中找到一个最小值，并将其与数组中的第一个元素互换位置。然后从第二个元素开始，继续寻找最小值，并将其与第二个元素互换位置。这个过程将一直持续到倒数第二个元素，并将其与最后一个元素相比较，如果顺序不对则互换位置。此时，整个数组将变成升序排列。下面的图示说明了整个过程：

原始顺序：

5	3	12	8	1	9

最小值与第一个元素互换：

242

1	3	12	8	5	9

下一个最小值与第二个元素互换：

1	3	12	8	5	9

下一个最小值与第三个元素互换：

1	3	5	8	12	9

下一个最小值与第四个元素互换：

1	3	5	8	12	9

下一个最小值与第五个元素互换：

1	3	5	8	9	12

最后数组变成升序排列：

1	3	5	8	9	12

下面函数中的步骤很简单，但还是很有必要用上面的例子来执行一遍函数。在函数执行过程中，要注意循环中下标k、m和j的变化过程。同样要注意的是，交换两个变量的值需

要三步操作（而非两步）。因为该函数没有返回值，所以返回类型是 void。下面就是函数的代码实现：

```
/*-----------------------------------------------------------------*/
/*  该函数将数组 x 按升序排序，其中数组元素个数为 npts            */

void sort(int x[],int npts)
{
   /*  声明变量  */
   int k, j, m;
   double hold;

   /*  执行选择排序算法  */
   for (k=0; k<=npts-2; k++)
   {
      /*  将最小值与下一个数组元素值互换  */
      m = k;
      for (j=k+1; j<=npts-1; j++)
         if (x[j] < x[m])
            m = j;
      hold = x[m];
      x[m] = x[k];
      x[k] = hold;
   }
   /*  无返回值  */
   return;
}
/*-----------------------------------------------------------------*/
```

243

如果要将此函数修改为将数组按降序排序，应该在内循环中将寻找最小值改为寻找最大值。

该排序函数的函数原型语句为：

```
void sort(int x[], int npts);
```

值得注意的是，该函数直接改变了原始数组。如果要保持原数组的排列顺序，就要在执行函数前，将此数组复制到另一个数组中。这样，就能同时得到原始序列的数组和排序之后的数组了。

字符的排序需要按照编码表（如 ASCII 码）从低到高的顺序来形成序列，这种顺序序列叫作整理序列（collating sequence）。将 ASCII 字符从小到大排序实际上就是字母表顺序。

修改

1. 编写一个 main 函数，首先初始化一个数组，然后引用上面的 sort 函数，最后打印排序完成后的数组结果。
2. 修改 sort 函数，使其按降序排列数组。使用问题 1 中的程序来检验修改后的函数。

5.7 搜索算法

对数组的另一个常用操作就是在数组中搜索一个特定的值。在实际应用中经常会关注数组中的某些信息，比如某个特定数值是否存在于数组中，它在数组中出现了几次，以及它第一次出现在什么位置。每一种搜索方式最终都能得出一个确定的数值，因此可以依据搜索方

式设计出相应的函数，并将结果作为函数的返回值。在本节将会设计几种数组搜索函数，这样一来，当程序中需要实现一个查找功能时，便可以将这些搜索函数略加改动或者直接运用其中。

搜索算法一般分为两类：一类是对无序数列的搜索，另一类是对有序数列的搜索。

5.7.1 无序数列

首先考虑无序数列的搜索，假设数组的元素并不需要排成升序序列（或者其他有助于搜索数组的序列）。搜索无序数列的算法其实就是简单的顺序搜索（sequential search）：首先检查第一个元素，接着检查第二个元素，以此类推。实现这个搜索函数的方式不止一种。可以将其设计成一个返回值是整型的函数，如果查找成功，则返回值为所求的元素在数列中的位置；如果查找失败，则返回值为 -1。也可以将其设计成返回值为所求元素在数组中出现的次数。此外，还可以将其设计成一个逻辑函数，当所求元素在数组中时返回 true（1），所求元素不在数组中时返回 false（0）。根据上面的思路，可以用程序将这些函数分别实现出来。这里已经设计了一个函数，其返回值或为所求元素在无序数组中的位置，或为 -1（即数组中没有找到所求元素）：

244

```
/*-------------------------------------------------------------*/
/*  该函数在无序数列中查找一个数。如果查找成功，则返回该数在数列中的位置  */
/*  （第一个元素的位置是0，第二个元素的位置是1，以此类推）如果找不到该元  */
/*  素，函数返回 -1                                              */

int search1(int x[],int npts,int value)
{
   /*  声明变量  */
   int k=0, index=-1;

   /*  查找元素  */
   while (k<=npts-1 && x[k]!=value)
      k++;
   if (k != npts)
      index = k;

   /*  返回 index 值  */
   return index;
}
/*-------------------------------------------------------------*/
```

5.7.2 有序数列

现在考虑对有序数列的搜索。假设现在有一列有序数组，要从中查找值 25：

```
-7
2
14
38
52
77
105
```

只要搜索到值 38，就立刻知道 25 一定不在数列中，这是因为该数列是以升序排列的。因此，不需要像无序数列那样每次都搜索整个数列；而只需要搜索到所查找元素本应所处位

置的下一个位置即可。如果数列是以升序排列的，那么就一直查找到刚好大于所求元素的位置；如果数列是以降序排列的，那么就一直查找到刚好小于所求元素的位置。下面的函数是对一个有序数列进行顺序查找。函数返回值或为所求元素在有序数列中的位置，或为 -1（查找失败）：

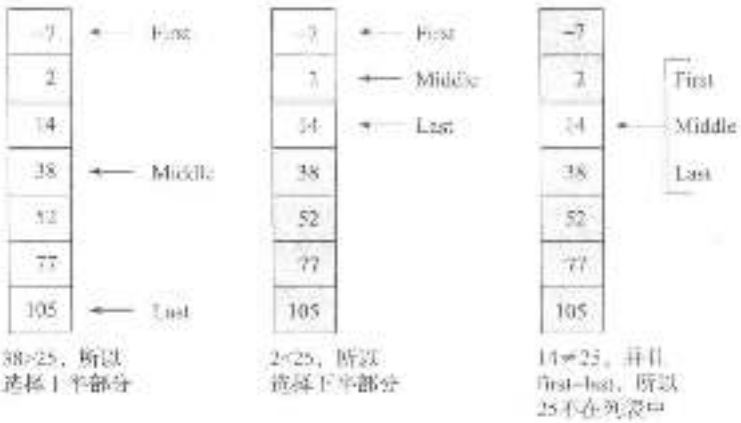
```
/*-----------------------------------------------------------------*/
/*  该函数在有序（升序）数列中查找一个数，如果找到该元素，则返回其在数列中  */
/*  的位置（第一个元素的位置是 0，第二个元素的位置是 1，以此类推）。如果没  */
/*  有找到该元素，则函数返回 -1                                      */

int search2(int x[],int npts,int value)
{
   /*  声明变量  */
   int k=0, index= 1;

   /*  查找元素  */
   while (k<=npts-1 && x[k]<value)
      k++;
   if (k <= npts-1)
      if (x[k] == value)
         index = k;

   /*  返回 index 值  */
   return index;
}
/*-----------------------------------------------------------------*/
```

[245]

关于搜索有序数列还有一个非常流行且高效的算法——二分搜索（又叫折半查找，binary search）。在二分搜索中，首先检查数组中间的元素，以确定要查找的元素是在数组的前半区还是后半区。如果所求元素在前半区，那么继续检查该半区的中间值，以确定要查找的元素是在第一个四分之一区还是在第二个四分之一区。这个过程一直持续，将数列不断划分成越来越小的部分，直到找到所求的元素，或者找到所求元素本应该所处的位置为止。因为这个方法就是不断将搜索范围一分为二，所以称为二分搜索。

现在用有序数列（-7，2，14，38，52，77，105）结合下文的图示来说明二分搜索算法的实现原理。假设现在要查找值 25。使用变量 first 来存储数组第一个元素的下标，用变量 last 存储最后一个元素的下标。然后将 first 和 last 相加，再除以 2，就得到数组中间位置的下标值（这个过程使用的是整数除法）。因为该数组有 7 个元素，所以 first 的值应该是 0，last 的值是 6；而 middle 的值经计算为 3。于是，将下标为 3 的元素与所查找的值相比较。由于 38 大于 25，就可以把搜索范围缩小至数组的前半部分。此时变量 first 仍为 0，同时将变量 last 的值更改为中间位置的前一个下标，也就是 2。现在继续将这部分数组折半，并计算中间点，也就是（0+2）/2=1。而下标为 1 的元素值为 2，小于 25，所以可以将搜索范围继续缩小至这部分数列的后半部分，也就是整个数组的第二个四分之一部分。现在 first 的值应该是中间点后面的第一个下标值，也就是 2，与此同时 last 的值也是 2。当 first 和 last 值相等时，就可以完全确定所求值应处在什么位置了。也就是说，此时要么恰好找到这个数，要么确定此数不在数列中。在这个例子中，下标为 2 的元素值为 14，所以值 25 并不在此数列中。然而，当数组的元素个数为偶数时，如果所求的数不在数列中，就很可能会出现 first 所在的下标要大于 last 的情况。

[246]



现在用函数来实现二分搜索算法。

```
/*-----------------------------------------------------------------*/
/*  本函数用二分搜索算法在一个有序(升序)数列中搜索一个值。如果查找成功，  */
/*  则函数返回这个值在数列中的序号(0代表第一个位置，1代表第二个位置，以  */
/*  此类推)。如果查找失败，则函数返回-1                              */

int search3(double x[],int npts,int value)
{
   /*  声明变量  */
   int done=0, top=0, bottom, mid;
   double index=-1;

   /*  在序列中搜索value值  */
   bottom = npts-1;
   while (top<=bottom && done==0)
   {
      /*  确定中间值  */
      mid = (top + bottom)/2;

      /*  检查中间值  */
      if (x[mid] == value)
         done = 1;
      else
         /*  value位于上半部分还是下半部分  */
         if (x[mid] > value)
            bottom = mid - 1;
         else
            top = mid + 1;
   }

   /*  确定value的序号  */
   if (done == 1)
      index = mid;

   /*  返回value的序号  */
   return index;
}
/*-----------------------------------------------------------------*/
```

247

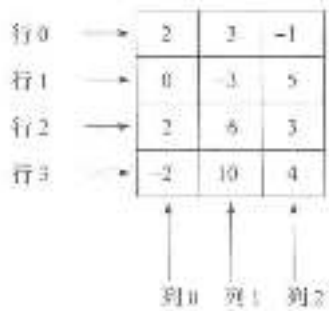
修改

1. 修改任意一个序列搜索函数来搜索字符数组，并编写一个驱动程序来测试函数。
2. 修改有序数列的搜索函数，返回给定值在有序数列中出现的次数。

3. 修改二分搜索函数，将原来的升序数列搜索改为降序数列搜索。同时编写一个驱动程序来测试函数。

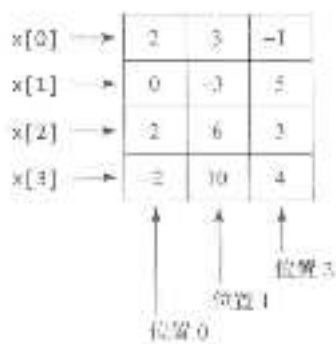
5.8 二维数组

当一组数据被形象化为一行或一列时，便很容易用一维数组表示。但是在很多场景中，数据需要利用网格或表格来表示，需要用行和列来定位描述数据组中的一个数据。下图就展示了一个4行3列的数组：



在C语言中，像这种网格化的数据表一般用二维数组（two-dimensional array）来表示。二维数组中的每个元素都是通过具有双下标的标识符来引用——一个行下标和一个列下标。行和列的下标值都从0开始，并且每个下标都包含在一组方括号内。因此，假设先前数组的标识符为x，则x[2][1]所在位置的值为6。在数组引用中可能出现的常见错误包括使用圆括号代替方括号，比如x(2)(3)；或者只用一组括号来包含下标，如x[2,3]或x(2,3)。 248

除此之外，还可以将数据网格或数据表整体看作一维数组，而其中的每个元素也都是一个数组。因此，前面的图表就可以看成是一个具有四个元素的一维数组，其中每个元素又分别是具有三个元素的一维数组。



通过这种表示方法，符号x[2][1]就可以理解为是一维数组x[2]内的参考位置[1]。因此，x[2][1]的值为6。但通常情况下，相较于数组的数组，一般还是更倾向于将二维数组按照具有行和列的网格数据来理解。

数组中的值必须具有相同的数据类型；数组中不能出现整数列和浮点数列交替混合；等等。

5.8.1 定义和初始化

要定义一个二维数组，需要在声明语句中指定行数和列数。首先要说明行数。同时，行

数和列数都包含在方括号内，如下列语句所示：

```
int x[4][3];
```

二维数组同样可以在声明语句中完成初始化。数组中的值通过一个用逗号分隔的序列来指定，并且每一行都包含在一组花括号内。最后还需要用一组花括号将整个数组括住，如下列语句所示：

```
int x[4][3]={{2,3,-1},{0,-3,5},{2,6,3},{-2,10,4}};
```

如果初始化序列的长度小于数组长度，则数组其余的值都被初始化为0。如果数组的第一个下标为空，同时还指定了一个初始化序列，则数组的大小由初始化序列的长度决定。因此，数组x也可以由下列语句定义：

249

```
int x[][3]={{2,3,-1},{0,-3,5},{2,6,3},{-2,10,4}};
```

同时，数组的初始化还可以通过程序语句来实现。对于二维数组，通常采用双层嵌套的for循环语句对数组进行初始化，并且一般使用i和j来表示下标。要定义并初始化一个二维数组，使得每个行元素都包含对应的行号，则可以通过下列语句来实现：

```
/*  声明变量  */
int i, j, t[5][4];
...
/*  初始化数组  */
for (i=0; i<=4; i++)
   for (j=0; j<=3; j++)
      t[i][j] = i;
```

上述语句被执行后，数组t中的值如下所示：

0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

二维数组也可以通过从数据文件中读入数据来进行初始化。在下面的语句中，假定数据文件中包含50个温度值，现在需要将其读取并存储在数组中。符号常量NROWS和NCOLS分别用来表示数组的行数和列数。可以将数组的行数和列数定义为符号常量，这样就很容易修改数组的大小；否则，改变数组大小就需要同时修改程序中的多条语句。相关语句如下所示：

```
#define NROWS 10
#define NCOLS 5
#define FILENAME "engine1.txt"
...
/*  声明变量  */
int i, j;
double temps[NROWS][NCOLS];
file *sensor;
...
/*  打开文件，将数据读入数组  */
sensor = fopen(FILENAME,"r");
```

```
for (i=0; i<=NROWS-1; i++)
   for (j=0; j<=NCOLS-1; j++)
      fscanf(sensor,"%lf",&temps[i][j]);
```

[250]

练习

写出下列每组语句所定义的数组内容。对没有被初始化的元素用问号标出。

```
1.int d[3][1]={{1},{4},{6}};
2.int g[6][2]={{3,2},{ 2,3}};
3.float h[4][4]={{0,0}};
4.int k, p[3][3]={{0,0,0}};
  ...
  for (k=0; k<=2; k++)
     p[k][k] = 1;
5.int i, j, g[5][5];
  ...
  for (i=0; i<=4; i++)
     for (j=0; j<=4; j++)
        g[i][j] = i + j;
6.int i, j, g[5][5];
  ...
  for (i=0; i<=4; i++)
     for (j=0; j<=4; j++)
        g[i][j] = pow(-1,j);
```

5.8.2 计算和输出

当引用二维数组元素进行计算和输出时，通常必须要指定两个下标值。为了说明问题，下面给出一个程序，首先读取一份数据文件，文件中包含一家电厂在8周内的输出功率。数据文件的每一行包含7个数值，分别表示一周的日输出功率。所有数据都存储在一个二维数组中。程序最后打印一份报告，分别列出这8周时间内，每周第一天的平均输出功率，第二天的平均输出功率，以此类推。现将该程序展示如下：

```
/*-----------------------------------------------------------*/
/*  程序chapter5_6                                           */
/*                                                           */
/*  本程序计算8周至天的平均值                                 */

#include <stdio.h>
#define NROWS 8
#define NCOLS 7
#define FILENAME "power1.txt"

int main(void)
{
   /*  声明变量  */
   int i, j;
   int power[NROWS][NCOLS], col_sum;
   FILE *file_in;
   /*  从数据文件中读取信息  */
   file_in = fopen(FILENAME,"r");
   if (file_in == NULL)
      printf("Error opening input file. \n");
   else
```

[251]

```
    {
       for (i=0; i<=NROWS-1; i++)
          for (j=0; j<=NCOLS-1; j++)
             fscanf(file_in,"%d",&power[i][j]);

       /*  计算和输出日均值  */
       for (j=0; j<=NCOLS-1; j++)
       {
          col_sum = 0;
          for (i=0; i<=NROWS-1; i++)
             col_sum += power[i][j];
          printf("Day %d: Average = %.2f \n",
                 j+1,(double)col_sum/NROWS)
       }
       /*  关闭文件  */
       fclose(file_in);
    }
    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
```

计算一周中每日的均值，就是先把表格中每列的值加起来，然后用总和除以总行数（也就是总的周数）。而列数是用来计算天数。本程序的一个输出样例如下所示：

```
Day 1: Average = 251.75
Day 2: Average = 191.50
Day 3: Average = 278.38
Day 4: Average = 188.63
Day 5: Average = 273.13
Day 6: Average = 321.38
Day 7: Average = 282.50
```

将二维数组的信息写入数据文件同一维数组的情况较为相似。在这两种情况下，都要使用换行符来指定何时在数据文件中开始写入新的一行。现在要将一组距离测量值写入数据文件 dist1.txt，其中每行写入 5 个测量值，可以通过下面的语句实现：

```
/*  声明变量  */
int i, j;
double dist[20][5];

FILE *file_out;
...
/*  将信息从数组写入文件  */
file_out = fopen("dist1.txt","w")
for (i=0, i<=19; i++)
{
   for (j=0; j<=4; j++)
      fprintf(file_out,"%f ",dist[i][j]);
   fprintf(file_out,"\n");
}
```

252

为了将数据值用空格分开，fprintf 语句中转换运算符后面的空格是必需的。

练习

假设数组 g 的声明语句如下所示：

```
int i, j, g[3][3]={{0,0,0},{1,1,1},{2,2,2}};
```

给出下列每组语句执行后 sum 的值。

```
1. sum = 0;
   for (i=0; i<=2; i++)
      for (j=0; j<=2; j++)
         sum += g[i][j];
2. sum = 1;
   for (i=1; i<=2; i++)
      for (j=0; j<=1; j++)
         sum *= g[i][j];
3. sum = 0;
   for (j=0; j<=2; j++)
      sum -= g[2][j];
4. sum = 0;
   for (i=0; i<=2; i++)
      sum += g[i][i];
```

5.8.3 函数参数

当数组被用作函数参数时，对它的引用是传址调用，而非传值调用。在 5.1 节中讨论了一维数组，了解到数组在函数中的引用是直接获取到原始数组，而不是对数组的副本进行操作。因此，必须要注意避免在不经意间改变原始数组的值。传址调用提供了另一种数据传递的方法。除了可以在函数调用结束时传递一个返回值之外，还能通过在函数中改变数组的值传送数据。 253

当使用一维数组作为函数参数时，只需要将数组名称放进参数中，就可以指定数组地址。当使用二维数组作为函数参数时，除了数组名称以外，还需要给出数组大小。一般而言，函数声明和原型语句中应该给出关于二维数组大小的完整信息。下面通过一个程序来说明二维数组作为函数参数时的用法。假设该程序要计算一个 4 行 4 列二维数组的所有元素之和。整个计算过程需要两层嵌套循环。用一个函数来实现这个计算过程，程序的可读性会更好。这样一来，主程序中只需一条语句就能引用该函数，如下列语句所示：

```
/*  声明变量和函数原型  */
int a[4][4];
int sum(int x[4][4]);
...
/*  利用函数计算数组总和  */
printf("Array sum = %i \n",sum(a));
```

如果在程序的其他地方同样需要计算数组之和，则该函数会显得更加便利和高效。当然，对于大小相同的其他数组，同样可以利用此函数来计算数组总和，如下列语句所示：

```
/*  声明变量和函数原型  */
int a[4][4], b[4][4];
int sum(int x[4][4]);
...
/*  利用函数计算数组总和  */
printf("Sum of a = %i \n",sum(a));
printf("Sum of b = %i \n",sum(b));
```

现在来分析刚才使用过的函数的代码实现：

```
/*-------------------------------------------------------------------*/
/*  该函数返回一个4*4列数组的元素之和                                */

int sum(int x[4][4])
```

```
{
   /*  声明和初始化变量  */
   int i, j, total=0;

   /*  计算数组总和  */
   for i=0; i<=3; i++)
      for (j=0; j<=3; j++)
         total += x[i][j];

   /* 返回数组总和  */
   return total;
/*-----------------------------------------------------------------*/
```

在本例中，函数定义和函数原型包含了全部行数和列数。而C语言允许省略第一个下标值，因此，可以使用下列语句来替换原来的函数定义和原型：

[234]

```
/*  声明变量和函数原型  */
int sum(int x[][4]);
```

为了让定义更加清晰，通常更倾向于在函数的形参列表和函数原型中将数组的行数和列数完整列出。

最后这个例子中，我们来设计一个函数计算数组中部分元素之和。假设要求和的元素存在于数组左上角的子数组中。则函数参数应该包括原始数组、子数组的行数和列数。函数原型如下所示：

```
/*  函数原型  */
int partial_sum(int x[4][4], int r, int c);
```

因此，如果要计算下列数组a中阴影部分所示的元素之和，需要调用函数partial_sum(a,2,3)。

2	3	-1	9
0	-3	5	7
2	6	3	2
-2	10	4	6

随后被调用的函数应该计算左上角2行3列的子数组元素之和。该函数最后返回一个值6。函数代码如下所示：

```
/*-----------------------------------------------------------------*/
/*  该函数返回4行4列数组a中的一个子数组的元素之和                    */

int partial_sum(int x[4][4],int r,int c)
{
   /*  声明和初始化变量  */
   int i, j, total=0;

   /*  计算子数组的元素之和  */
   for i=0; i<=r-1; i++)
      for (j=0; j<=c-1; j++)
         total += x[i][j];

   /* 返回子数组的元素之和  */
   return total;
/*-----------------------------------------------------------------*/
```

在使用一维数组时，在函数的数组参数中指定数组的大小是没有作用的，而需要用一个专门的参数用于标识数组的长度。因此，这样的函数可以处理各种大小的数组。例如，在4.2节中设计了一个函数用于计算一维数组元素的平均值。如果要计算包含10个元素的数组a的元素平均值，则函数引用为mean(a,10)。如果要计算包含50个元素的数组y的元素平均值，则函数引用为mean(y,50)。如果要编写函数来处理大小可变的二维数组，就必须使用指针作为函数参数。具体的实现方法将会在第6章详细介绍。 255

练习

假设一个主函数包含下列语句：

```
int a[4][4]={{2,3,-1,9},{0,-3,5,7},
             {2,6,3,2},{-2,10,4,6}};
```

本小节设计了函数partial_sum，手动计算出下列函数引用的值：

1. partial_sum(a,1,4);
2. partial_sum(a,1,1);
3. partial_sum(a,4,2);
4. partial_sum(a,2,4);

后面的4个小节中包含了使用二维数组的例子。5.9节介绍了有关地形导航的应用；5.10节使用二维数组来表示矩阵；5.11节和5.12节讨论并设计了联立方程组的求解方法，其中使用二维数组存储方程组的系数。

5.9 解决应用问题：地形导航

地形导航是无人驾驶飞行器（UAV，简称无人机）设计的关键部分。机器人或汽车是地面交通工具，而无人机或者飞机是在空中飞行的交通工具，它们的导航方法存在很大的不同。一般无人机系统都会搭载机载计算机，其中存储了操作区域的地形信息。因为无人机时刻都能确定自己所在的位置（通常是利用全球定位系统接收器进行定位），因此它可以选择最佳路径到达指定地点。如果目的地发生改变，无人机还可以依照内部地图重新计算并规划新的路径。

引导无人机执行各种操作的机载软件必须要在各种各样的地面形态和拓扑结构中完成测试。计算机数据库中的存储通常将陆地划分成网格，逐块存储各类信息，其中包括海拔信息。如果要测量一块陆地网格对于实施地形导航的“难易程度”，常用的一种方法就是要确定该网格中山峰的个数，其中，山峰是指四周海拔都低于中心的位置点。这个问题实际上就变成了要确定网格位置grid[m][n]是否是一个山峰。假设下图所示的4个位置都是网格位置grid[m][n]的相邻点。

	grid[m-1][n]	
grid[m][n-1]	grid[m][n]	grid[m][n+1]
	grid[m+1][n]	

256

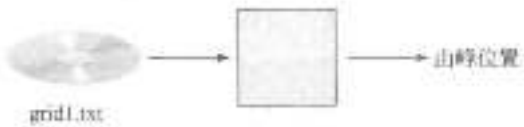
编写程序，从数据文件grid1.txt中读取海拔数据，并输出山峰的数量和位置。假设数据文件的第一行包含了该网格信息的行数和列数。随后按照行序依次列出了网格位置的海拔高度。该网格的最大尺寸为25行25列。

1. 问题陈述

确定并输出一块网格地区中山峰的数量及位置。

2. 输入/输出描述

下面的I/O图显示，程序输入是包含海拔数据的文件，输出是山峰位置的列表。



3. 手动演算示例

假设下面的数值代表网格的海拔数据，该网格边缘纵向有6个点，横向有7个点（山峰已被下划线标出）：

5039	5127	5238	5259	5248	5310	5299
5150	5392	5410	5401	5320	5820	5321
5290	5560	5490	5421	5530	5831	5210
5110	5429	5430	5411	5459	5630	5319
4920	5129	4921	<u>5821</u>	4722	4921	5129
5023	5129	4822	4872	4794	4862	4245

为了标定山峰的位置，我们首先需要设计一个数据的定位方法。由于是采用C语言来实现，所以选用二维数组来存储数据，相应的用下标和数据的位置建立关联，地图上左上角的位置是[0][0]，随着向页面下方移动，行号加1，向右移动列号加1。这样一来，出现山峰的位置分别为[2][1]、[2][5]和[4][3]。

在确定山峰的过程中，主要是将要判断的中心点同它的4个邻近点的海拔高度相比较。如果4个邻近点的海拔高度都小于该点，则判定该点是一个山峰。另外要注意的是，数组或网格边缘的点不应该作为中心点来参与判定，因为这些点四周的海拔信息不完整。

4. 算法设计

257 首先设计分解提纲，将解决方案分解成一组可以顺序执行的步骤。

分解提纲

1）将地形数据读入数组。

2）确定并输出山峰位置。

在步骤1）中，程序读取数据文件，并将信息存储在二维数组中。

在步骤2）中，利用一个循环体来判定所有潜在的山峰位置，如果确定为山峰，则输出该点的位置。程序明显不需要其他函数，因此只对主函数来设计提炼后的伪代码：

```
[提炼后的伪代码]
主函数：从数据文件中读取行数 nrows 和列数 ncols
        将地形数据读入 elevation 数组
        i=1;
        while i ≤ nrows-2
          j=1;
          while j ≤ ncols-2
```

```
        if elevation[i][j] 大于它的四个邻近点
          输出该山峰的位置
        j++;
      i++;
```

伪代码中的步骤足够详细，可将其直接转化为C程序：

```
/*-----------------------------------------------------------------*/
/*  程序 chapter5_7                                                */
/*                                                                 */
/*  本程序根据海拔数据确定网格中山峰的位置                          */

#include <stdio.h>
#define N 25
#define FILENAME "grid1.txt"

int main(void)
{
   /*  声明变量  */
   int nrows, ncols, i, j;
   double elevation[N][N];
   FILE *grid;

   /*  从数据文件读取信息  */
   grid = fopen(FILENAME,"r");
   if (grid == NULL)
      printf("Error opening input file\n");
   else
   {
      fscanf(grid,"%d %d",&nrows,&ncols);
      for (i=0; i<=nrows-1; i++)
         for (j=0; j<=ncols-1; j++)
            fscanf(grid,"%lf",&elevation[i][j]);

      /*  确定并输出山峰位置  */
      printf("Top left point defined as row 0, column 0 \n");
      for (i=1; i<=nrows-2; i++;)
         for (j=1; j<=ncols-2; j++)
            if ((elevation[i-1][j]<elevation[i][j]) &&
                (elevation[i+1][j]<elevation[i][j]) &&
                (elevation[i][j-1]<elevation[i][j]) &&
                (elevation[i][j+1]<elevation[i][j]))
               printf("Peak at row: %d column: %d \n",i,j)

      /*  关闭文件  */
      fclose(grid);
   }

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

258

5. 测试

使用手动演算示例中的数据文件作为程序输入，得到如下输出结果：

```
Top left point defined as row 0, column 0
Peak at row: 2 column: 1
Peak at row: 2 column: 5
Peak at row: 4 column: 3
```

不要忘了，在数据文件中的第一行比较特殊，它用来指定海拔数据的行数和列数。

修改

修改程序 chapter5_7，确定关于网格海拔数据的相关信息：

1. 打印出网格中山峰的总数目。
2. 打印出网格中山谷的位置。假设山谷就是海拔低于周围四个邻近位置的中心点。
3. 找到并输出海拔数据中最高点和最低点的位置及海拔。
4. 假设垂直方向和水平方向的两个点之间的距离为 100 英尺，从左下角开始，以英尺为单位，绘出所有山峰的位置。
5. 利用全部 8 个邻近点（而不仅仅是 4 个邻近点）来确定网格中的山峰。

259

*5.10 矩阵和向量

矩阵（matrix）就是排列在具有行和列的矩形网格中的一组数据。一个 4 行 3 列的矩阵的大小被指定为 4×3，如下所示：

$$A = \begin{vmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & 1 \end{vmatrix}$$

要注意，矩阵内的值都要用一个大的方括号括起来。仅有一行的矩阵被称为行向量（row vector），仅有一列的矩阵被称为列向量（column vector）。而术语向量（vector）本身是没有行列之分的。

在数学符号中，矩阵的名字通常用大写的黑体字母表示。同时，使用行号和列号来引用单个矩阵元素，并且行、列号都从 1 开始。在正式的数学符号中，大写字母（也就是矩阵名称）表示整个矩阵，而带下标的小写字母就表示一个特定元素。因此，在矩阵 A 中，$a_{3,1}$ 代表的值是 −2。如果一个矩阵的行数和列数相等，则称其为方阵（square matrix）。

二维数组可以用来存储矩阵，但是在将矩阵中变量的角标转换为 C 语言中的数组下标时必须要多加注意，因为两者的标号用法不同。矩阵符号假设行号和列号从 1 开始，但 C 语句假设数组的行号和列号从 0 开始。另外，在处理向量时，虽然可以将向量存储为只有一行或一列的二维数组，但一般还是习惯用一维数组来存储向量。如此一来，对于行向量和列向量一般就不作区分了。

在解决工程问题时常常会用到矩阵运算，所以下面将给出矩阵和向量的常见运算。同时，对这些运算分别用 C 语言进行实现。另外，在本章结尾还会给出关于其他矩阵运算的编程习题。

5.10.1 点积

点积（dot product）是两个相同类型的向量经过计算得到的数值。它的值等于两个向量对应位置元素的乘积之和，如下面的加和公式所示，假设向量 A 和 B 中各有 n 个元素：

$$点积 = A \cdot B = \sum_{i=1}^{n} a_i b_i$$

下面举个简单的例子，假设 A 和 B 是如下两个向量：

$$A = |4 \quad -1 \quad 3| \qquad B = |-2 \quad 5 \quad 2|$$

则两向量的点积为

$$
\begin{aligned}
A \cdot B &= 4 \cdot (-2) + (-1) \cdot 5 + 3 \cdot 2 \\
&= (-8) + (-5) + 6 \\
&= -7
\end{aligned}
$$

点积也被称为内积（inner product）。

在 C 语言中，可以通过下面的函数来计算两个一维向量的点积： [260]

```
/*-----------------------------------------------------------------*/
/*  该函数返回两个向量的点积                                       */

double dot_product(double a[],double b[],int n)
{
   /*  声明变量并初始化  */
   int k;
   double sum=0;

   /*  计算点积  */
   for (k=0; k<=n-1; k++)
      sum += a[k]*b[k];

   /*  返回点积  */
   return sum;
}
/*-----------------------------------------------------------------*/
```

需要注意的是，在公式中使用的从 1 到 n 的下标在 C 程序中变为了从 0 到 $n-1$。

5.10.2 行列式

矩阵的行列式（determinant）就是矩阵元素通过特定的计算后得到的数值。行列式在工程领域中有着多种应用，包括计算逆以及联立方程组的求解。对于一个 2×2 的矩阵 A，其行列式的定义如下：

$$A\text{的行列式} = |A| = a_{1,1}a_{2,2} - a_{2,1}a_{2,1}$$

因此，对于下面的矩阵，其行列式就等于 8：

$$A = \begin{vmatrix} 1 & 3 \\ -1 & 5 \end{vmatrix}$$

而对于一个 3×3 的矩阵 A，其行列式可以通过如下等式来计算：

$$
\begin{aligned}
|A| = {} & a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} - a_{3,1}a_{2,2}a_{1,3} \\
& - a_{3,2}a_{2,3}a_{1,1} - a_{3,3}a_{2,1}a_{1,2}
\end{aligned}
$$

如果 A 的矩阵定义为

$$A = \begin{vmatrix} 1 & 3 & 0 \\ -1 & 5 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

那么 $|A|$ 就等于 $5 - 6 + 0 - 0 - 4 - (-3)$，也就是 10。

当矩阵拥有更多的行和列时（多于 3 行 3 列），要得到对应行列式就需要更为复杂的计算过程。在本章末尾的问题中将会对此展开讨论。 [261]

5.10.3 转置

一个矩阵的转置（transpose）就是将原来的行变为列，从而得到一个新矩阵。通常在矩阵名称后加一个上标T来表示转置。例如，下面就是一个矩阵和它的转置：

$$B=\begin{vmatrix} 2 & 5 & 1 \\ 7 & 3 & 8 \\ 4 & 5 & 21 \\ 16 & 13 & 0 \end{vmatrix}, \qquad B^{T}=\begin{vmatrix} 2 & 7 & 4 & 16 \\ 5 & 3 & 5 & 13 \\ 1 & 8 & 21 & 0 \end{vmatrix}$$

如果要观察其中元素的变化，可以看到位置（3，1）的值现在移动到了位置（1，3），位置（4，2）的值移动到了位置（2，4）。实际上，转置的过程就是将位置（i，j）上的元素移动到位置（j，i），也就是将行列下标互换。同时还要注意的是，转置矩阵的大小一般是不同于原矩阵的（除非原矩阵是一个方阵）。

现在要设计一个函数来生成转置矩阵，函数的形参需要两个二维数组，分别用来存储原矩阵和转置矩阵。为了保证函数的灵活性，假设已经定义了符号常量NROWS和NCOLS用来指定原矩阵的行数和列数。由于使用符号常量就等同于使用其中给出的数值，因此可以在数组定义和原型语句中使用NROWS和NCOLS。要注意的是，该函数并不需要返回结果，因此返回类型为void。另外，符号常量NROWS和NCOLS应该在使用该函数的程序里事先定义：

```
/*-------------------------------------------------------------*/
/*  该函数生成一个转置矩阵，NROWS和NCOLS是符号常量，需要在调用该函      */
/*  数不程序中声明                                              */

void transpose(int b[NROWS][NCOLS],int bt[NCOLS][NROWS])
{
   /*  声明变量  */
   int i, j;

   /*  将元素转换为转置矩阵  */
   for (i=0; i<=NROWS-1; i++)
      for (j=0; j<=NCOLS-1; j++)
         bt[j][i] = b[i][j];

   /*  无返回值  */
   return;
}
/*-------------------------------------------------------------*/
```

262

5.10.4 矩阵加减法

两个矩阵的加法（或减法）运算就是将矩阵中对应位置的元素相加（或相减）。因此，相加（或相减）的矩阵必须大小相同；运算得出的结果当然也是相同大小。现在考虑下面的矩阵：

$$A=\begin{vmatrix} 2 & 5 & 1 \\ 0 & 3 & -1 \end{vmatrix}, \qquad B=\begin{vmatrix} 1 & 0 & 2 \\ -1 & 4 & -2 \end{vmatrix}$$

几种加减法运算如下：

$$A+B=\begin{bmatrix}3 & 5 & 3\\ -1 & 7 & -3\end{bmatrix},\quad A-B=\begin{bmatrix}1 & 5 & -1\\ 1 & -1 & 1\end{bmatrix},$$

$$B-A=\begin{bmatrix}-1 & -5 & 1\\ -1 & 1 & -1\end{bmatrix}$$

5.10.5 矩阵乘法

同矩阵加减法不同，矩阵乘法（matrix multiplication）并不是简单地将两个矩阵的对应元素相乘。乘积矩阵 $\boldsymbol{C}$ 中位置 $c_{i,j}$ 上的值等于被乘数矩阵的第 i 行与乘数矩阵的第 j 列之间的点积，如下列等式所示：

$$c_{i,j}=\sum_{k=1}^{n}a_{i,k}b_{k,j}$$

由于点积运算要求向量具有相同数量的元素，也就是说，被乘数矩阵（$\boldsymbol{A}$）每行的元素个数应该同乘数矩阵（$\boldsymbol{B}$）每列的元素个数相同。因此，如果矩阵 $\boldsymbol{A}$ 和矩阵 $\boldsymbol{B}$ 都拥有 5 行和 5 列，那么矩阵乘积也是 5 行 5 列。而且，对于矩阵相乘，可以计算出 $\boldsymbol{AB}$ 和 $\boldsymbol{BA}$，但通常两者并不相等。

如果矩阵 $\boldsymbol{A}$ 有 2 行 3 列，而矩阵 $\boldsymbol{B}$ 有 3 行 3 列，那么乘积 $\boldsymbol{AB}$ 将会有 2 行 3 列。为了说明这个过程，现在考虑下面的矩阵：

$$\boldsymbol{A}=\begin{bmatrix}2 & 5 & 1\\ 0 & 3 & -1\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}1 & 0 & 2\\ -1 & 4 & -2\\ 5 & 2 & 1\end{bmatrix}$$

乘积 $\boldsymbol{C}=\boldsymbol{AB}$ 的第一个元素等于

$$\begin{aligned}C_{1,1}&=\sum_{k=1}^{3}a_{1k}b_{k1}\\&=a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,3}b_{3,1}\\&=2\times1+5\times(-1)+1\times5\\&=2\end{aligned}$$

263

同上面的过程类似，矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 乘积中余下的元素计算之后，得到结果如下：

$$\boldsymbol{AB}=\boldsymbol{C}=\begin{bmatrix}2 & 22 & -5\\ -8 & 10 & -7\end{bmatrix}$$

在这个例子中，计算的是矩阵乘积 $\boldsymbol{AB}$，但是无法计算 $\boldsymbol{B}$，因为矩阵 $\boldsymbol{B}$ 每行的元素个数同矩阵 $\boldsymbol{A}$ 每列的元素个数不相等。

有一种简单的方法来确认矩阵乘积是否存在，就是将两个矩阵的大小并排写出。如果内侧的两个数字相同，则乘积存在；同时外侧的两个数字决定了乘积矩阵的大小。为了说明问题，以先前的矩阵乘法为例，矩阵 $\boldsymbol{A}$ 的大小为 2×3，矩阵 $\boldsymbol{B}$ 的大小为 3×3。计算乘积 $\boldsymbol{AB}$ 之前，先将两矩阵大小并排写出：

$$2\times3\qquad 3\times3$$

内侧的两个数字都是3，所以乘积 AB 存在，并且它的大小由外侧两个数字确定，即 2×3。如果要计算乘积 BA，那么再次将两矩阵大小并排写出：

$$3\times3 \quad 2\times3$$

内侧的两个数字不相等，所以乘积 BA 并不存在。

下面给出一个函数来计算矩阵乘积 $C=AB$。在该函数中，每个数组的大小都是 $N\times N$，其中 N 是一个符号常量：

```
/*-----------------------------------------------------------------*/
/*  该函数通过乘积和来计算两个N×N矩阵的乘积。N是一个符号常量，需要  */
/*  在调用该函数的主程序中进行定义                                  */

void matrix_mult(int a[N][N],int b[N][N],int c[N][N])
{
   /*  声明变量  */
   int i, j, k;

   /*  计算乘积和  */
   for (i=0; i<=N-1; i++)
      for (j=0; j<=N-1; j++)
      {
         c[i][j] = 0;
         for (k=0; k<=N-1; k++)
            c[i][j] += a[i][k]*b[k][j];
      }

   /*  无返回值  */
   return;
}
/*-----------------------------------------------------------------*/
```

264

练习

于算下列表达式的结果。然后编写程序，使用本节设计的函数来验证于算的结果。使用下列矩阵和向量参与运算：

$$A=\begin{bmatrix}2 & 1\\ 0 & -1\\ 3 & 0\end{bmatrix},\quad B=\begin{bmatrix}-2 & 2\\ -1 & 5\end{bmatrix},\quad C=\begin{bmatrix}3 & 2\\ -1 & -2\\ 0 & 2\end{bmatrix},\quad D=(1\quad 2)$$

1. $D\cdot D$　　2. $|B|$　　3. $C^{\mathrm{T}}+A^{\mathrm{T}}$

4. DB　　5. $B(C^{\mathrm{T}})$　　6. $(CB)D^{\mathrm{T}}$

在本章末尾的问题中使用了本节讨论的矩阵运算操作；同时也给出了其他一些矩阵运算的定义。

*5.11 数值方法：联立方程组求解

在面对工程问题时，常常需要求取联立方程组的解。求解方程组有多种方法，每种方法都有其优缺点。在本节，将会使用高斯消元的方法来求解联立线性方程组（simultaneous linear equation）。之所以称作线性方程是因为方程中只包含线性关系的项（一次幂），诸如 x、

y、z。在给出高斯消元的实现细节之前，首先介绍如何使用图像法解方程组。

5.11.1 图像阐释

具有两个变量的方程。例如 $2x-y=3$，定义了一条直线。这样的方程通常写作 $y=mx+b$，其中 m 表示直线的斜率，b 表示直线在 y 轴的截距。因此 $2x-y=3$ 便可以写作 $y=2x-3$。如果有两个线性方程，那么可以表示为三种可能的情况：两条直线相交于一点、两条直线互相平行或者两个方程定义了同一条直线。这三种情况都展示在图 5-3 中。代表两条相交直线的方程组很容易辨别，因为它们的斜率一定不同，就像 $y=2x-3$ 和 $y=-x+3$。而代表两条平行直线的方程组一定具有相同的斜率和不同的 y 轴截距，比如 $y=2x-3$ 和 $y=2x+1$。最后，代表同一条直线的方程组则一定具有相同的斜率和截距，比如 $y=2x-3$ 和 $3y=6x-9$。 265

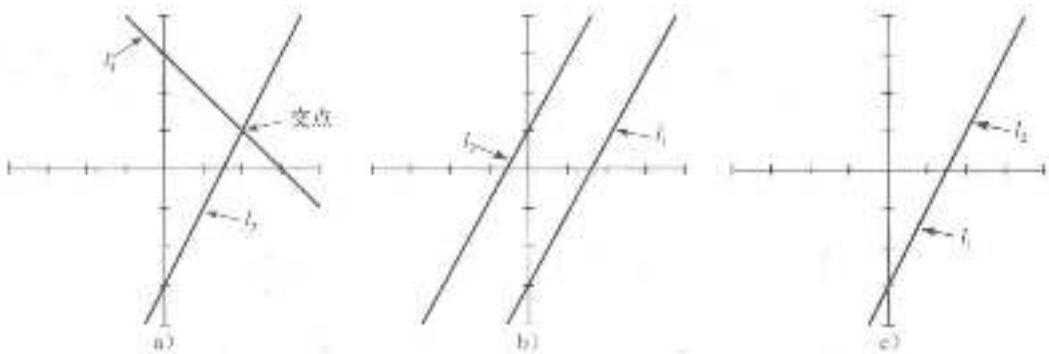


图 5-3 两条直线的位置关系

如果一个线性方程包含三个变量 x、y 和 z，则该方程表示三维空间中的一个平面。如果方程组中包含两个这样的方程，则该方程组可以表示为以下三种可能的情况：两个平面相交于一条直线、两个平行平面或者同一个平面。这三种情况展示在图 5-4 中。如果一个方程组包含三个这样的方程，假定这三个方程分别定义了三个不同平面，则该方程组可以表示三个平面相交于一点，或相交于一个平面，也可能没有交点，或者这三个方程表示同一平面。以上各种可能已经在图 5-5 中举例说明。 266

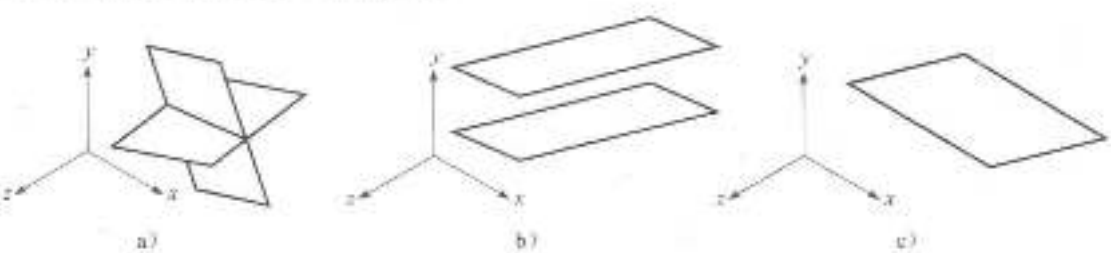


图 5-4 两个平面的位置关系

上面的思想还可以延伸到三个变量以上的情况，但是相对应的场景就难以描绘了。由超过三个变量组成的方程定义的点集叫作超平面（hyperplane）。我们研究的联立线性方程求解问题，就是一组包含 n 个未知量的线性方程，方程共有 m 个，并且每个方程定义的超平面都各不相同。如果 $m<n$，则该方程组称为欠定方程，此时方程组不存在唯一解。如果 $m=n$，

并且每个方程代表的超平面之间都不平行，则方程组存在唯一解。如果 $m>n$，则该方程组称为超定方程，此时方程组不存在唯一解。一组联立的方程也称为一个方程组（system of equation）。此外，具有唯一解的方程组叫作非奇异（nonsingular）方程组，而不具有唯一解的方程组就叫作奇异方程组。

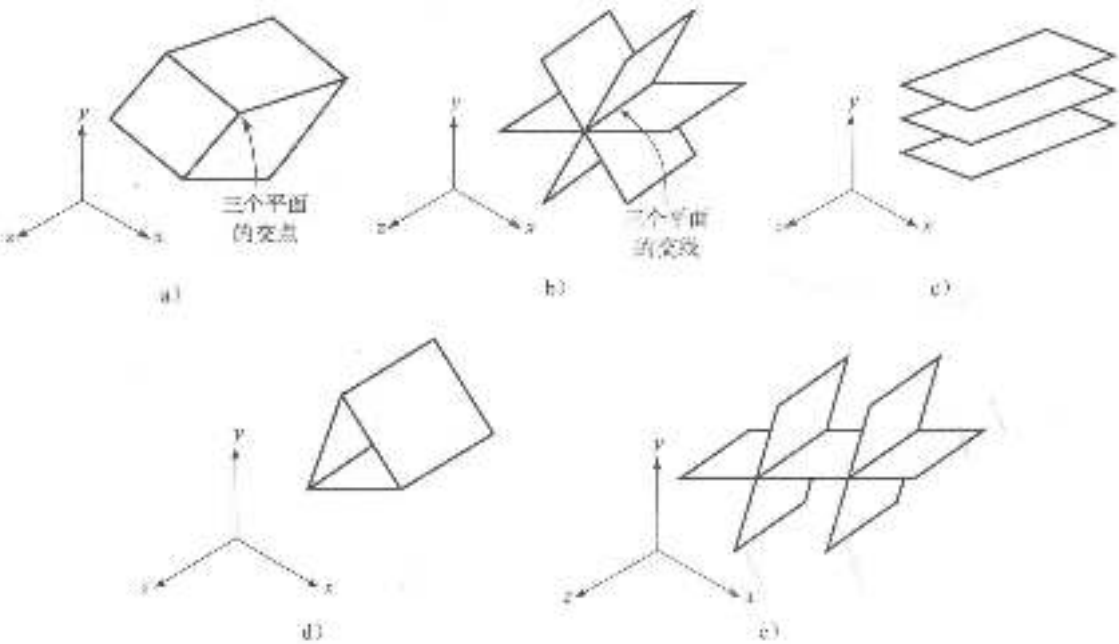


图 5-5 三个不同平面间的位置关系

这里给出一个具体示例，考虑下面的方程组：

$$
\begin{aligned}
3x+2y-z&=10\\
-x+3y+2z&=5\\
x-y-z&=-1
\end{aligned}
$$

267 ~ 268 该方程组的解是点（-2，5，-6）。可以将这些值代入方程组中加以验证。

在解决本章前面提出的问题时，并不要求了解到矩阵的实质。然而，如果你尝试去了解其本质，就会发现其实联立方程组可以被表示为矩阵相乘的形式。为了说明问题，使用下列矩阵来表示上述方程中的信息：

$$
\boldsymbol{A}=\begin{bmatrix}3&2&-1\\-1&3&2\\1&-1&-1\end{bmatrix},\quad \boldsymbol{X}=\begin{bmatrix}x\\y\\z\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}10\\5\\-1\end{bmatrix}
$$

因此，利用矩阵乘法，可以将方程组写作以下形式：

$$\boldsymbol{AX}=\boldsymbol{B}$$

将相应的矩阵代入计算，就可以发现该矩阵方程完全等价于原始方程组。

在许多工程问题中，我们一直都希望确定是否存在一种适用解决来求解联立方程组。如果适用解法存在，就期望找到该解法。在接下来的内容里，将介绍如何使用高斯消元方法来

求解联立线性方程组。 269

5.11.2 高斯消元法

在提出高斯消元（Gauss elimination）方法的一般性描述之前，首先通过一个特殊的例子来对高斯消元进行说明。使用前面给出的方程组：

$$
\begin{aligned}
3x + 2y - z &= 10 \qquad \text{（方程1）}\\
-x + 3y + 2z &= 5 \qquad \text{（方程2）}\\
x - y - z &= -1 \qquad \text{（方程3）}
\end{aligned}
$$

第一步是消元（elimination），也就是从方程 1 后面的每个方程中消除第一个变元。具体是通过对方程 1 进行适当比例的缩放，然后分别加到每个方程上来实现。找到关于第一个变元 x 的项，在方程 2 中是 $-x$。因此，如果对方程 1 乘以 1/3，然后再加到方程 2 上，就能得到一个变元 x 被消除的新方程：

$$
\begin{array}{lll}
-x - 3y + 2z = 5 & & \text{（方程2）}\\
x + \frac{2}{3}y - \frac{1}{3}z = \frac{10}{3} & & \text{（方程1乘以1/3）}\\
\hline
0x + \frac{11}{3}y + \frac{5}{3}z = \frac{25}{3} & & \text{（两方程之和）}
\end{array}
$$

第一步消元之后，方程组变为：

$$
\begin{aligned}
3x - 2y - z &= 10\\
0x + \frac{11}{3}y + \frac{5}{3}z &= \frac{25}{3}\\
x - y - z &= -1
\end{aligned}
$$

现在用同样的方法从方程 3 中消除变元 x：

$$
\begin{array}{lll}
x - y - z = -1 & & \text{（方程3）}\\
-x - \frac{2}{3}y + \frac{1}{3}z = -\frac{10}{3} & & \text{（方程3乘以1/3）}\\
\hline
0x - \frac{5}{3}y - \frac{2}{3}z = -\frac{13}{3} & & \text{（两方程之和）}
\end{array}
$$

第二步消元之后，方程组变为：

$$
\begin{aligned}
3x + 2y - z &= 10\\
0x + \frac{11}{3}y + \frac{5}{3}z &= \frac{25}{3}\\
0x - \frac{5}{3}y - \frac{2}{3}z &= -\frac{13}{3}
\end{aligned}
$$

现在除了方程 1 以外，其余所有方程中的第一个变元都已被消除。

类似的，继续消除余下方程中的第二个变元，除了方程 1 和方程 2 外，其他方程中第二个变量的系数都消为 0。对方程 2 进行适当比例的缩放，然后加到方程 3 上： 270

$$0x - \frac{5}{3}y - \frac{2}{3}z = -\frac{13}{3} \qquad \text{（方程3）}$$

$$0x + \frac{5}{3}y + \frac{25}{33}z = \frac{125}{33} \qquad \text{（方程2乘以5/11）}$$

$$0x + 0y + \frac{3}{33}z = -\frac{18}{33} \qquad \text{（所方程之和）}$$

消元之后的方程组为：

$$\begin{aligned} 3x - 2y - z &= 10 \\ 0x + \frac{11}{3}y + \frac{5}{3}z &= \frac{25}{3} \\ 0x + 0y + \frac{3}{33}z &= -\frac{18}{33} \end{aligned}$$

由于方程 3 是最后一个方程，所以该部分算法执行完毕。

现在通过回代（back substitution）来确定方程组的解。由于最后一个方程仅有一个变元，那么就可以选择一个比例因子乘以该方程，使其唯一的变量系数等于 1。因此，对方程 3 乘以 33/3（或 11），可以得到

$$0x + 0y + z = -6$$

接下来将 z 的值代入倒数第二个方程，可以得到

$$0x + \frac{11}{3}y + \frac{5}{3}(-6) = \frac{25}{3}$$

简化方程，使所有的常数项都在方程等号右侧，可以得到

$$0x + \frac{11}{3}y = \frac{55}{3}$$

此时方程 2 仅有一个变元，所以选择一个比例因子相乘，使其唯一的变量系数等于 1：

$$0x + y = 5$$

继续回代至下一个方程，也就是本例中最后一个方程：

$$3x + 2y - z = 10$$

将前面已确定的值回代至方程 1，得到

$$3x + 2(5) - (-6) = 10$$

也就是

$$3x = -6$$

因此，x 的值为 -2。

从上面的例子可以看到，高斯消元法共分为两部分——消元和回代。首先，通过对方程进行适当变换，将第 k 个方程之后的所有方程中的第 k 个变元全部消除。然后，从最后一个方程开始计算最后一个变元，接着代入倒数第二个方程中，继续计算倒数第二个变元。这个回代的过程一直持续到最终确定了所有变元值为止。此外，如果方程组中出现一个变元的所有系数都为 0 或是非常接近于 0，那么该方程组就是病态（ill-conditioned）的，或者说不

存在唯一解。

采用选主元操作可以有效地提高高斯消元法的准确性。在执行高斯消元之前，通过矩阵变换将合适的参数值移动到对角线的位置用于消元。行主元消元法需要改变行间顺序，列主元消元法需要改变列间顺序，完全主元消元则是需要对行列顺序同时进行改变。在本章末尾的习题中将会对这些变换过程进行讨论。

练习

使用高斯消元法找到这些联立线性方程组的解。

1. $2x + y = -3$
   $x + y = 3$
2. $3x + 5y + 2z = 8$
   $2x + 3y - z = 1$
   $x - 2y - 3z = -1$

271

*5.12 解决应用问题：电路分析

对于电路特性的分析常常会需要去解一组联立线性方程组。为了得到对应的方程，需要去分析电路中各节点处的电流情况来得到相应的电流方程；或者分析电路中每一个回路中的电压情况来得到相应的电压方程。例如，考虑图 5-6 中的电路。图中这三个回路中的电压情况可以通过下面的方程来表示：

$$
\begin{aligned}
-V_1 + R_1 i_1 - R_2(i_1 - i_2) &= 0 \\
R_2(i_2 - i_1) - R_3 i_2 - R_4(i_2 - i_3) &= 0 \\
R_4(i_3 - i_2) - R_5 i_3 - V_2 &= 0
\end{aligned}
$$

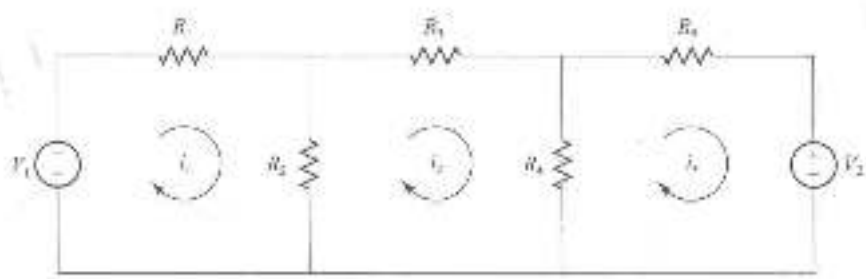


图 5-6 带有两个电源的电路

272

如果假设电阻（R_1, R_2, R_3, R_4 和 R_5）的值和电源（V_1、V_2）的值均是已知，并且回路电流（i_1, i_2 和 i_3）是未知，那么可将下列方程重新整理为如下形式：

$$
\begin{aligned}
(R_1 + R_2)i_1 - R_2 i_2 + 0i_3 &= V_1 \\
-R_2 i_1 + (R_2 + R_3 + R_4)i_2 - R_4 i_3 &= 0 \\
0i_1 - R_4 i_2 - (R_4 + R_5)i_3 &= -V_2
\end{aligned}
$$

编写程序，令用户输入这 5 个电阻及 2 个电源电压的值，然后计算出 3 个回路电流的值。

1. 问题陈述

计算图5-6中电路的三个回路电流。

2. 输入/输出描述

从下面的I/O图可以看到，程序的输入是电阻值和电压值，三个回路电流是程序的输出值。

3. 手动演算示例

通过使用电阻值和电压值，可以将这三个联立的方程整理为如下形式：

$$
\begin{aligned}
(R_1 + R_2)i_1 &- R_2 i_2 + 0i_3 = V_1 \\
-R_2 i_1 &+ (R_2 + R_3 + R_4)i_2 - R_4 i_3 = 0 \\
0i_1 &- R_4 i_2 + (R_4 + R_5)i_3 = -V_2
\end{aligned}
$$

例如，假设每个电阻值是1 Ω，两个电源都为5 V。那么这方程组化为如下形式：

$$
\begin{aligned}
2i_1 &- i_2 + 0i_3 = 5 \\
-i_1 &- 3i_2 - i_3 = 0 \\
0i_1 &- i_2 + 2i_3 = -5
\end{aligned}
$$

一旦方程组确定了，就可以按照前面章节的手动示例中的步骤求解方程。可以得出方程组的解为 $i_1 = 2.5$，$i_2 = 0$，$i_3 = -2.5$。

273

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤：

分解提纲

1）读取电阻值和电压值。

2）确定方程组中的系数。

3）执行高斯消元来确定电流值。

4）打印出电流值。

步骤1)，读取电路中必要的元件特性信息。步骤2)，使用这些信息来指定方程组的系数。步骤3)，设计具体的消元和回代步骤。为了保持main函数的精炼性与可读性，使用单独的函数来分别实现消元和回代的功能。该解决思路的程序结构图已在图4-1中展示。

联立方程组的系数存储在一个二维数组中，方程组的解存储在一维数组中。变量index表示在elimination函数中被消去的变量。为了与C语言规定的下标一致，该变量值是从0到$n-1$。

高斯消元算法用伪代码来描绘是相当复杂的，因为算法中有很多繁复的下标操作。为了理清这些下标的操作流程，可以借助伪代码的逻辑手动演算一次。

[提炼后的伪代码]

主函数：读取电阻值和电压值

```
        指定数组系数，a[i][j]
        将 index 置为 0
        while index ≤ n-2
          eliminate (a, n, index)
          index 自增 1
        back_substitute (a, n, soln)
        打印电流值
eliminate (a, n, index):
        将 row 置为 index+1
        while row ≤ n-1
          将 scale_factor 置为 -a[row][index] / a[index][index]
          将 a[row][index] 置为 0
          将 col 置为 index+1
          while col ≤ n
            将 a[index][col] * scale_factor
              加到 a[row][col] 中
            col 自增 1
          row 自增 1
back_substitute (a, n, soln):
        将 soln[n-1] 置为 -a[n-1][n] / a[n-1][n-1]
        将 row 置为 n-2
        while row ≥ 0
          将 col 置为 n-1
          while col ≥ row + 1
            令 a[row][n] 减去 soln[col] * a[row][col]
          令 col 减去 1
        将 soln[row] 置为 a[row][n] / a[row][row]
        令 row 减去 1
```

一旦熟悉了伪代码中的流程，就很容易将其直接转换为 C 程序了。

```
/*-----------------------------------------------------------*/
/*  程序 chapter5_8                                          */
/*                                                           */
/*  该程序用高斯消元来确定电路中的回路电流                    */
#include <stdio.h>
#define N 3  /*  未知电流值的数量  */

int main(void)
{
   /*  声明变量和函数原型  */
   int index;
   double r1, r2, r3, r4, r5, v1, v2, a[N][N+1], soln[N];
```

274

```
   void eliminate(double a[N][N+1],int n,int index);
   void back substitute(double a[N][N+1],int n,
                        double soln[N]);
   /*  获得用户输入  */
   printf("Enter resistor values in ohms: \n");
   printf("(R1, R2, R3, R4, R5) \n");
   scanf("%lf %lf %lf %lf %lf",&r1,&r2,&r3,&r4,&r5);
   printf("Enter voltage values in volts: \n");
   printf("(V1, V2) \n");
   scanf("%lf %lf",&v1,&v2);
   /*  指定方程组的系数  */
   a[0][0] = r1 + r2;
   a[0][1] = a[1][0] = -r2;
   a[0][2] = a[2][0] = a[1][3] = 0;
   a[1][1] = r2 + r3 + r4;
   a[1][2] = a[2][1] = -r4;
   a[2][2] = r4 + r5;
   a[0][3] = v1;
   a[2][3] = -v2;
   /*  执行消元步骤  */
   for (index=0; index<=N-2; index++)
      eliminate(a,N,index);
   /*  执行回代步骤  */
   back substitute(a,N,soln);
   /*  打印方程组的解  */
   printf("\n");
   printf("Solution: \n");
   for (index=0, index<=N-1; index++)
      printf("Mesh Current %d: %f \n",index+1,soln[index]);
   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数执行消元步骤                                             */
void eliminate(double a[N][N+1],int n,int index)
{
   /*  声明变量  */
   int row, col;
   double scale_factor;
   /*  从方程组中消除变量  */
   for (row=index+1; row<=n-1; row++)
   {
      scale_factor = -a[row][index]/a[index][index];
      a[row][index] = 0;
      for (col=index+1; col<=n; col++)
         a[row][col] += a[index][col]*scale_factor;
   }
   /*  无返回值  */
   return;
}
/*-----------------------------------------------------------------*/
/*  该函数执行消元步骤                                             */
void back substitute(double a[N][N+1],int n,
                     double soln[N])
{
   /*  声明变量  */
   int row, col;
   /*  在每个方程中执行回代步骤  */
   soln[n-1] = a[n-1][n]/a[n-1][n-1];
```

275

```
    for (row=n-2; row>=0; row--)
    {
        for (col=n-1; col>=row-1; col--)
            a[row][n] -= soln[col]*a[row][col];
        soln[row] = a[row][n]/a[row][row];
    }
    /*  无返回值  */
    return;
}
/*-----------------------------------------------------------*/
```

276

如果要求解更大的方程组，符号常量N要随之改变；而高斯消元的步骤不需要做修改。

5. 测试

使用手动演算示例中的数据得到的程序执行结果如下所示：

```
Enter resistor values in ohms:
(R1, R2, R3, R4, R5)
1 1 1 1 1
Enter voltage values in volts:
(V1, V2)
5 5

Solution:
Mesh Current 1: 2.500000
Mesh Current 2: 0.000000
Mesh Current 3: -2.500000
```

该程序是假定方程组有解的情况，也就是说方程组中没有相同的或等价的方程。我们可以通过在程序中增加语句或函数来检验上述条件。

修改

用本小节开发的程序来回答下面的问题：

1. 如果电路中所有电阻都是5Ω，所有电源都是10V，计算此时的回路电流。
2. 使用本小节中讨论的矩阵乘法来验证问题1中得出的答案。（该问题假设你已经学习了前面小节中的矩阵和向量部分的知识。）
3. 如果电路中的电阻值分别为2Ω、8Ω、6Ω、6Ω和4Ω，并且电压为40V和20V，计算此时的回路电流。
4. 将问题3中得到的答案回代到原始方程组中加以验证。

*5.13 多维数组

C语言中允许数组可以被定义为超过两个下标的形式。例如，下面的语句定义了一个三维数组（three-dimensional array）：

```
int b[3][4][2];
```

三个下标共同确定一个特定元素，同时也在三维空间中标定了一个位置的x、y、z三个轴上的坐标，如图5-7所示。如此，图中阴影的位置对应着数组元素b[2][0][1]。

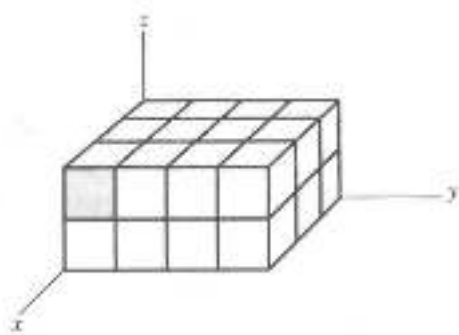


图5-7 三维数组

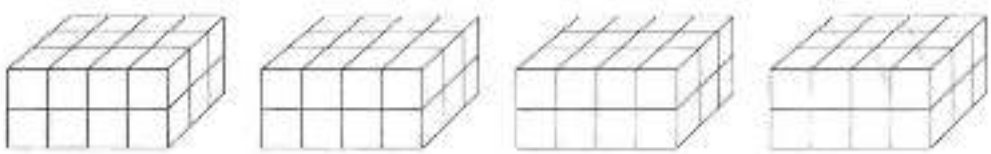
277

对于大多数需要用到数组的工程问题来讲，通

常一维和二维数组就能解决问题。然而在某些特殊场景下，更高维度的数组就会派上用场。这些问题包含的数据通常由几个参数来指定，同时，这些参数要么是顺序的整数，要么很容易转换为顺序的整数。例如，假设现在有一组从大型化学反应室中测量的地面温度数据。这组数据是在化学反应过程中按照指定的时间间隔测量的。这种情况下，采用三维数组存储数据是个很好的选择。其中第一个下标表示特定的时间间隔，另外两个下标表示地面位置的坐标。按照C语言的语法规则，该下标都是从0开始。那么，下标[3][2][5]就表示该温度是在第四个时间点，坐标为[2][5]的位置上测量的。

超过三个下标的数组几乎不会被用到，因为很难将其形象化表示。但是，可以设计一种简单方法来表示这类数组。首先，将三维数组想象成一座建筑。建筑带有地面，并且每一层都是长方形的网格状房间。假设每间屋子都包含一个数值。这个三维数组用三个下标来唯一指定一个房间：第一个下标表示楼层号，另外两个下标指定对应楼层中房间的行列号。

一个四维数组（four-dimensional array）就是一排这样的建筑，如图5-8所示。其中第一个下标指定相应的建筑，剩下的三个下标来确定该建筑中的房间。



278

图5-8 四维数组

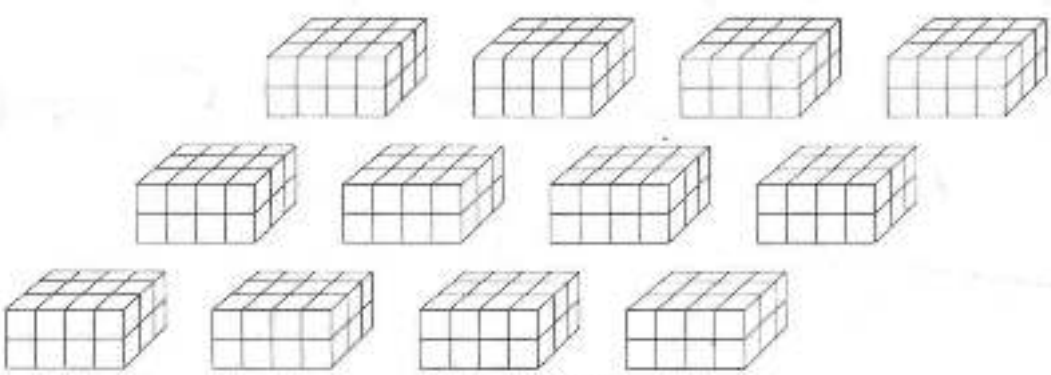


图5-9 五维数组

一个五维数组（five-dimensional array）就是由许多建筑组成的街区，如图5-9所示。前两个下标指定街区中对应的建筑物，然后剩下的三个下标来确定建筑中的房间。

这种类比还可以继续推演成一排街区、一座城市、一组城市、一个州（省）等。尽管上面展示了如何表示多维数组，但是在使用中仍然要特别注意。多维数组随着下标数量的增加程序的开销会急剧增加：不仅需要增加额外的下标数，而且当以组为单位访问数组的值时，还会增加额外的循环开销。一般来讲，多维数组会使程序调试和维护的复杂度大大增加。因此，只有当多维数组可以精简问题模型同时简化解决方案时，才会加以采用。

本章小结

数组是数据结构的一种，常用来存储工程问题中待分析的数据。如果数据适合用一列信息来表示，则应该使用一维数组；如果数据适合用表格或网格信息来表示，则应该使用二维数组。本章已经

介绍了许多例子来说明数组的定义、初始化、计算、输入/输出等基本使用方法，以及数组作为函数参数等问题。同时，针对一维数组的分析和排序等常见问题，设计了一组统计函数来专门计算数组的统计学相关数据。另外，本章还介绍了高斯消元方法来求取一组联立线性方程组的解，并且给出了该方法的C语言程序实现。

关键术语

array（数组）
binary search（二分搜索）
call-by-address（传址调用）
collating sequence（排序序列）
determinant（行列式）
dot product（点积）
element（元素）
mean（均值）
median（中位数）
nonsingular（非奇异矩阵）
one-dimensional array（一维数组）
parsing（解析）
power（功率）
selection sort algorithm（选择排序算法）
sequential search（顺序查找）
simultaneous linear equations（联立线性方程组）
sorting（排序）
Gauss elimination（高斯消元）
hyperplane（超平面）
ill conditioned（弱条件）
inner product（内积）
magnitude（振幅）
matrix（矩阵）
matrix multiplication（矩阵乘法） 229
square matrix（方阵）
standard deviation（标准差）
subscripts（下标）
system of equations（方程组）
transpose（转置）
two-dimensional array（二维数组）
variance（方差）
vector（向量）
zero crossing（过零点）

C语句总结

数组声明：

```
int a[5], b[]={2,3,-1};
char vowels[]={'a','e','i','o','u'};
double x[10][5];
```

注意事项

1. 一维数组的下标通常用变量 k 表示。
2. 使用符号常量去声明数组大小，以方便修改。
3. 在程序的注释文档部分，用一个有行和列标志的网格来描述二维数组以方便理解。
4. 二维数组的下标通常用变量 i 和 j 表示。
5. 在形参列表及函数原型语句中需要列出数组行和列的大小。

调试注意事项

1. 如果需要将所有数据都存储在内存中，那么只能使用数组来存储数据。
2. 当引用一个数组元素时，注意不要超过下标值的上限。
3. for 循环的条件判断语句中，使用大于等于符号，比较的值设置成下标的最大值。这样可以避免数

组访问越界。

4. 声明数组时，数组的容量要大于等于其存储数据的最大可能值。

5. 因为在函数中引用的数组属于传址调用，所以要注意不要在函数中错误地修改数组中的值。

6. 当引用一个多维数组元素时，要确保每个下标都被自己的一对方括号括住。

7. 将矩阵标记转化为C语言时，要记得矩阵的第一行和第一列在数学问题中的下标为1，而在C语言中的下标为0。

280 8. 多维矩阵会增加程序逻辑的复杂性，所以要确保只在必要的时候才使用多维矩阵。

习题

简述题

判断题

判断下列语句的正（T）误（F）。

1. 如果初始序列的长度小于数组的长度，那么余下元素的值都被初始化为0。 T F

2. 如果数组的长度没有被定义，但是被初始化为一个序列，那么该数组的大小是任意的。 T F

3. 如果数组的下标值大于数组元素的最大下标，通常会发生执行错误。 T F

4. 如果在打印语句中给出一个数组的标识符而不指定下标，那么整个数组元素都会被打印出来。 T F

多选题

5. 数组是（　　）。

(a) 一组有着统一变量名的数值，并且都具有相同的数据类型

(b) 一组具有不同数据类型的数据元素，并且都存储在相邻的内存区域

(c) 一个变量，它存储着多个具有相同数据类型的数值

(d) 一块内存区域，存储着多个数据类型相同的数值

6. 要找到数组中的一个元素，可以通过指定（　　）。

(a) 数组名和元素个数　　(b) 数组名

(c) 元素个数后面跟着数组名　　(d) 元素个数

7. 数组下标标识了数组中指定元素的（　　）。

(a) 位置　　(b) 数值

(c) 范围　　(d) 名称

8. 下面这段代码是（　　）。

```
sum = 0;
for (row=0; row<N_ROW; row++)
   sum += table[row][2];
```

(a) 第二行数值之和　　(b) 第二列数值之和

(c) 行N_ROW中的数值之和　　(d) 以上都不是

内存快照题

给出下面每组语句执行时的内存快照（对于未被初始化的数组元素用问号标识）：

9.
```
int k, t[5];
...
t[0] = 5;
for (k=0; k<=3; k++)
   t[k+1] = t[k] + 3;
```

281

10.
```
int r, c, x[4][5];
...
for (r=0; r<=3; r++)
   for (c=0; c<=4; c++)
      x[r][c] = r + c;
```

程序输出题

参考如下语句，回答 11 题和 12 题：

```
int sum, k, i, j;
int x[4][4]={{1,2,3,4},{5,6,7,8},{9,8,7,3},{2,1,7,1}};
```

11. 写出下列语句执行后 sum 的值：

```
sum = x[0][0];
for (k=1; k<=3; k++)
   sum += x[k][k];
```

12. 写出下列语句执行后 sum 的值：

```
sum = 0;
for (i=1; i<=3; i++)
   for (j=0; j<=3; j++)
      if (x[i][j] > x[i-1][j])
         sum++;
```

编程题

线性插值。下列问题中使用了一些风洞试验测试数据。这些数据可以从配套的网站上下载到。数据文件中包含了航迹角（用角度值表示），同时文件中每一行都包含了相应的提升力系数。其中，航迹角为升序排列。

13. 编写程序，读取风洞试验数据，并且让用户输入一个航迹角。如果该角度在数据集的范围之内，通过线性插值计算相应的升力系数。（可以参考 2.5 节中有关线性插值的介绍。）
14. 修改 13 题中编写的程序，使其在屏幕上显示出文件中包含的角度的范围。
15. 编写函数来检查航迹角是否是升序排列的。如果角度值是按升序排列，函数返回值为 1；否则返回值为 0。假设对应的函数原型为

```
int ordered(double x[],int num_pts);
```

16. 编写函数，以两个一维数组作为参数。两个数组分别包含航迹角和相应的升力系数。调整航迹角数据的顺序使其按升序排列，同时在升力系数数组中要保持与航迹角的对应关系。假设对应的函数原型为

```
void reorder(double x[],double y[],int num_pts);
```

17. 修改 14 题中编写的程序，使其通过 15 题中设计的函数来确定数据是否按序排列。如果没有按序排列，就使用 16 题中设计的函数来重新排序。

噪声信号。在工程模拟中，常常会需要一个具有指定均值和方差的浮点数序列。第 4 章设计的函数能够生成在边界 a 和 b 之间的数值，但是无法指定均值和方差。通过使用概率论中的相关结论，可以得出一个均匀分布的随机序列同它的理论均值 μ、方差 σ^2 之间的关系：

282

$$\sigma^2 = \frac{(b-a)^2}{12}, \quad \mu = \frac{a+b}{2}$$

18. 编写程序，用第 4 章设计的函数 rand_float 来生成一个随机浮点数序列，数值在 4 ~ 10 之间。然后计算出生成的随机数列的均值和方差，并同理论均值与理论方差相比较。生成的随机数越多，

计算值与理论值应该越接近。

19. 编写程序，用第4章设计的函数rand_float来生成两个分别具有500个元素的数列。两个数列的理论均值应该都为4，但方差一个为0.5，另一个为2。计算生成数列的均值，并同理论均值相比较。（提示：用前面的两个方程来组成带有两个未知数的方程组，然后手算出方程组的两个解。）

20. 编写程序，用第4章设计的函数rand_float来生成两个分别具有500个元素的数列。两个数列应该具有相同的方差3.0，但均值一个为0.0，另一个为-4.0。计算生成数列的均值和方差，并同理论均值与方差相比较。（提示：用前面的两个方程来组成带有两个未知数的方程组，然后手算出方程组的两个解。）

21. 编写程序，用第4章设计的函数rand_float来设计一个名为rand_mv的函数。函数参数为指定的均值和方差。每次调用该函数都能生成一个符合该均值和方差的随机浮点数。假设对应的函数原型为

```
double rand_mv(double mean,double var);
```

电厂数据。文件power1.dat包含了一个电厂在8个星期内的电力输出（以百万瓦特记）。文件中的每行数据都包含7个整数，代表着一周7天的数据。在下面的程序设计中，使用数组来存储数据。并用符号常量NROWS和NCOLS来表示数组的行数和列数。（请手动生成一个文件以测试下列问题。）

22. 编写程序，计算在这段时间内的平均输出功率，并打印结果。同时，还应该打印出大于平均输出功率的天数。

23. 编写程序，找出产生最小功率输出的时间是那一星期的哪一天。如果同时有几天都产生了最小功率输出，分别打印出每一天的信息。

24. 假设有一个二维数组，行数和列数分别为NROWS和NCOLS。设计函数来计算数组指定列的平均值。函数的参数应该是一个整型数组和一个指定的列号。假设对应的函数原型为

```
double col_ave(int x[NROWS,NCOLS],int col);
```

25. 编写程序，使用24题中设计的函数来打印一份报告，其中要列出一周之中第一天的平均功率输出，第二天的平均功率输出，以此类推。打印信息的格式如下：

```
Day x: Average Power Output in Megawatts:  xxxx.xx
```

283 26. 假设有一个二维数组，行数和列数分别为NROWS和NCOLS。设计函数来计算数组指定行的平均值。函数的参数应该是一个整型数组和一个指定的行号。假设对应的函数原型为

```
double row_ave(int x[NROWS,NCOLS],int row);
```

27. 编写程序，使用26题中设计的函数来打印一份报告，其中要列出第一周的平均功率输出，第二周的平均功率输出，以此类推。打印信息的格式如下：

```
Week x: Average Power Output in Megawatts:  xxxx.xx
```

28. 编写程序，计算并打印出电厂输出数据的均值和方差。

密码学。几个世纪以来，一直都有很多人对密码学兴趣浓厚。一些简单的编码方式是将某个字符或某一组字符替换成另一个或另一组字符。而解码的关键，就是要能够找到这些字符之间的替换关系。这种关键数据常被称为“密钥”。近年来，计算机已经能将很多原本被认为无法破解的密码成功解码。下面的这组问题涉及一些简单密码以及如何解密的方案。生成文件测试下列程序。

29. 有一种简单的密码设计方法，就是将序列中的每个字符都替换成与其相隔固定距离的另一个字符。例如，如果每个字母都要替换成它右侧距离为2的字母，那么字母a要被替换成字母c。字母b要

被替换成字母 d，以此类推。编写程序，读取一个文本文档，并生成一个加密过的新文档。加密方案就使用上面介绍的方法。注意不要改变其中的换行符和 EOF（文件结束符）。

30. 编写程序，对 29 题中的加密方案进行解码，并使用 29 题中生成的文件来测试该程序。

31. 如果要在不知道加密方案的情况下，对一个简单密码进行解密（例如 29 题中设计的加密文件），第一步需要计算每一个字符出现的次数。然后，已知最常见的英文字母是 e，于是将加密信息中最常出现的字母用 e 来替换。随后便根据加密信息中字母的出现频率与已知的英文字母出现频率来继续进行类似的替换。这种解密方式经常能得到足够多的正确的替换关系，从而也能据此来判断出其中错误的替换关系。为解决以上问题，首先需要编写程序，读取一个数据文件，并确定出文件中每个字母的出现频次。然后将字母同它的出现次数逐个打印出来。没有出现过的字母不需要打印。（提示：基于字母的 ASCII 码值，使用一个数组来存储字母的出现频次。）

32. 另一种简单的文本信息编码方式是，将真实信息用每个单词的首字母来表示。单词之间不留空格，但是解码的人可以轻易地提取出这些字母以拼成单词。编写程序，读取一个数据文件，提取单词首字母得出一中字符序列来确定真实信息。

33. 假设 32 题中的真实信息是通过存储每个单词第二个字母来加密的。编写程序，读取一个数据文件，然后确定出文件中的真实信息。

34. 假设 32 题中的真实信息通过如下方式加密：将每个单词的首字母替换成其右侧距离为 3 的字母。编写程序，读取一个数据文件，然后确定出文件中的真实信息。

35. 编写程序，使用一个名为 key 的键盘数组来对文本信息进行加密。数组中包含了 26 个字符。数组中的元素从键盘读取；其中第一个字符用来替换数据文件中的字母 a，第二个字符用来替换文件中的字母 b，以此类推。假设所有标点符号用空格来替换。同时检查数组，要确保在加密过程中不会出现将两个不同字母映射成同一个字符的情况。 284

36. 编写程序，将 35 题中的输出文件进行解码。解码时使用与上题相同的键盘数组 key，在程序中同样从键盘读取。要注意的是，标点字符将无法恢复。

温度分布。假设一个金属薄盘，盘上各边的温度是稳定的，此时金属盘上的温度分布可以构建成一个二维网格模型，如图 5-10 所示。在一个典型模型中，网格中的点数是指定的，同时 4 个边上的恒定温度也是指定的。金属盘内部的温度一开始都初始化为零，但它们会随着四周的温度而改变。一个内部点的温度可以通过计算它的 4 个相邻点温度的平均值来得到；在图 5-10 中，阴影部分的 4 个点就代表网格中 x 点的 4 个相邻点。每当一个内部点的温度发生变化时，其相邻点的温度也随之改变。这些变化会一直持续，直到达到一个热平衡，所有温度也都趋于恒定。

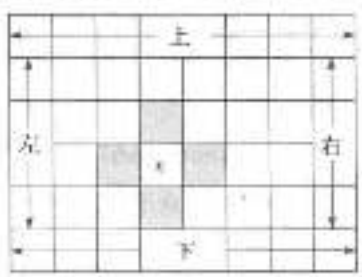


图 5-10 金属盘上的温度分布网格

37. 编写程序来构建一个 6 行 8 列的温度分布网格模型。令用户输入 4 个边的温度值，并使用数组存储。那么当一个点的温度更新时，更新值也会用来更新下一个点的温度。逐行持续更新这些点的温度，直到所有更新后的两个点之间的温差小于一个用户输入的容许值为止。

38. 修改 37 题中编写的程序，使得温度更新是按列进行的。两个程序使用不同的公差，比较它们最后达到的恒定温度。恒定温度值应该是非常接近的（可能存在一个很小的误差容许值）。

39. 修改 37 题中编写的程序，使得行和列两个方向上的温度同时更新。为了方便计算，可以使用两个数组，一个数组保存更新之前的温度值，另一个数组用来计算温度更新。

高斯消元。高斯消元方法可以通过选主元的过程来提高准确性。为了进行行主元消元，首先将方程组

重新排列，对方程的第一个系数取绝对值，值最大的方程便作为第一个方程。然后从第二个方程开始，消除方程中的第一个变元。接下来，从第二个方程开始，将其余的方程组继续重新排列，跟前面的过程类似，选取方程中第二个变元系数（绝对值）最大者作为第二个方程。然后从第三个方程开始，消除方程中的第二个变元。对余下的变元继续执行类似的过程。假设符号常量N代表方程数。

285

40. 以5.12节中开发的程序作为参考，设计一个求解联立方程组的函数。该函数的第一个参数是一个N×N大小的double型二维数组a，第二个参数是大小为N的double型一维数组soln。数组a中存储了待求解的方程组；而数组soln中存储方程组的解。假设相应的函数原型为：

```
void gauss(double a[N][N+1],double soln[N]);
```

41. 编写函数，参数分别为一个二维数组和进行系数主元选择的行数j。该函数要从第j个方程开始对所有方程重新排列，使得第j个方程在第j个位置的系数（绝对值）最大。假设函数引用的二维数组大小为N×（N+1）。相应的函数原型为：

```
void pivot_r(double a[N][N+1],int j);
```

42. 修改40题中设计的函数，使得在每次消元操作进行之前先选出行主元，并用主元系数进行消元。利用41题中编写的函数来解题。

43. 列变换和行变换类似：通过交换列使得最大系数（绝对值）在自己最感兴趣的位置。当列被交换，一定要注意追踪相关变元顺序的变化情况。编写函数来执行列变换，函数参数中要包含变元的顺序变化。假设相应的函数原型为：

```
void pivot_c(double a[N][N+1],int j,int reorder k[N]);
```

44. 修改40题中设计的函数，使得在每个变元被消除之前完成列变换。用列主元进行消元。利用43题中编写的函数来解题。

45. 修改40题中的函数，使得在每个变量被消除之前完成行变换和列变换，用完全主元进行消元。利用41题和43题中编写的函数来解题。

行列式。下列问题定义了方阵的代数余子式和余子式，利用它们来完成行列式计算。

46. $a_{i,j}$是矩阵A中的一个元素，则它的余子式定义为，将包含给定元素$a_{i,j}$的行和列移除之后得到的矩阵的行列式。因此，如果初始矩阵有4行4列，那么余子式就是一个3行3列的矩阵的行列式。编写函数计算一个4行4列方阵的余子式。函数参数为矩阵A与i和j的值。假设相应的函数原型为：

```
double minor(double a[4][4],int i,int j);
```

47. 矩阵A的代数余子式$A_{i,j}$就是元素$a_{i,j}$的余子式和因子$(-1)^{i+j}$的乘积。编写函数，计算一个4行4列方阵的代数余子式。函数参数为矩阵A与i和j的值。解题时需要调用46题中的函数。假设相应的函数原型为：

286

```
double cofactor(double a[4][4],int i,int j);
```

48. 方阵A的行列式可以用以下方法计算：

(a) 选择任意一列。

(b) 该列中的每一个元素分别与对应的代数余子式相乘。

(c) 将步骤（b）中得到的乘积相加。

编写函数det_c，使用这种方法来计算一个4行4列矩阵的行列式。可以调用47题中编写的函数。假设相应的函数原型为：

```
double det_c(double a[4][4]);
```

49. 方阵 A 的行列式可以用以下方法计算：

(a) 选择任意一行。

(b) 该行中的每一个元素分别与它的代数余子式相乘。

(c) 将步骤 (b) 中得到的乘积相加。

编写函数 det_r，使用这种方法来计算一个 4 行 4 列矩阵的行列式。可以调用 47 题中编写的函数。假设相应的函数原型为：

```
double det_r(double a[4][4]);
```

标准化方法。对于一组值有许多标准化或缩放处理的方法。一种常用的标准化方法是将目标值缩放，使其最小值变为 0，最大值变为 1，其他值以相应比例进行缩放。例如，使用这种方法可以将下列数组进行标准化：

数组值				标准化的数组值			
−2	−1	2	0	0.0	0.25	1.0	0.5

计算数组中 x_k 对应的标准化值的计算公式为：

$$\text{标准化的 } x_k = \frac{x_k - \min_x}{\max_x - \min_x}$$

其中，$\min_x$ 和 $\max_x$ 分别表示数组 x 中的最小值和最大值。用 x_k 代替最小值，则公式的分子为 0，所以数组最小值的标准化值为 0。如果用 x_k 代替最大值，则分子和分母相同，因此数组最大值的标准化值为 1.0。

50. 编写函数，以一个一维的 double 型数组和数组中的元素个数作为函数参数。使用上述方法对数组值进行标准化处理，假设相应的函数原型为：

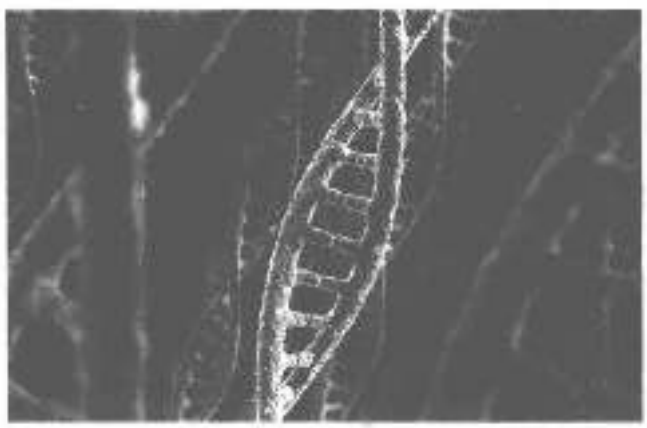
```
void norm_1D(double x[],int num_pts);
```

51. 编写函数，以一个二维的 double 型数组作为函数参数。使用上述方法对数组值进行标准化处理。假设符号常量 NROWS 和 NCOLS 指定了数组的行数和列数，并且在函数中可直接使用。假设相应的函数原型为：

```
void norm_2D(double x[NROWS][NCOLS]);
```

指针编程

犯罪现场调查：DNA 分析



嫌疑人的 DNA 和从犯罪现场收集到的 DNA 样本之间的匹配结果被认为是一项强有力的证据。美国联邦调查局（FBI）拥有一个称为 CODIS（Combined DNA Index System，DNA 联合索引系统）的数据库，涵盖了超过 900 万个 DNA 图谱。这些图谱包含了给定 NDA 片段的核苷酸序列。DNA 核苷酸是腺嘌呤（A）、胞嘧啶（C）、鸟嘌呤（G）和胸腺嘧啶（T）中的任意一种。所以说，DNA 图谱实际上就是由代表核苷酸的字母组成的顺序字符串，比如 AAACGTACAGGT。虽然对每个人来说，人类 99.9% 的 DNA 序列都是相同的，但是在 DNA 匹配工作中仍存在大量差异需要比对。判断未知 DNA 样本与数据库中 DNA 样本是否匹配，需要基于两个字符串间的大量短字符串（称为短串联重复序列（Short Tandem Repeats，STR））的对比比对。DNA 测序仪现在已经可以自动确定 DNA 片段的核苷酸序列，因此 DNA 的司法鉴定成为执法工作的一个有力工具。这些仪器使用激光来分析附着在核苷酸上的荧光物所发出的光信号以确定它们的类型。这项技术最初是由英国遗传学家 Alec John Jeffreys 于 1984 年研究发现的，后来以此为基础发展成了 DNA 匹配技术，也称为基因指纹技术。在本章将设计一个 C 程序，在一个较大的 DNA 链中找到较小的 DNA 片段。

288

学习目标

在本章，我们将学到以下解决问题的方法：

- 指向变量的指针。
- 指向数组的指针。
- 在函数调用中使用指针。
- 指向字符串的指针。
- 动态内存分配。

6.1 地址和指针

当 C 程序执行时，存储单元被分配给程序中的变量。每个存储单元都有一个唯一确定的地址（address），是一个用于标定位置的正整数。当一个变量被赋值，这个值便存储在相应位置的存储单元中。程序中的语句既可以读取变量值，也可以更改变量值。变量的具体地

址在每次程序执行时确定，并且每次执行时地址都有可能发生变化。

为了便于理解，有时会将内存分配比作一组邮局信箱。如果邮局拥有 100 个信箱，编号从 1 到 100，那么邮箱号就对应着内存地址。每个邮箱都分配给一个人，并以这个人的名字来命名；这个名字对应于变量的标识符。此时变量已经分配到了特定的存储位置上。而邮箱内容对应于存储单元内的值；该值可以被检查和更改。 289

邮局信箱编号	用户名称	内容
78	John Ruiz	目录册
内存地址	标识符	内容
66572	x	105

这样的类比其实并不完全贴切，因为两个人可能拥有同一个名字，但是两个标识符却不能完全相同。此外，邮箱既可以为空也可以包含很多信件，但是存储单元却总是包含一个数值。

6.1.1 地址运算符

C 语言中，变量地址可以通过地址运算符（address operator）& 来引用。该运算符在第 2 章里连同 scanf 语句一同介绍过。例如，使用如下语句可以从键盘读取一个浮点数值，并存储在变量 x 中：

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

这条语句指定了从键盘读取的值要存储在由 &x 指定的地址单元中，也就是变量 x 的地址。以下程序演示了使用地址运算符获得变量的内存地址的方法。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter6_1                                                */
/*                                                                 */
/*  该程序说明变量和地址间的关系                                    */

#include <stdio.h>

int main(void)
{
   /*  声明和初始化变量  */
   int a=1, b=2;

   /*  输出 a 和 b 的内容和地址  */
   printf("a = %d;  address of a = %u \n",a,&a);
   printf("b = %d;  address of b = %u \n",b,&b);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

注意，地址是用 %u 标识符输出，此标识符专门用于输出无符号整数。以下是本程序可能出现的一个输出结果：

```
a = 1;  address of a = 1245052
b = 2;  address of b = 1245048
```

以下内存快照显示了当 printf 语句执行时两个存储单元中的值：

290

我们通常并不会在图表中展示内存地址，因为程序使用的地址是系统相关的。可以通过修改程序使得变量 a 和 b 没有指定初始值，修改后的程序如下所示：

```
/*-----------------------------------------------------------------*/
/*  程序 chapter6_2                                                */
/*                                                                 */
/*  该程序说明变量和地址间的关系                                    */

#include <stdio.h>

int main(void)
{
   /*  声明和初始化变量  */
   int a, b;

   /*  输出 a 和 b 的内容和地址  */
   printf("a = %d;  address of a = %u \n",a,&a);
   printf("b = %d;  address of b = %u \n",b,&b);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

在 printf 语句执行时，内存快照应该显示变量中的内容是一个问号，因为程序中并没有给出变量值：

以下是本程序可能出现的一个输出结果：

```
a = -858993460;  address of a = 1245052
b = -858993460;  address of b = 1245048
```

从结果中可以看到，这些变量是有值的（即使在程序中并没有赋值给变量），并且这些变量值是不可预知的。这个例子说明了，在变量用于其他语句之前，一定要先对其进行赋值。

修改

1. 在计算机上将程序 chapter6_1 执行两次。观察计算机使用了同样的地址还是不一样的地址？和同学之间比较执行的结果。
2. 运行本节给出的程序 chapter6_2。观察分配给 a 和 b 的存储单元中的值是多少？如果这些值是 0，则编译器可能会给未初始化的变量分配 0 值。这不是美国国家标准协会（ANSI）所规定的标准，因而不应该假定未初始化的变量值都为 0。程序每次执行后观察这些值是否发生变化？

291

6.1.2 指针赋值

C 语言允许用一种特殊类型的变量来存储内存单元的地址，也就是指针（pointer）。当一个指针变量被定义时，它所指向的变量类型也必须要被定义。因此一个指向整型变量的指针不能用来指向浮点型变量。

C 语言中使用星号来标识一个指针变量；因此，这个星号又被称为间接引用（dereferencing）运算符或复引用（indirection）运算符。假设一条语句定义了两个整型变量和一个指向整型变量的指针。这条语句写法如下：

```
int a, b, *ptr;
```

该语句表明应该为三个变量分配内存地址——两个整型变量和一个指向整型变量的指针。语句中没有为 a 和 b 赋值，也没有指明存储在 ptr 中的地址。因此，这条声明语句执行后的内存快照应该显示所有变量都不具有初始值。下图用箭头表明 ptr 是一个指针变量：

为了使 ptr 指向变量 a，应该使用一条赋值语句在 ptr 中存储 a 的地址：

```
int a, b, *ptr;
ptr = &a;
```

也可以在声明语句中为指针赋值：

```
int a, b, *ptr=&a;
```

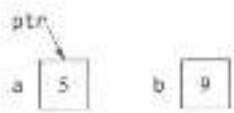
无论哪种方法，在声明语句执行后的内存快照如下图所示：

需要注意的是，我们只关心 ptr 指向的变量，而没有必要显示 ptr 的内容。

考虑下面这组语句：

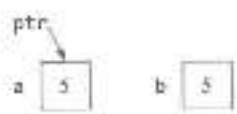
```
/*  声明和初始化变量  */
int a=5, b=9, *ptr=&a;
...
/*  将 ptr 的内容赋值给 b  */
b = *ptr;
```

最后一条语句应该读作“将 ptr 中的地址位置上所存储的内容赋值给 b”，或者是“将 ptr 指向的值赋给 b”。在声明语句执行后，内存快照如下所示：



292

在赋值语句执行后，内存快照如下所示：



可以看到，b 被赋予了 ptr 所指向的值。

现在考虑下面这组语句：

```
/*  声明和初始化变量  */
int a=5, b=9, *ptr=&a;
...
/*  将 b 的值赋给 ptr 所指向的变量  */
*ptr = b;
```

在赋值语句执行之前，内存快照如下所示：

赋值语句执行后，内存快照如下所示：

可以看到，ptr 所指向的变量被赋予了 b 的值。

现在扩展程序 chapter6_1 来说明变量、地址和指针之间的关系。考虑下面的程序：

```
/*-----------------------------------------------------------*/
/*   程序 chapter6_3                                         */
/*                                                           */
/*   该程序说明了变量、地址和指针之间的关系                  */

#include <stdio.h>

int main(void)
{
   /*  声明和初始化变量  */
   int a=1, b=2, *ptr=&a;

   /*  输出变量和指针内容  */
   printf("a = %d;  address of a = %u \n",a,&a);
   printf("b = %d;  address of b = %u \n",b,&b);
   printf("ptr = %u;  address of ptr = %u \n",ptr,&ptr);
   printf("ptr points to the value %d \n",*ptr);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------*/
```

293

以下为本程序可能的一个输出结果：

```
a = 1;  address of a = 1245052
b = 2;  address of b = 1245048
ptr = 1245052;  address of ptr = 1245044
ptr points to the value 1
```

注意 ptr 是指向 a 的指针，那么 ptr 的值实际上就是 a 的地址。

练习

给出下列每组语句执行后的内存快照：

```
1. int a=1, b=2, *ptr;
   ...
   ptr = &b;
```

```
2. int a=1, b=2, *ptr=&b;
   ...
   a = *ptr;
```

```
3. int a=1, b=2, c=5, *ptr=&c;
   ...
   b = *ptr;
   *ptr = a;
```

```
4. int a=1, b=2, c=5, *ptr;
   ...
   ptr = &c;
   c = b;
   a = *ptr;
```

在第3章中使用了指针来访问数据文件。回想一下，指向文件的指针（称为文件描述符）是使用FILE声明语句来定义的，例如

```
FILE *sensor;
```

其中FILE数据类型是在头文件<stdio.h>中定义。通过使用fopen语句将文件指针同一个特定文件相关联，如下列语句所示：

```
sensor = fopen("sensor1.txt","r");
```

294

这条语句还说明了sensor1.txt是一个输入文件，其中参数r表明从文件中读取信息。而在fscanf语句中再次使用该指针，它指向要读取数据的文件：

```
fscanf(sensor,"%f %f",&t,&motion);
```

函数fscanf中的文件指针是必需的，因为在同一个程序中可能要分别从几个不同文件中读取信息。在fprintf函数中指针变量的用法类似，只是需要把文件指针指向一个输出文件。

6.1.3 地址运算

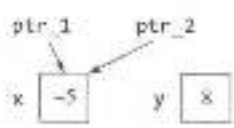
指针（或地址）运算具有以下限制：

- 指针可以赋值给另一个相同类型的指针。
- 指针可以加减一个整数值。
- 指针可以被赋值为整数0或符号常量NULL；同样，指针也可以与整数0或NULL进行比较。其中符号常量NULL被定义在头文件<stdio.h>中。
- 作为访问数组元素的一种方法，指向同一数组元素的指针之间可以相减或比较。

一个指针只能指向一个地址，但是不同的指针却可以指向同一地址。下列语句执行后，ptr_1和ptr_2便指向同一个变量：

```
/*  声明和初始化变量  */
int x=-5, y=8, *ptr_1, *ptr_2;
...
/*  将x的地址赋给两个指针  */
ptr_1 = &x;
ptr_2 = ptr_1;
```

语句执行后的内存快照如下所示：



为了说明在进行指针操作时可能会出现的常见错误，现在列出一些使用指针变量的无效语句：

```
&y = ptr_1;          /* 无效语句——试图改变 y 的地址 */
ptr_1 = y;           /* 无效语句——试图将 ptr 更改为一个非地址的值 */
*ptr_1 = ptr_2;      /* 无效语句——试图将地址赋给一个整型变量 */
ptr_1 = *ptr_2;      /* 无效语句——试图将 ptr_1 更改为一个非地址的值 */
```

295

画出这些无效语句的内存快照将有助于加深对指针用法的理解。这些非法语句都试图在指针中存储一个变量值，或者在变量中存储一个指针值。为了避免这些错误，应该为指针变量使用专门的标识符名称，以明确表明标识符与指针的关联关系。

当定义简单变量时，不能想当然地假设被分配的存储单元之间的关系。例如，声明语句定义了两个整数 a 和 b，不能就此假设两个值在内存中相邻，也不能假设在内存中谁先谁后。简单变量的内存分配是系统相关的。但是，对数组的内存分配则可以确定是一组连续的存储单元。因此，如果数组 x 包含 5 个整数，则 x[1] 的内存位置紧邻在 x[0] 的内存位置之后，x[2] 的内存位置紧邻在 x[1] 的内存位置之后，以此类推。因此，如果 ptr_x 是一个指向整数的指针，要令它指向整数 x[0]，则可以通过下面的语句来实现初始化：

```
ptr_x = &x[0];
```

要将指针移动，使其指向 x[1]，可以将 ptr_x 加 1 使其指向紧邻在 x[0] 后面的值，或者也可以直接将 x[1] 的地址赋值给 ptr_x。因此，可以使用下列任一条语句，使当前指向 x[0] 的指针 ptr_x 指向 x[1]：

```
++ptr_x;             /* 指针 ptr_x 自增 1，来指向内存中的下一个值 */
ptr_x++;             /* 指针 ptr_x 自增 1，来指向内存中的下一个值 */
ptr_x += 1;          /* 指针 ptr_x 加 1，来指向内存中的下一个值 */
ptr_x = &x[1];       /* 将 x[1] 的地址赋值给 ptr_x */
```

类似的，语句

```
ptr_x += k;
```

是将 ptr_x 改为指向 ptr_x 原来指向的位置后面 k 个存储单元的地址。前面介绍的都是为指针增加一个整数值的例子。类似的，指针也可以减去一个整数值。在 6.2 节会将讨论范围从一维数组扩展到多维数组。

当指针增加或减去一个整数值时，不是简单地对地址的值进行加减操作。这个整数表示的是指针要向前或向后移动时，跨过的变量的个数。例如，下面的语句

```
ptr++;
```

表明了 ptr 的值会被修改，以使其指向内存中的下一个存储单元，紧邻在 ptr 自增前所指向内存地址的后面。因为不同类型的值需要的内存大小不同，实际加到 ptr 的数值取决于变量类型。浮点型值比短整型值要求更大的内存空间，因此浮点型的地址增量大于短整型的地址增量。例如，连续短整型的内存地址可能是 45530 和 45532，而连续浮点型值的内存地址可能是 50200 和 50204。幸运的是，当用指针增加或减去一个整数值时，编译器会自动根据数据类型确定正确的内存地址变化量。

296

指针运算可以和其他运算同时出现在一条语句中，因此需要正确理解优先级和结合性，

以确保程序的正确性。地址运算符属于单目运算符，因此它应该在双目运算符之前计算。单目运算符也是从右向左执行的。表 6-1 中列出了相关的优先级规则。注意，运算优先级的改变可以通过加入圆括号来实现。

表 6-1 运算符优先级

优先级	运算符	结合性
1	( ) [ ]	从内向外
2	++ -- + - ! (type) &*	从右至左（单目）
3	* / %	从左至右
4	+ -	从左至右
5	<< >> >>=	从左至右
6	< <= > >=	从左至右
7	&&	从左至右
8	||	从左至右
9	?:	从右至左
10	= += -= *= /= %=	从右至左
11	,	从左至右

指针错误往往会导致一些难以调试的问题。更糟糕的是，在出现指针错误时，得到的计算结果是错误的，但是程序通常会照常运行并提示结果正常。许多指针错误是由于使用指针之前没有初始化而导致的。因此，要养成在程序开始时就初始化所有指针的好习惯。如果初始没有为指针分配内存地址，则应该为指针赋初值 NULL。可以使用包含形如（ptr_1==NULL）条件的 if 语句，来确定程序中是否为指针 ptr_1 分配了内存地址。为了加深对指针的理解，在继续 6.2 节的学习之前，先完成下面的练习中的问题。

练习

给出下列每个问题中，语句执行后变量和指针的内存快照。尽可能包含更多的信息。没有被初始化的存储单元用问号标记。

```
1. double x=15.6, y=10.2, *ptr_1=&y, *ptr_2=&x;
   ...
   *ptr_1 = *ptr_2 + x;
2. int w=10, x=2, *ptr_2=&x;
   ...
   *ptr_2 -= w;
3. int x[5]={2,4,6,8,3};
   int *ptr_1=NULL, *ptr_2=NULL, *ptr_3=NULL;
   ...
   ptr_3 = &x[0];
   ptr_1 = ptr_2 = ptr_3 + 2;
4. int w[4], *first_ptr=NULL, *last_ptr=NULL;
   ...
   first_ptr = &w[0];
   last_ptr = first_ptr + 3;
```

6.2 指向数组元素的指针

在第 5 章介绍了数组和数组操作，并通过使用下标来指定单个数组元素。除此之外，指

针也可以用来访问单个数组元素。使用指针和地址来引用数组通常都会快于使用下标引用；因此，如果考虑运行速度，指针引用通常是更好的选择。我们在6.1节讨论过，数组的内存分配是连续的，通过指针来引用数组正是基于这一特点实现的。

6.2.1 一维数组

考虑下面定义和初始化一维浮点型数组的声明语句：

```
double x[6]={1.5,2.2,4.3,7.5,9.1,10.5};
```

以下为该条语句执行后的内存快照：

x[0]	1.5
x[1]	2.2
x[2]	4.3
x[3]	7.5
x[4]	9.1
x[5]	10.5

x[0]表示数组中的第一个元素，x[1]表示数组中的第二个元素，x[k]表示数组中的第k+1个元素。类似的引用可以通过指针来实现。假设指针ptr被定义为double型，然后用下面的语句进行初始化：

```
ptr = &x[0];
```

297 ~ 298

该语句执行后，数组中第一个元素的地址随即被存储在指针ptr中。因此，*ptr表示x[0]，*(ptr+1)表示x[1]，*(ptr+k)表示x[k]。*(ptr+k)中k的值是相对于数组首元素的偏移(offset)。下图显示了本例中的数组的分配内存布局，同时加上了偏移量标识：

		偏移量
x[0]	1.5	0
x[1]	2.2	1
x[2]	4.3	2
x[3]	7.5	3
x[4]	9.1	4
x[5]	10.5	5

要计算数组x中所有元素的总和，可以通过数组下标和for循环来实现：

```
/*  声明和初始化变量  */
int k;
double x[6], sum=0;
...
/*  对数组x中的值求和  */
for (k=0; k<=5; k++)
   sum += x[k];
```

下面的语句是使用指针实现的，与前面使用数组下标的实现完全等价。

```
/*  声明和初始化变量  */
int k;
double x[6], sum=0, *ptr=&x[0];
...
/*  对数组x中的元素求和  */
for (k=0; k<=5; k++)
   sum += *(ptr+k);
```

注意，引用 *(ptr+k) 要求使用圆括号以保证正确的操作顺序：将 ptr 中的地址加 k，再由间接引用运算符取出 ptr+k 指向的值。如果写成 *ptr+k 的形式，结果就会出错，因为单目运算符的运算优先级高于双目运算符，程序会按照 (*ptr)+k 运算。

在第 5 章关于数组的讨论中，我们使用了数组名称作为函数参数，传递数组首元素的地址到程序中。在其他语句中，数组名称也可以被用于表示数组首元素的地址。例如，以下两条语句是等价的：

```
ptr = &x[0];
ptr = x;
```

类似的，*(ptr+k) 也等价于 *(x+k)，因此计算数组 x 的元素之和可以简化为： 299

```
/*  声明和初始化变量  */
int k;
double x[6], sum=0;
...
/*  对数组x中的元素求和  */
for (k=0; k<=5; k++)
   sum += *(x+k);
```

这个例子展示了将数组名作为指针的用法，这里数组名被当作一个指向该数组首地址的指针来使用。在大多数语句中，数组名可以代替指针，但是它不能写在赋值语句的左侧，因为其值不能被更改。

练习

假设数组 g 通过下列语句定义：

```
int g[]={2,4,5,8,10,32,78};
int *ptr1=&g[0], *ptr2=&g[3];
```

绘出数组元素的内存分配图，同时指出距离数组首元素的偏移量，并根据上述信息写出下列引用值：

1. *g	2. *(g+1)
3. *g+1	4. *(g+5)
5. *ptr1	6. *ptr2
7. *(ptr1+1)	8. *(ptr2+2)

6.2.2 二维数组

下图展示了一个数组定义、相应的图示和内存分配布局图。可以看到，该二维数组以行序存储在一段连续的内存单元里。注意内存分配图同时还显示了距离数组首元素的偏移量：

数组定义：　　int s[2][3] = {{2,4,6},{1,5,3}};

数组表：

2	4	6
1	5	3

内存分配：

		偏移量
s[0][0]	2	0
s[0][1]	4	1
s[0][2]	6	2
s[1][0]	1	3
s[1][1]	5	4
s[1][2]	3	5

[300]

练习

画出下列每个数组的内存分配布局图，并指出内存单元中存储的值，没有赋初值的用问号表示，同时指出每个数组元素距离首元素的偏移量。

1. int d[4][2]={{1,6}};
2. int g[3][4]={{5,2,-2,3},{1,2,3,4}};
3. float h[3][3]={{0}};

指针可以利用距离数组首元素的偏移量来引用二维数组中的元素。如果指针 ptr 已被初始化为指向元素 s[0][0]，那么引用 *(ptr+k) 就是利用偏移量 k 访问数组元素。下面举个简单的例子，假设现在要计算一个两行三列数组 s 中的所有元素之和，下面通过两组语句来比较使用下标和使用指针间接引用这两种方法：

方法 1

```
/*  声明和初始化变量  */
int s[2][3], srows=2, scols=3, i, j, sum=0;
...
/*  计算数组 s 中的元素之和  */
for (i=0; i<=srows-1; i++)
   for (j=0; j<=scols-1; j++)
      sum += s[i][j];
```

方法 2

```
/*  声明和初始化变量  */
int s[2][3], s_count=6, k, sum=0, *ptr=&s[0][0];
...
/*  计算数组 s 中元素之和  */
for (k=0; k<=s_count-1; k++)
   sum += *(ptr+k);
```

两种方法都能够正确计算数组的元素和。注意，因为数组的行标和列标都需要依次遍历，所以方法 1 需要使用循环嵌套。然而，方法 2 仅需要一个循环来指定距离数组首元素的偏移量。这两种方法都按照行序，逐行访问数组中的元素。通过将两个 for 循环互换，方法 1 可以被修改为逐列访问。

练习

假设整型数组 x 由下列语句定义：

```
int x[2][4]={{1,8,7,6},{2,4,-1,0}}, *xptr=&x[0][0];
```

画出内存分配布局图，并给出下列每个引用的值：

1. *xptr　　2. *(xptr+2)
3. *xptr + 2　　4. *(xptr+1) + *(xptr+3)

301

要根据引用 s[i][j] 来计算指针 sptr 的偏移量，必须要知道数组的列数，其中指针已被初始化为 &s[0][0]。如果假定 scols 表示的是数组 s 在内存中分配的列数，那么由于元素位于第 i 行 j 列，其偏移量就应该等于 i*scols+j。为了证明偏移量公式的有效性，考虑下面数组 s 的内存分配情况：

内存分配：

		偏移量
s[0][0]	2	0
s[0][1]	4	1
s[0][2]	6	2
s[1][0]	1	3
s[1][1]	5	4
s[1][2]	3	5

假设要将引用 s[0][1] 转换为距离数组初始值的偏移量。根据公式，偏移量应该是 0*scols+1，也就是 1，对应的引用应该是 *(sptr+1)。类似的，因为该数组有三列，所以引用 s[1][2] 的偏移量为 1*scols+2，也就是 5，对应的引用为 *(sptr+5)。可以使用类似公式将更高维的数组引用转换为距离首元素的偏移量。

在 C 语言中，可以将二维数组看作是一个一维数组，同时数组中的每一个元素也都是个一维数组。例如，2 行 3 列的数组 s 可以被看作是包含两个元素的一维数组，而其中每个元素又都是包含 3 个元素的一维数组。因此，引用 s[1][0] 等价于于 *s[1]。因为 s[1] 代表第一行第一个元素的地址。类似的，*(s[0]+4) 引用的是数组位置为 s[1][1] 中的值。为了增加可读性，在间接引用一个二维数组时，通常还是会使用带偏移量的指针，而不是带偏移量的一维引用。

练习

假设 a 被定义为一个包含 4 行 6 列的数组。同时，数组元素值已从数据文件中读入。使用带有偏移量的指针来执行下列操作。

1. 计算第二行元素和。
2. 计算第三列元素和。
3. 找出前三行中的最大值。
4. 找出最后四列中的最小值。

302

6.3 解决应用问题：厄尔尼诺－南方涛动现象

沿着赤道，正常情况下的海面温度是西太平洋较为温暖，而东太平洋则比较寒冷。但有一个现象会重复出现，那就是会有暖流使东太平洋（沿着加利福尼亚、墨西哥和南美洲的西海岸）的海面温度升高至 180°F。并且这种现象经常在圣诞节前后发生；因此，通常称其为厄尔尼诺现象（在西班牙语中，厄尔尼诺是男童的意思）。与此相反，海面温度变冷的现象也会在西太平洋发生，被称为拉尼娜现象（西班牙语中，拉尼娜指女童）。这些反常现象与暖流和东西向气压变化之间的南方涛动有关。厄尔尼诺－南方涛动（El Niño Southern

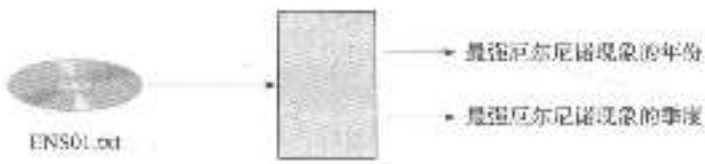
Oscillation，ENSO）指数是由一系列变量计算出的度量标准，这些变量包括气压、风力和海温。当ENSO指数为正，海水温度呈现厄尔尼诺现象；当ENSO指数为负，海水温度呈现拉尼娜现象。指数越大，温度变化越大。编写程序，从数据文件读取数据（该文件包含年份、季度和相应时期的ENSO指数），确定并输出具有最强厄尔尼诺现象的年份和季度。

1. 问题陈述

确定最强厄尔尼诺现象的年份和季度。

2. 输入/输出描述

下列I/O图显示，该程序以数据文件为输入，年份和季度为输出。



3. 手动演算示例

假设数据文件包含以下数据：

年份	季度	ENSO指数	年份	季度	ENSO指数
1990	1	0.6	1996	1	−0.3
1991	1	0.2	1997	1	−0.1
1992	1	1.1	1998	1	2.2
1993	1	0.5	1999	1	−0.7
1994	1	0.1	2000	1	−1.1
1995	1	1.2			

[303]

则下面为相应的输出结果：

```
Maximum El Nino Conditions in Data file
Year: 1998,  Quarter: 1
```

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤：

分解提纲

1）将ENSO数据读入数组，并确定最大正指数。

2）输出匹配最大正指数的年份和季度。

以下为提炼后的伪代码：

```
[提炼后的伪代码]
主函数：if文件没有打开
            输出错误信息
        else
            读取数据并计算最大正指数
            输出最大正指数相对应的年份和季度
```

伪代码中的步骤足够详细，可将其直接转换为C语言。

```
/*-----------------------------------------------------------------*/
/*  程序chapter6_4                                                 */
/*                                                                 */
/*  本程序读取包含了ENSO指数的数据文件，并确定文件中的最强厄尔尼诺现象  */

#include <stdio.h>
#define FILENAME "ENSO1.txt"
#define MAX_SIZE 1000

int main(void)
{
   /*  声明变量和函数原型  */
   int k=0, year[MAX_SIZE], qtr[MAX_SIZE], max_k=0;
   double index[MAX_SIZE];
   FILE *enso;
   /*  读取数据文件  */
   enso = fopen(FILENAME,"r");
   if (enso == NULL)
      printf("Error opening input file. \n");
   else
   {
      while (fscanf(enso,"%d %d %lf",
                    year+k,qtr+k,index+k)==3)
      {
         if (*(index+k) > *(index+max_k))
            max_k = k;
         k++;
      }
      /*  输出最强厄尔尼诺现象的相关数据  */
      printf("Maximum El Nino Conditions in Data File \n");
      printf("Year: %d, Quarter: %d \n",
             *(year+max_k),*(qtr+max_k));
      /*  关闭文件  */
      fclose(enso);
   }
   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

304

5. 测试

使用手动演算示例中的数据文件作为程序输入，得到如下输出结果：

```
Maximum El Nino Conditions in Data File
Year: 1998, Quarter: 1
```

修改

1. 修改程序，找到并输出文件中的最强拉尼娜现象。
2. 修改程序，找到ENSO指数最接近于0的厄尔尼诺现象。这应该是最接近于正常状态的情况。
3. 修改程序，输出所有发生厄尔尼诺现象的年份和季度。

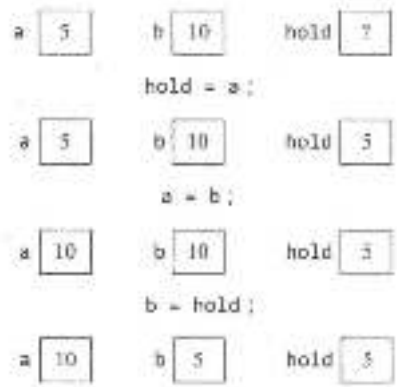
305

6.4 函数调用中的指针

C语言中，大多数函数引用为值调用。在这种情况下，实参的值被复制到形参中。函数

中所有计算均使用形参，因此函数中的实参不会发生改变。但在第4章中给出了此规则的一个例外：在函数引用中，以数组名为参数，此时数组地址被传入函数，对数组值的所有引用其实都是使用实际的数组地址。因此，函数中的语句可以对数组值进行更改。当然还有其他例外情况，比如用指针作为函数参数。

为了说明指针作为函数参数的用法，下面设计一个函数来对两个存储单元中的内容进行互换。不知道读者是否还记得，交换两个位置的数值一共需要三条语句，现将正确的交换语句和相应的内存快照展示如下：



在解决值互换的问题中，为了精简程序结构，应该通过函数来执行交换操作。现在思考下面的函数，该函数通过几个简单的变量参数（值调用）来实现交换：

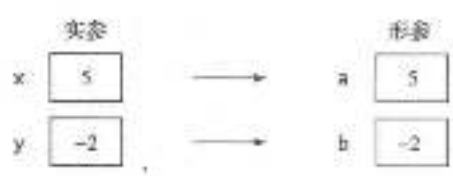
```
/*-------------------------------------------------------------*/
/*  交换两个变量值的错误函数                                    */
void switch1(int a, int b)
{
   /*  声明变量  */
   int hold;

   /*  交换a和b的值  */
   hold = a;
   a = b;
   b = hold;

   /*  无返回值  */
   return;
}
/*-------------------------------------------------------------*/
```

306

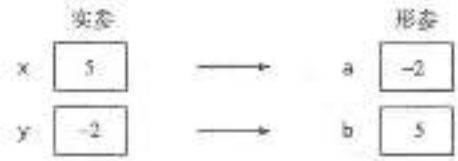
假设使用如下语句调用该函数：

```
switch1(x,y);
```

如果x和y的值分别为5和-2，则以下为函数开始执行时从实参到形参的值传递情况：



函数执行后，实参和形参的值如下所示：



因为该函数使用传值调用，实参的值（而不是地址）传递到形参。形参中的值被交换，但这些值并没有传回到实参中。

在看了上面的错误方法后，现在开始重新设计函数，这次使用指针来交换两个简单变量中的内容。该函数包含两个参数，分别是指向两个待交换变量的指针。下面是函数的原型语句：

```
void switch2 (int *a,int *b);
```

可以看到，该函数没有返回值，并且它的参数是两个指向整数的指针。下面就是可以实现两个变量值互换的正确函数：

```
/*-----------------------------------------------------------------*/
/*  交换两个变量值的正确函数                                        */
void switch2(int *a,int *b)
{
   /*  声明变量  */
   int hold;

   /*  交换a和b的值  */
   hold = *a;
   *a = *b;
   *b = hold;

   /*  无返回值  */
   return;
}
/*-----------------------------------------------------------------*/
```

307

如果 x 和 y 是两个简单变量，则下面的函数调用是合法的：

```
switch2(&x,&y);
```

如果 ptr_1 指向变量 x，ptr_2 指向变量 y，则 x 和 y 的值互换可以通过以下函数调用来实现：

```
switch2(ptr_1,ptr_2);
```

元素 x[i] 和 x[j] 的值同样可以通过以下语句实现交换：

```
switch2(&x[i],&x[j]);
switch2(x+i,x+j);
```

但是不能使用语句

```
switch2(x[i],x[j]);  /*  无效语句  */
```

指针参数对应的实参必须是一个地址或指针。

练习

思考下面对函数 switch2 的调用语句。如果是非法调用，请说明原因。如果是合法调用，则给出调用之前和之后的内存快照。

```
1. float x=1.5, y=3.0, *ptr_x=&x, *ptr_y=&y;
   ...
   switch2(ptr_x,ptr_y);
2. int f=2, g=7, *ptr_f=&f, *ptr_g=&g;
   ...
   switch2(ptr_f,ptr_g);
3. int f=2, g=7, *ptr_f=&f, *ptr_g=&g;
   ...
   switch2(*ptr_f,*ptr_g);
4. int f=2, g=7, *ptr_f=&f, *ptr_g=&g;
   ...
   switch2(&ptr_f,&ptr_g);
5. int f=2, g=7, *ptr_f=&f, *ptr_g=&g;
   ...
   switch2(&f,&g);
6. int f=2, g=7, *ptr_f=&f, *ptr_g=&g;
   ...
   switch2(f,g);
```

308

6.5 解决应用问题：地震监测

地震仪（seismometer）是一种特殊传感器，专门用来收集地球运动信息。这些地震仪可以用在被动环境中，记录包括地震和潮汐在内的各种地球运动信息。利用从若干地震仪中收集的数据来分析地震时的地表运动情况，就很有可能确定震中位置和地震强度。地震强度通常使用里氏震级（Richter scale）来衡量，它是用美国地震学家 C. F. Richter 的名字命名的，将地震强度划分为从 1 ~ 10 共 10 个等级。

编写函数，从数据文件 seismic1.txt 中读取一组地震仪测量数据。其中，文件的第一行包含两个值——后面要读取的地震仪数据的数目和连续测量的时间间隔（以秒为单位）。时间间隔是浮点型值，假设所有测量值是在相同的时间间隔内测量的。在读取测量值并存储完毕后，程序使用功率比来确定可能发生的地震情况，又被称为地震事件（seismic event）。这个比率是在指定时间点上，长时功率（long-time power）测量值除以瞬时功率（short-time power）测量值得到的商。如果比率大于给定的阈值（threshold），则在此时间点上可能发生过一次地震。一个指定点的瞬时功率，就是这个点以及该点之前一小部分测试点的功率的平均值，或者平均平方值。长时功率是指定点与该点之前一大部分点相加得到的测量功率的平均值（计算中使用的点集有时又被称为数据窗口（data window））。为了避免将静止不动时的数据误判为地震事件，采用的阈值一般大于 1，这是因为如果数据值完全相同，那么瞬时功率和长时功率会相等。在下面的问题中，假设用于瞬时功率和长时功率使用的测量点的数目是从键盘读入的，并将阈值设为 1.5。

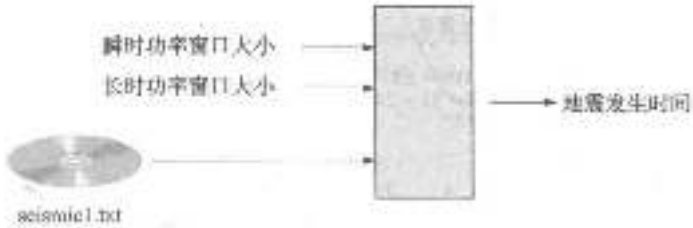
1. 问题陈述

利用从数据文件中取得的一组地震仪测量值，来确定可能发生地震的位置。

2. 输入 / 输出描述

程序输入是数据文件 seismic1.txt 以及利用瞬时功率和长时功率测量的数目。程序输

由是给出地震可能发生的时间的报告。



309

3. 手动演算示例

假设数据文件包含的数据为点的总数目（11）和点间的时间间隔（0.01），之后是对应的11个数值 x_0，x_1，…，x_{10}：

```
11 0.01
1   2   1   1   1   5   4   2   1   1   1
```

如果瞬时功率测量值由两个样本点得到，长时功率测量值由5个样本点得到，那么就可以计算功率比，在窗口中从最左边开始计算：

```
1   2   1   1   1   5   4   2   1   1   1
            瞬时窗口
        长时窗口
```

点：　瞬时功率 = (1 + 1)/2 = 1
长时功率 = (1 + 1 + 1 + 4 + 1)/5 = 1.6
比率 = 1/1.6 = 0.63

```
1   2   1   1   1   5   4   2   1   1   1
            瞬时窗口
        长时窗口
```

点：　瞬时功率 = (25 + 1)/2 = 13
长时功率 = (25 + 1 + 1 + 1 + 4)/5 = 6.4
比率 = 13/6.4 = 2.03

```
1   2   1   1   1   5   4   2   1   1   1
            瞬时窗口
        长时窗口
```

点：　瞬时功率 = (16 + 25)/2 = 20.5
长时功率 = (16 + 25 + 1 + 1 + 1)/5 = 8.8
比率 = 20.5/8.8 = 2.33

```
1   2   1   1   1   5   4   2   1   1   1
            瞬时窗口
        长时窗口
```

点：　瞬时功率 = (4 + 16)/2 = 10
长时功率 = (4 + 16 + 25 + 1 + 1)/5 = 9.4
比率 = 10/9.4 = 1.06

310

1 2 1 1 1 5 4 2 1 1 1

瞬时窗口

长时窗口

点： 瞬时功率 = (1 + 4)/2 = 2.5
长时功率 = (1 + 4 + 16 + 25 + 1)/5 = 9.4
比率 = 2.5/9.4 = 0.27

1 2 1 1 1 5 4 2 1 1 1

瞬时窗口

长时窗口

点： 瞬时功率 = (1 + 1)/2 = 1
长时功率 = (1 + 1 + 4 + 16 + 25)/5 = 9.4
比率 = 1/9.4 = 0.11

1 2 1 1 1 5 4 2 1 1 1

瞬时窗口

长时窗口

点： 瞬时功率 = (1 + 1)/2 = 1
长时功率 = (1 + 1 + 1 + 4 + 16)/5 = 4.6
比率 = 1/4.6 = 0.22

根据之前计算过的比率来看，可能的地震事件在点 x_5 和 x_6 处发生。因为点间的时间间隔为 0.01 秒，所以这两个地震事件对应的时间应该为 0.05 秒和 0.06 秒。（假定文件中的第一个点事件发生在 0.0 秒。）

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤：

分解提纲

1）从数据文件中读取地震数据，同时从键盘读入计算功率的测量点数，即两个数据窗口的大小。

2）计算功率比率，打印可能发生的地震事件的时间。

步骤 1）包含了读取数据文件，并将信息存储在数组中。由于该数组的确切大小是未知的，在数组定义时就需要指定一个最大值。接下来还需要从键盘读入功率测量的数据窗口，以便进行功率计算。

[311] 步骤 2）包含了计算功率比率，并将结果同阈值相比较，来确定一个可能的地震事件是否发生。由于每一个可能的地震定位都需要计算两个功率测量值，所以将功率测量过程设计成一个函数。main 函数和 power 函数的伪代码提炼如下所示：

[提炼后的伪代码]

主函数：将阈值置为 1.5
读取 npts 和 time-interval
将数值存入传感器数组中
从键盘读入短窗口、长窗口
将 k 置为长窗口 -1

```
while k <= npts-1
        将短时功率代入函数 power (sensor, short-window, k)
        将长时功率代入函数 power (sensor, long-window, k)
        将 short-power/long-power 值赋给 ratio
        if ratio > 阈值
          打印 k，时间间隔
          k 自增 1

power (x, length, n):
        将 xsquare 置为 0
        将 k 置为 n
        while k>=n-length+1
        将 x[k]·x[k] 加到 xsquare 中
        返回 xsquare/length
```

现在将伪代码转换为 C 程序。

```
/*-----------------------------------------------------------------*/
/*   程序 chapter6_5                                               */
/*                                                                 */
/*   该程序读取一个地震数据文件，并确定可能的地震事件发生的时间    */

#include <stdio.h>
#define FILENAME "seismic1.txt"
#define MAX_SIZE 1000
#define THRESHOLD 1.5

int main(void)
{
   /*  声明变量和函数原型  */
   int k, npts, short_window, long_window;
   double sensor[MAX_SIZE], time_incr, short_power,
          long_power, ratio;
   FILE *file_ptr;
   double power_w(double *ptr,int n);

   /*  读取传感器数据文件  */
   file_ptr = fopen(FILENAME,"r");
   if (file_ptr == NULL)
      printf("Error opening input file. \n");
   else
   {
      fscanf(file_ptr,"%d %lf",&npts,&time_incr);
      if (npts > MAX_SIZE)
         printf("Data file too large for array. \n");
      else
      {
         /*  将数据存入数组  */
         for (k=0; k<=npts-1; k++)
            fscanf(file_ptr,"%lf",&sensor[k]);

         /*  从键盘读入窗口尺寸大小  */
         printf("Enter number of points for short window: \n");
         scanf("%d",&short_window);
         printf("Enter number of points for long window: \n");
```

312

```
            scanf("%d",&long_window);

            /*  计算功率比率并搜索地震事件  */
            for (k=long_window-1; k<=npts-1; k++)
            {
               short_power = power_w(&sensor[k],short_window);
               long_power = power_w(&sensor[k],long_window);
               ratio = short_power/long_power;
               if (ratio > THRESHOLD)
                  printf("Possible event at %f seconds \n",
                         time_incr*k);
            }

            /*  关闭文件  */
            fclose(file_ptr);
         }
      }
      /*  退出程序  */
      return 0;
   }
   /*-----------------------------------------------------------------*/
   /*  该函数计算 double 型数组中一个指定窗口的平均功率                 */

   double power_w(double *ptr, int n)
   {
      /*  声明并初始化变量  */
      int k;
      double xsquare=0;

      /*  计算数组 x 中框中的数值之和  */
      for (k=0; k<=n-1; k++)
         xsquare += *(ptr-k)*(*(ptr-k));

      /*  返回框中数值的平均值  */
      return xsquare/n;
   }
   /*-----------------------------------------------------------------*/
```

[313]

5. 测试

使用手动演算示例中的数据文件作为程序输入，得到的输出结果如下所示：

```
Enter number of points for short-window:
2
Enter number of points for long-window:
5
Possible event at 0.050000 seconds
Possible event at 0.060000 seconds
```

修改

修改上面的地震探测程序，添加一些新功能。

1. 允许用户输入阈值。同时确保输入的值是一个大于 1 的正数。
2. 打印程序探测出的地震事件的次数。（假设时间相邻的事件算作同一地震事件的一部分。因此在手动演算示例中，被探测出的事件次数为 1。）

6.6 字符串

如果一个数组的每个元素都是一个字符变量，那么该数组就是一个字符数组。字符串（character string）就是一个字符数组，数组的最后一个元素为空字符 '\0'，其对应的 ASCII

码为整数0。本节将集中讨论字符串。

6.6.1 字符串定义与输入/输出

字符串常量是包含在双引号中的，比如“sensor1.txt”、“r”和“15762”。字符数组的初始化可以使用字符串常量，或者字符常数来完成，下面的三条语句是等价的：

```
char filename[12] = "sensor1.txt";
char filename[] = "sensor1.txt";
char filename[] = {'s','e','n','s','o','r',
                   '1','.','t','x','t','\0'};
```

为了从键盘读入一行数据，并将其存储为一个字符串，可以使用下列语句实现：

```
/*  声明变量  */
int k=0, nchars;
char line[50];
...
/*  将字符存储为字符串，直到遇到换行符为止  */
while ((line[k]=getchar()) != '\n')
   k++;
line[k] = '\0';
nchars = k + 1;
```

314

从数据文件中读取一行字符串可以用类似的语句实现。如果要打印字符串，可以使用下面的语句：

```
for (k=0; k<=nchars-2; k++)
   putchar(line[k]);
putchar('\n');
```

要注意的是，打印字符串并不包括最后一个字符，即line[nchar-1]，因为这是一个空字符；然而，为了使打印结果显示在单独的一行，同其他信息区分开，还需要打印一个换行符。打印字符串也可以使用%s说明符，它不会打印空字符。这样一来，字符串line便可以通过下面的语句来打印：

```
printf("String: %s \n",line);
```

如果要打印的带有%s说明符的字符串没有以空字符作结尾，那么该字符串后面的字符会持续被打印，直到遇到一个空字符为止。

6.6.2 字符串函数

首先我们给出一个用字符串作参数的自定义函数，随后再介绍一组使用字符串的库函数。下面的自定义函数是以字符串作为参数，来确定字符串长度，而字符串的长度是由其中字符的个数决定的，结尾的空字符不计在内。该函数的原型语句如下所示：

```
/*-------------------------------------------------------------*/
/*  该函数确定一个字符串的长度                                 */
int strg_len_1(char s[])
{
   /*  声明变量  */
   int k=0;
   /*  计算字符个数  */
   while (s[k] != '\0')
      k++;
```

```
    /*  返回字符串长度  */
    return k;
}
/*-----------------------------------------------------------------*/
```

标准C语言库包含了一系列字符串处理函数。为了后文讨论这些函数和参数方便，假定s和t就代表待处理的字符串；变量n的数据类型是size_t，即无符号整数；c是要被转换成字符的整型变量。还要注意的是，这其中有一些函数返回的是字符串指针。下面这些函数的原型语句都包含在了头文件string.h中。

315

strlen(s)	该函数返回字符串s的长度。
strcpy(s,t)	该函数将字符串t复制到字符串s中。函数返回值为字符串s的指针。
strncpy(s,t,n)	该函数将字符串t中最多n个字符复制到字符串s中。如果t的字符个数比s少，那么字符串s中剩余部分用空字符填充。函数返回值为字符串s的指针。
strcat(s,t)	该函数将字符串t连接（concatenates）在字符串s的末端。这样字符串s将会包含原有的字符和字符串t中的字符；并且t中的第一个字符将会覆盖掉s末端的空字符。函数返回值为字符串s的指针。
strncat(s,t,n)	该函数将字符串t中最多n个字符连接到字符串s的末端。如果t中字符个数大于n，那么只将t的头n个字符连接到s中。而t中的第一个字符会覆盖掉s末尾的空字符；同时在合并后的字符串s的末端加上一个空字符。函数返回值为字符串s的指针。
strcmp(s,t)	该函数将字符串s与字符串t逐字比较，从s[0]和t[0]的比较开始。如果s<t，返回一个负值；如果s=t，返回0；如果s>t，则返回正值。
strncmp(s,t,n)	该函数将字符串s中最多n个字符与字符串t逐字比较，从s[0]和t[0]的比较开始。如果s<t，返回一个负值；如果s=t，返回0；如果s>t，则返回正值。
strchr(s,c)	该函数返回在字符串s中第一个出现的字符c的指针。如果在s中不包含该字符，则返回空指针NULL。
strrchr(s,c)	该函数返回在字符串s中最后出现的字符c的指针。如果在s中不包含该字符，则返回空指针NULL。
strstr(s,t)	该函数返回在字符串s中出现的字符串t的头指针。如果在s中不包含t，则返回空指针NULL。
strspn(s,t)	该函数返回字符串s中从头开始连续包含在字符串t中的字符个数。（即如果返回值为n，则说明s的前n个字符包含在t中的）
strcspn(s,t)	该函数返回字符串s中开头没有包含在字符串t中的连续字符个数。（即如果返回值为n，则说明s的前n个字符都没有在t中出现）
strpbrk(s,t)	该函数返回在字符串s中第一次出现的字符串t中的字符的位置指针。如果在s中未包含任何t中的字符，则返回值为空指针NULL。

为了说明上述函数的用法，现给出一个简单的例子：

```
/*-----------------------------------------------------------------*/
/*  程序chapter6_6                                                 */
/*                                                                 */
/*  该程序说明了字符串函数的用法                                   */

#include <stdio.h>
```

```
#include <string.h>

int main(void)
{
   /*  声明和初始化变量  */
   char strg1[]="Engineering Problem Solving: ";
   char strg2[]="Fundamental Concepts", strg3[50];
   /*  打印字符串的长度  */
   printf("String lengths: %d %d \n",
          strlen(strg1),strlen(strg2));

   /*  将两个字符串联结成一个  */
   strcpy(strg3,strg1);
   strcat(strg3,strg2);
   printf("strg3: %s \n",strg3);
   printf("strg3 length: %d \n",strlen(strg3));

   /*  退出程序  */
   return 0;

}
/*-----------------------------------------------------------*/
```

316

该程序的输出结果如下所示：

```
String lengths: 29 20
strg3: Engineering Problem Solving: Fundamental Concepts
strg3 length: 49
```

在很多工程应用中都会用到字符串，包括密码学和模式识别。通过指针来进行字符串操作是非常便利的。前面讨论过的许多字符串函数都要求用字符指针作参数，很多函数的返回值也是字符指针。现在，来看一看通过指针引用字符串的语法问题。

在本节前面的内容里，设计了一个用户自定义函数，通过在 while 循环中使用下标来确定字符串的长度。现在使用指针重写这个函数：

```
/*-----------------------------------------------------------*/
/*  该函数通过在 while 循环中使用指针，来确定一个字符串的长度          */

int strg_len_2(char *s)
{
   /*  声明变量  */
   int count=0;

   /*  计算字符个数  */
   while (*s != '\0')
   {

      count++;
      s++;

   }
   /*  返回字符串长度  */
   return count;
}
/*-----------------------------------------------------------*/
```

317

先前介绍的函数会计算字符串中的每一个字符，直到找到空字符为止。标准 C 语言库中包含一个名为 strlen 的函数，可以返回一个字符串的长度；因此与其自己编写函

数，不如直接使用 strlen 函数。在标准C语言库中还有个叫作 strstr 的函数。这个函数 strstr(s,t) 是以字符串 s 的指针和字符串 t 的指针作为函数参数，并且返回在字符串 s 中出现的字符串 t 的头指针。如果在字符串 s 中没有出现字符串 t，则返回一个 NULL 指针。在下一节中将会用到该函数。

6.7 解决应用问题：DNA 测序

在本节中，将使用本章新介绍的函数语句来解决关于DNA测序的问题。在本章开头的讨论中已经介绍过，DNA 图谱就是一个字符串序列，它由一系列代表着核苷酸顺序的字母组成，如 AAACGTACAGGT。一个未知 DNA 序列和数据库中 DNA 序列的匹配过程实际上是基于两个链之间的一些短链的匹配工作（即短串联重复，Short Tandem Repeats，STR）。

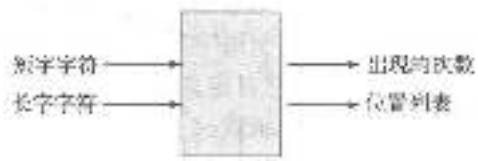
本节将会计算一个短字符串在长字符串中出现的次数，并打印出这些字符串匹配的相对位置。前面关于字符串函数的讨论将为问题的解决提供基础。

1. 问题陈述

给定一个长字符串和一个短字符串，找出短串在长串中出现的次数，并打印每次出现的短字符串开头的位置。

2. 输入/输出描述

下图展示了该程序的输入是一个长字符串和一个短字符串。程序输出为每次在长串中出现的短串开头的位置，以及出现的次数。



3. 手动演算示例

假设长字符串是 AAACTGACATTGGACCTACTTTGACT，短字符串是 ACT。那么在长字符串中，短字符串一共出现了三次，并且分别是在位置 3、18、24 上出现的。

318

4. 算法设计

在解决问题时将会使用在前面小节里讨论过的函数 strstr。该函数返回在长字符串中出现的短字符串的头指针。如果在长串中不包含短串，则返回空指针 NULL。为了找到下一个出现的短字符串，需要从上一个短字符串出现的位置开始继续搜索。随后函数 strstr 将会返回下一个在长字符串中出现的短字符串的头指针。这个过程将会一直持续直到 strstr 函数返回一个 Null 值为止。

分解提纲

1）将长字符串和短字符串初始化。

2）计算并打印短字符串在长字符串中出现的位置及次数。

步骤2）中包含了一个循环结构。在循环中会持续计算短字符串出现的次数，直到搜索到长字符串的末尾。下面是提炼后的伪代码：

[设计后的伪代码]

主函数：将长字符串和短字符串初始化

将 count 置为 0

将指针 ptr1 置为长字符串的开头

将指针 ptr2 置为短字符串的开头

while 没有到达长字符串的末尾

if 短字符串在剩余的长字符串中出现

count 自增 1

打印出现的位置

长字符串的指针自增

打印 count

现在将伪代码转化成 C 程序。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter6_7                                                */
/*                                                                 */
/*  该程序初始化一个长字符串和一个短字符串。然后打印短字符串在长字符串中出  */
/*  现的位置。同时程序还打印短字符串在长字符串中出现的次数          */

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

int main(void)
{
   /*  声明和初始化变量  */
   int count=0;
   char long_str[]="AAACTCACATTGGACCTACTTTCACT",
        short_str[]="ACT";
   char *ptr1=long_str, *ptr2=short_str;
   /*  计算 short_str 在 long_str 中出现的字数  */
   /*  若函数 strstr 没有返回 NULL 时，count 自增 1，并且将指针 ptr1 移动到长字 */
   /*  符串的下一个字符                                              */
   while ((ptr1=strstr(ptr1,ptr2)) != NULL)
   {
      printf("location %i \n",ptr1-long_str+1);
      count++;
      ptr1++;
   }

   /*  打印短字符串出现的次数  */
   printf("number of occurrences: %i \n",count);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

319

5. 测试

首先用手动演算示例中的数据来测试程序。程序执行的结果如下所示：

```
location 3
location 18
location 24
number of occurrences: 3
```

程序的执行结果同手动演算示例中的结果一致，所以下面可以再增加其他数据来测试程序。

修改

下面的问题是由本节开发的程序延伸而来。

1. 修改程序，打印出长字符串和短字符串。
2. 修改程序，允许用户自行输入长字符串和短字符串。
3. 修改程序，使得长字符串是从一个数据文件中读取，短字符串通过用户输入。
4. 将程序修改为专门处理大写字母或小写字母。
5. 修改程序，检查两个字符串长度，以确保短字符串的长度小于长字符串。

320

*6.8 动态内存分配

一般的内存分配工作是在程序编译时进行的。而动态内存分配（dynamic memory allocation）允许程序员在程序执行过程中进行内存分配。当程序中使用数组，但又无法事先确定数组大小的时候，使用动态内存分配就尤为重要。否则用户就必须要指定数组的最大长度。对于内存有限的系统来说，如果将程序中的所有数组都指定最大长度，那么在程序执行过程中就很可能会出现内存不足的情况。

要实现动态内存分配，应该使用 malloc 函数或 calloc 函数，在程序执行过程中实现"内存分配"或"干净的内存分配"的操作。两个函数都能在系统中预留出一段连续的内存空间；此外，calloc 函数还会将已分配的内存初始化为二进制零。函数 malloc 的参数为所需内存的字节数（一个字节（byte）就是一个包含 8 bits 的内存单元。）函数 calloc 有两个参数，一个是所需存储单元的数量，另一个是每一个存储单元包含的字节数。下面给出两个函数的原型语句（包含在头文件 <stdlib.h> 中），同时对函数参数和函数返回值进行深入阐释。

```
void *malloc(size_t n);
void *calloc(size_t n,size_t size);
```

由于存储一个特定数据类型（比如 int 型）的数值所需的字节数是系统相关的，因此就需要使用操作符 sizeof 来确定该数据类型所需的字节数。表达式 sizeof(int) 表示存储一个整型值所需的字节数，表达式 sizeof(double) 就表示存储一个 double 型值所需的字节数。同时，sizeof 操作符返回一个无符号整数，也就是 size_t 类型值。其中 size_t 是依赖于编译系统的，但是通常为 unsigned int 或 unsigned long。因此，要为 200 个整数分配内存，下面两组语句都可以实现：

```
num_pts = 200;
int *p;
p = (int *)malloc(num_pts*sizeof(int));

num_pts = 200;
int *p;
p = (int *)calloc(num_pts,sizeof(int));
```

两个函数都返回一个指针值。如果有可用内存，则指针中包含已分配内存的地址；如果未能分配成功，则指针包含一个 NULL 值。函数返回的指针被称为 void 指针（void pointer），这是因为存储在内存单元里的变量类型没有被指定。因此，对于该函数的返回值应该使用一个强制转换符，以得到一个合适的指针类型。

321

为了进一步说明函数的用法，假设现在需要动态分配内存，来存储一个 double 型的数

组，数组名为 x，拥有 npts 个元素。（在本例中，直接给出了 npts 的值，但是同样也可以通过其他语句来计算出数组元素个数，当然也可以从键盘输入或数据文件中读取到 npts 的值。）内存动态分配的过程可以由下面两组语句实现：

方法 1

```
/*  声明变量  */
int npts = 500;
double *x;
...
/*  动态分配内存  */
x = (double *)malloc(npts*sizeof(double));
```

方法 2

```
/*  声明变量  */
int npts = 500;
double *x;
...
/*  动态分配内存  */
x = (double *)calloc(npts,sizeof(double));
```

如果使用的是函数 calloc，那么被动态分配的内存也会被初始化为 0。不论使用哪种方法，都应该将 x 值同常量 NULL 作比较，以确认内存是否被分配成功。具体做法如下所示：

```
if (x == NULL)
   printf("Memory requested not available. \n");
```

在确定了内存已经分配成功后，在程序中引用数组 x，如 x[k]，便是合法引用了。

如果要将被动态分配的内存释放掉，可以使用 free 函数，它的函数原型为：

```
void free(void *ptr);
```

这样一来，被分配给数组 x 的内存空间就可以通过下面的语句被释放：

```
free(x);
```

通常，当被动态分配的内存不再使用时，就应该将其释放。这样内存就可以被其他动态分配继续使用。

函数 realloc 可以改变被函数 calloc、malloc 或先前执行的 realloc 函数动态分配的内存大小。它的原型语句为

```
void *realloc(void *ptr,size_t size);
```

如果 ptr 包含一个 NULL 值，则该函数的用法同 malloc 一致。如果 ptr 包含了在程序中 calloc，malloc 或者先前执行的 realloc 函数的返回值，那么相应的动态分配的内存大小就会被更改为新的 size 值。如果新的 size 值比原来的内存空间要大，那么新分配的内存空间值是未知的。如果新的 size 值比原来要小，则新的 size 值保持不变。如果额外的内存空间不可用，则原内存空间不变，同时函数返回一个 NULL 值。 322

通过使用动态内存分配和动态内存释放，就可以在程序执行期间的任何时候随意分配或小的内存空间。对于同时执行多个程序的系统来说，动态内存分配及释放的使用能够允许更多的程序同时运行。

下面的程序可以获得执行期间动态内存分配能够获得的连续内存空间的最大值。最大可用内存空间的值一般与正在使用系统的其他用户以及存储在系统中的其他软件有关。因此，

不同的系统、不同的时间来执行程序可能会得到不同的结果。下面就是程序代码：

```
/*-----------------------------------------------------------------*/
/*  程序 chapter6_8                                                */
/*                                                                 */
/*  该程序能够计算出在一次程序执行时通过动态分配获得的连续内存空间的最大值  */

#include <stdio.h>
#include <stdlib.h>
#define UNIT 1000000

int main(void)
{
   /*  声明和初始化变量  */
   int k=1, *ptr;

   /*  找到可用的最大连续内存空间  */
   /*  以1000000为单位  */
   ptr = (int *)malloc(UNIT*sizeof(int));
   while (ptr != NULL)
   {
      free(ptr);
      k++;
      ptr = (int *)malloc(k*UNIT*sizeof(int));
   }

   /*  打印可用的最大内存空间  */
   printf("Maximum contiguous memory available: \n");
   printf("%k integers \n",(k-1)*UNIT);

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
```

使用个人计算机执行该程序，可以得到如下典型的运行结果：

```
Maximum contiguous memory available:
31000000 integers
```

323

修改

修改程序 chapter6_8，计算出使用下列数据类型申请内存时，能够获得的最大连续内存中包含的变量的个数，结果以千计算即可：

1. 长整型　　2. 双精度型　　3. 长双精度型

在前面的章节中已经开发了地震预测程序：从数据文件中读取一组地震探测数据，然后从这些数据中找到可能发生的地震事件的时间点。此程序使用了符号常量 MAX_SIZE 来分配数组，存储数据。如果数据文件中的数据量超过了 MAX_SIZE，那么程序会中断，并返回错误信息。而在下面的程序中使用了动态内存分配，使得程序不必在编译阶段分配数组的最大长度（相反，数组的内存空间会根据地震探测数据文件的大小来动态分配）：

```
/*-----------------------------------------------------------------*/
/*  程序 chapter6_5mod                                             */
/*                                                                 */
/*  该程序读取地震探测数据文件，并确定可能发生的地震事件的次数。程序中  */
/*  使用了动态内存分配                                             */

#include <stdio.h>
```

```
#define FILENAME "seismic1.txt"
#define THRESHOLD 1.5

int main(void)
{
   /*  声明变量和函数原型  */
   int k, npts, short_window, long_window;
   double *sensor, time_incr, short_power,
          long_power, ratio;
   FILE *file_ptr;
   double power_w(double *ptr,int n);

   /*  读取数据头，并进行内存分配  */
   file_ptr = fopen(FILENAME,"r");
   if (file_ptr == NULL)
      printf("Error opening input file. \n");
   else
   {
      fscanf(file_ptr,"%d %lf",&npts,&time_incr);
      sensor = (double *)malloc(npts*sizeof(double));
      if (sensor == NULL)
         printf("Not enough memory available. \n");
      else
   /*  其余部分同程序 chapter6_5 完全一致  */
```

324

*6.9 快速排序算法

本节中会通过以指针作为参数的递归函数（同既 4.8 节内容）来实现快速排序（quicksort）算法。

快速排序算法首先选择一个枢轴值（pivot value），然后将剩下的数值分为两组——一组值小于枢轴值，一组值大于枢轴值。序列中的第一个元素和位于中间点的元素都常常被选作枢轴值。一旦分组完成，枢轴值在序列中的正确位置就已经确定，位于两组值的中间。由于划分出的两组中的值还没有正确排序，所以需要重复上述操作。选择其中较小的一组，并在其中选取新的枢轴值，该组值又被分为新的两组——一组小于新枢轴值，一组大于新枢轴值。这个过程会一直持续下去，直到最终得到一个足够小的分组，组中没有包含值、有一个值或者有两个值。如果仅包含两个值，需要的话可以交换两个值的位置就可以完成排序。这样一来，在原始序列中，小于初始枢轴值的这半个序列顺序完全正确。接下来，对于原始序列中大于初始枢轴值的另半个序列，继续重复上述过程。当该部分序列顺序正确，则整个序列排序完成。该算法可以用递归的方式去描述，因为算法的每一步都是相似的过程，只是分组更小。当分组只有两个及以下的数值时，算法停止。下面是一组手动演算示例：

原始列表：

4　　10　　3　　6　　−1　　0　　2　　5

分为两组，分别小于枢轴值和大于枢轴值：

[3　　−1　　0　　2]　　4　　[10　　6　　5]

其中的每一组都有一个枢轴值，将每个组都分为小于枢轴值和大于枢轴值的两个更小的组：

[−1　　0　　2]　　3　　4　　[6　　5]　　10

其中的每一组都有一个枢轴值，将每个组都分为小于枢轴值和大于枢轴值的两个更小的组：

−1　　[0　　2]　　3　　4　　5　　6　　10

其中的每一组都有一个枢轴值，将每个组都分为小于枢轴值和大于枢轴值的两个更小的组：

-1　　0　　2　　3　　4　　5　　6　　10

在快速排序算法的实现过程中，引用了一个函数 separate，该函数能够改变数组中的元素位置，使得枢轴值正确放置于 break_pt 所表示的位置上。所有比枢轴值小的元素排在数组的左侧（如果将数组看作一个从左到右的横行），所有比枢轴值大的元素排在右侧。然后通过一条语句来递归调用函数 quicksort，该条语句指定了排序过程需要处理的是从元素 x[0] 开始的 break_pt 个数值：

325

```
quicksort(x, break_pt);
```

函数 quicksort 同时要执行另一条语句递归调用，该语句指定了排序过程需要处理的是从元素 x[break_pt+1] 开始的 n-break_pt-1 个数值：

```
quicksort(&x[break_pt+1],n-break_pt-1);
```

这里要注意，函数实参使用的是 &x[break_pt+1]，也就是对元素 x[break_pt+1] 地址的引用。这里采用地址引用是必要的，因为使用 x[break_pt+1] 发生的是传值调用，而不是传址调用。quicksort 函数和 separate 函数使用了 6.2 节中设计的 switch2 函数。由于快速排序算法较为复杂，因此最好使用示例中的数据结合下面语句手动演算排序的过程：

```
/*-----------------------------------------------------------*/
/*   程序 chapter6_9                                         */
/*                                                           */
/*   该程序测试 quicksort 函数                               */

#include <stdio.h>

int main(void)
{
   /*  声明和初始化变量  */
   int x[8]={4,10,3,6,-1,0,2,6}, npts=8,  k;
   void quicksort(int w[],int n);
   int separate(int y[],int m);
   void switch2(int *a,int *b);

   /*  排序并输出数组  */
   printf("Before: ");
   for (k=0; k<=7; k++)
      printf("%d ",x[k]);
   printf("\n");
   quicksort(x,npts);
   printf("After: ");
   for (k=0; k<=7; k++)
      printf("%d ",x[k]);
   printf("\n");

   /*  退出程序  */
   return 0;

}
/*-----------------------------------------------------------*/
/*   该函数实现快速排序算法                                  */

void quicksort(int w[],int n)
{
```

```
    /*  声明变量和函数原型  */
    int break_pt;
    int separate(int y[],int m);
    void switch2(int *a,int *b);
    /*  如果仅有两个元素，将其正确排序  */
    if (n == 2)
    {
       if (w[0] > w[1])
          switch2(&w[0],&w[1]);
    }
    else
    /*  如果多于两个元素，将其分为大于枢轴值和小于枢轴值的两组  */
       if (n > 2)
       {
          break_pt = separate(w,n);
          quicksort(w,break_pt);
          quicksort(&w[break_pt+1],n-break_pt-1);
       }
    /*  返回值为空  */
    return;
}
/*-----------------------------------------------------------------*/
/*  该函数对数组重新排序，初始数组中 y[0] 处于正确的位置，同时 y[0] 大于它  */
/*  左侧的数值，并小于它右侧的数值                                  */

int separate(int y[],int m)
{
    /*  声明变量和函数原型  */
    int k1=1, k2=1, count=0, pivot;
    void switch2(int *a,int *b);

    /*  将数组分为两组  */
    pivot = y[0];
    while (k1<m && k2<m)
    {
       while ((k1<m) && (y[k1]>pivot))
          k1++;
       while ((k2<m) && (y[k2]<pivot))
          k2--;
       if ((k1<m) && (k2<m))
       {
          switch2(&y[k1],&y[k2]);
          count++;
       }
    }
    /*  将枢轴值放于正确位置  */
    if (count > 0)
       switch2(&y[0],&y[count]);
    else
    {
       k1 = 0;
```

326

327

```
        while ((k1<m-1) && (y[k1]>y[k1+1]))
        {
            switch2(&y[k1],&y[k1+1]);
            k1++;
        }
        count = k1;
    }
    /*  返回count值  */
    return count;
}
/*-------------------------------------------------------------*/
/*  两个变量值进行互换的正确函数实现                              */
void switch2(int *a,int *b)
{
    /*  声明变量  */
    int hold;
    /*  交换a和b的值  */
    hold = *a;
    *a = *b;
    *b = hold;
    /*  无返回值  */
    return;
}
/*-------------------------------------------------------------*/
```

本章小结

本章主要讲述了指针和指向的变量之间的关系。首先给出样例来说明如何定义和初始化指针，并列举出能够参与指针运算的运算符，同时还更新了包含地址运算符和间接引用运算符在内的运算符优先级列表。用示例程序说明了将指针用作函数参数和通过指针引用数组的情况。除此之外，还给出了一组操作字符串的函数，其中的多数函数都使用指针作为函数参数。最后，本章还介绍了一些C语句，可以通过使用指针来实现动态内存分配。

328

关键术语

address（地址）
address operator（取地址运算符）
byte（字节）
character string（字符串）
concatenate（串联）
dereference（间接引用）
dynamic memory allocation（动态内存分配）
indirection（间接引用）
NULL character（空字符）
offset（偏移量）
pointer（指针）
void pointer（空指针）

C语句总结

指针声明：

```
int *ptr_1;
double a, *ptr_2=&a;
```

动态内存分配：

```
x = (double *)malloc(npts*sizeof(double));
x = (double *)calloc(npts,sizeof(double));
```

注意事项

1. 要选择合适的指针变量名，能够清晰地表达指针与相应变量的关联关系。
2. 如果初始化时没给指针分配内存地址，则为其赋初值为 NULL 来表明该指针无内存分配。

调试注意事项

1. 确保变量在使用之前已经被初始化。
2. 确保指针变量在引用地址之前已经被初始化。
3. 函数中的指针参数对应的实参必须是地址或指针。

习题

简述题

判断题

判断下列语句的正（T）误（F）。

1. 取址运算符和间接运算符都是单目运算符。 T F
2. 指针能够间接访问一个指定数据项的值。 T F
3. 在指针指向一个变量之前，该变量必须已经被声明和初始化。 T F
4. 在程序编译时，动态存储空间的内存地址是确定的。 T F 329

多选题

选出下列题目的最佳答案。

5. 内存中的一块存储空间是（　　）。
   (a) 变量声明时被分配　　(b) 变量在程序中使用时被分配
   (c) 可以同时保存几个不同的值　　(d) 一旦被赋值就不能被再次使用
6. 指针变量（　　）。
   (a) 包含了存储在内存地址的数据
   (b) 包含了内存地址
   (c) 可以用于输入语句，但不能用于输出语句
   (d) 在输入输出语句中都可以被修改为不同的值
7. 如何用指针 a 所指向的变量的值为 name 赋值？（　　）
   (a) a = &name ;　　(b) name = &a ;
   (c) a = *name ;　　(d) name = *a ;
8. 假设 a 和 b 都是整型指针，并且 a 指向变量 name，对语句“b=a;”的运行效果描述最贴切的是（　　）。
   (a) 将 name 的值复制到了 b 中　　(b) 将存储在 a 中的内存地址复制到了 b 中
   (c) 将存储在 b 中的内存地址复制到了 a 中　　(d) 指针 a 现在指向了另一个变量

内存快照题

9. 假设 name 的地址是 10，x 的地址是 14（指的是 name 存储在内存单元 10，x 存储在内存单元 14），给出下列语句执行后对应变量的内存快照：

```
float name, x=20.5;
float *a = &x;
...
name = *a;
```

程序分析题

根据下面的语句回答 10 – 13 题：

```
int i1, i2;
int *p1, *p2;
...
i1 = 5;
p1 =  &i1;
i2 = *p1/2 + 10;
p2 = p1;
```

10. i1 的值是什么？

330

11. i2 的值是什么？

12. *p1 的值是什么？

13. *p2 的值是什么？

编程题

一般函数。当需要函数具有多个返回值时，通常使用指针作为函数参数。对于下列每个问题，按照要求编写函数，然后设计 main 函数来进行测试：

14. 编写函数，将一个圆的半径、直径和面积测量值的单位从英寸和平方英寸转化为英尺和平方英尺。假设相应的函数原型语句为：

```
void convert_ft(double *r,double *d,double *a);
```

其中，r、d 和 a 是分别指向变量 radius、diameter 和 area 的指针。

15. 编写函数，按升序重新排列三个整型变量。假设相应函数原型语句为：

```
void reorder(int *a,int *b,int *c);
```

其中 a、b、c 是分别指向三个变量的指针。

16. 编写函数，确定一个一维整型数组的最大值和最小值。假设相应的函数原型语句为：

```
void ranges(int x[],int npts,int *max_ptr,
            int *min_ptr);
```

其中 npts 是数组 x 中的元素个数，max_ptr 和 min_ptr 是分别指向数组最大值和最小值的指针。

17. 编写函数，返回一个一维整型数组的平均值，且返回值数据类型为 double。此外，还要确定数组中大于平均值的元素个数。假设相应的函数原型语句为：

```
double average(int x[],int npts,int *gtr);
```

其中，npts 是数组 x 中的元素个数，而指针 gtr 指向的变量用于存储 x 中大于平均值的元素个数。

18. 编写函数，返回一个整型数组中正数、零和负数的元素个数。假设相应的函数原型语句为：

```
void signs(int x[],int npts,int *npos,
           int *nzero,int *nneg);
```

其中，npts 是数组 x 中的元素个数，指针 npos、nzero 和 nneg 分别指向存储正数、0 和负数元素个数的变量。

向量函数。向量是用一维数组表示的一组数值。下列问题需要设计函数来操作向量中的数值。然后通过函数调用实现对向量中部分元素的计算操作，而不是一次性操作整组向量元素。在函数中不能使用下标操作，而要使用指针引用。假设数组中第一个位置是下标 0 所引用的位置。 [331]

19. 编写函数，将 0 赋值给向量中的元素。假设函数原型语句为：

```
void zeros(int x[],int n);
```

其中 x 是包含 n 个元素的一维数组。通过调用该函数，将数组 a 中位置 20 ~ 25 的元素赋值为 0。

20. 编写函数，将 1 赋值给向量中的元素。假设函数原型语句为：

```
void ones(int x[],int n);
```

其中 x 是包含 n 个元素的一维数组。通过调用该函数，将数组 y 中位置 5 到 k 的元素赋值为 1。

21. 编写函数，计算向量的元素之和。假设函数原型语句为：

```
int v_sum(int x[],int n);
```

其中 x 是包含 n 个元素的一维数组。通过调用该函数，计算数组 y 中最后 10 个元素的和，数组 y 包含 npts 个元素。

22. 编写函数，将向量中的元素重新倒序排列。假设函数原型语句为：

```
void v_rev(int x[],int n);
```

其中 x 是包含 n 个元素的一维数组。通过调用该函数，将数组 y 中位置 5 ~ 20 的元素重新倒序排列。

23. 编写函数，将数组中的值全部替换为相应的绝对值。假设函数原型语句为：

```
void v_abs(int x[],int n);
```

其中 x 是包含 n 个元素的一维数组。通过调用该函数，将数组 t 中的前 5 个元素以外的所有元素全部替换为相应的绝对值，数组 t 包含 npts 个元素。

字符函数。在许多工程领域问题的解决方案中，需要搜索信号中的特定模式信息。为此，下面将设计一组函数来解决相关问题。

24. 编写函数，参数为一个指向字符串的指针和一个字符。函数返回该字符在字符串中出现的次数。假设以下为函数原型语句：

```
int charcnt(char *ptr,char c);
```

25. 编写函数，参数为一个指向字符串的指针，函数返回字符串中出现的重叠字符个数。例如，字符串“Mississippi”含有一组重复字符（ss, ss, pp）。重复出现的字格不计算在内。如果重叠字符的连续出现次数超过 2 次，只计为一组重复字符。因此，字符串“hisssss”只含有一组重复字符。假设以下为函数原型语句： [332]

```
int repeat(char *ptr);
```

26. 重新编写 25 题中的函数，每一个字符当与它前面的字符相同时，都计为一组重复字符。因此，字符串“hisssss”含有 4 组重复字符。假设以下为函数原型语句：

```
int repeat2(char *ptr);
```

27. 编写函数，参数为指向两个字符串的指针，函数返回第二个字符串在第一个字符串中出现的次数，其中不计算重叠出现次数。因此，“101”在字符串“110101”仅出现一次。假设以下为函数原型语句：

```
int pattern(char *ptr1,char *ptr2);
```

28. 重新编写27题中的函数，还是返回第二个字符串在第一个字符串中出现的次数，但此时允许计算重叠出现次数。因此，"101"在字符串"110101"出现两次。假设以下为函数原型语句：

```
int overlap(char *ptr1,char *ptr2);
```

表函数。为下列问题设计一组函数，来计算二维数组或数据表中的数据值。在函数中要使用指针引用而不是下标引用。

29. 编写函数，在一个10行8列的表格中，计算第i行的数据和。假设以下为函数原型语句：

```
double row_sum(double table[10][8],int i);
```

30. 编写函数，在一个10行8列的表格中，计算第j列的数据和。假设以下为函数原型语句：

333

```
double col_sum(double table[10][8],int j);
```

利用结构体编程

犯罪现场调查：指纹识别

指纹识别是最古老的一种生物识别技术。通过指纹来识别身份最早可以追溯到17世纪意大利的博洛尼亚大学。在19世纪末，一群英国科学家发现指纹上的漩涡图案对于身份识别具有重大作用，于是在全世界范围内掀起了建立指纹系统数据库的工作浪潮。美国联邦调查局的自动指纹识别系统（Automatic Fingerprint Identification System，AFIS）涵盖了超过2亿个指纹数据。全美的执法人员都有权访问并查询指纹数据库。在伊拉克执行重要任务的美国士兵可以采集恐怖分子/犯罪嫌疑人的指纹，通过卫星信号将指纹信息传输到西弗吉尼亚州的AFIS数据库，在15分钟内就能得到一份指纹匹配报告。指纹信息可以取自单一一根手指（通常是食指），也可以来自全握的十根手指。指纹信息的获取方式分为很多种，其中，"掌拍"可在同一时间获取多个指纹，而指印则是在犯罪现场发现的指纹信息，并且通常情况下是不完整的，需要通过化学途径或其他替代光源来进行指纹恢复。对于指纹匹配，则是匹配指纹样的结构，包括环、漩涡和拱等形状。如果这些全局结构匹配上了，那么接下来就要分析指纹中单独的点（也叫作细节点）来做进一步匹配。这些细节点包括弯分岔、弯网络末梢和奇点等形态。在7.3节会设计一个C函数用来辅助指纹分析。

学习目标

在本章，我们将学到以下解决问题的方法：

- 在main函数中使用结构体。
- 结构体在函数中的使用。
- 结构数组。
- 动态数据结构。

7.1 结构体

在解决工程问题时，常常需要处理大批量的数据。第5章介绍了如何使用数组来存储和处理大型数据集。但数组本身是有局限性的，只有在所有数据类型相同的情况下才能使用数组。更多情况下，需要处理的数据是一个对象或是具有多个数据类型的一组信息。例如，在第5章讨论飓风问题的工程实践中，飓风本身有一个名称，并且根据风力强度来区分等级。

那么在表示飓风信息时，可能需要包括飓风的名称、出现的年份以及它的强度等级。其中，名称可以用一个字符串来表示，年份和强度可以用整型变量表示。结构体（structure）就是这样一种可以包含一组数据的数据类型，并且结构体中每种数据可以具有不同类型。因此，飓风信息可以使用一个结构体来表示成如下形式：

```
struct hurricane
{
   char name[10];
   int year, category;
};
```

现在便可以定义结构体 hurricane 类型的变量，甚至是数组。每个该类型的变量或数组元素都包含三个数据值——一个字符串和两个整数。

结构体通常被称为聚合型数据类型，因为它可以将多个数据值聚合成一个单独的数据类型。结构体中的每个数据值叫作数据成员（data member），每个数据成员都有一个名称。在前面的例子中，hurricane 结构体的数据成员名称分别是 name、year 和 category。对数据成员的引用可以通过结构体变量名+英文句号（称作结构体成员运算符（structure member operator））+数据成员名称的形式来实现。要注意区分引用结构体中的数值和引用数组中的数值之间的不同：数组值是通过数组名和下标来引用的。

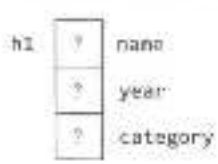
335

7.1.1 定义和初始化

要在程序中使用结构体，首先要定义结构体。在C语言中，结构体名称（也叫作结构体标签（tag））和结构体中包含的数据成员要通过关键字 struct 来定义。定义了结构体，接下来便可以使用声明语句来声明结构体变量。例如，前面对结构体 hurricane 的定义中，结构体名称是 hurricane，三个数据成员是 name，year 和 category，最后在结构体定义结束后要以分号作为结尾。结构体可以定义在 main 函数之前，通常将结构体的定义保存在头文件里。需要特别注意的是，上面提到的这些语句并不会分配内存——它们只是定义了一个结构体。如果要定义该结构类型的变量，可以在程序的声明语句部分通过如下语句实现：

```
struct hurricane h1;
```

这条声明语句定义了一个名为 h1 的变量，该变量拥有三个数据成员，如下图所示：



上面的声明语句为三个数据成员分配了内存，但并没有分配初始化值，所以数据成员的值此时是未知的。

结构体数据成员的初始化既可以在结构体声明语句中进行，也可以在主程序中实现。如果要在声明语句中初始化结构体，就需要用一对大括号来给结构体设置初始值，顺序为每一个成员变量赋值，并用逗号隔开。下面的声明语句就以结构体变量 h1 为例进行定义和初始化：

```
struct hurricane h1={"Camille",1969,5};
```

语句执行后，h1 中的数据成员便完成了初始化：

h1	'Camille'	name
	1969	year
	5	category

如果在主程序中完成结构体 h1 的初始化，可以通过下列语句实现：

```
h1.name = "Camille";
h1.year = 1969;
h1.category = 5;
```

336

如果要引用某个数据成员，便可以使用“结构体变量名称.数据成员名称”的形式实现。

练习

考虑下面的结构体：

```
struct hurricane
{
   char name[10];
   int year, category;
};
```

在下列语句执行后，写出每一个结构体变量的数据成员的值。

```
1. struct hurricane h1={"Andrew",1969,5};
2. struct hurricane h2;
3. struct hurricane h3;
   ...
   h3.name = "Hugo";
```

7.1.2 输入和输出

与前面学习的变量使用的规则类似，可以使用 scanf 和 fscanf 语句将数据读入结构体的数据成员中，也可以使用 printf 和 fprintf 来打印数据成员的值。只是这时变量时，一定要使用成员运算符来指定结构体的数据成员。在下面的程序中，将从文件 storms2.txt 中读取一组飓风信息，并将这些信息打印到屏幕上。

```
/*------------------------------------------------------------*/
/*  程序chapter7_1                                            */
/*                                                            */
/*  该程序从数据文件中读取一组飓风信息，并打印该信息          */

#include <stdio.h>
#define FILENAME "storms2.txt"

/*  定义一个表示飓风信息的结构体  */
struct hurricane
{
    char name[10];
    int year, category;
};

int main(void)
{
    /*  声明变量  */
    struct hurricane h1;
    FILE *storms;
```

337

```
    /*  从文件中读取信息并打印信息  */
    storms = fopen(FILENAME,"r");
    if (storms == NULL)
       printf("Error opening data file. \n");
    else
    {
       while (fscanf(storms,"%s %d %d",h1.name,&h1.year,
              &h1.category) == 3)
       {
           printf("Hurricane: %s \n",h1.name);
           printf("Year: %d, Category: %d \n",h1.year,
                  h1.category);
       }
       fclose(storms);
    }
    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
```

注意，在 fscanf 语句中引用飓风名称时用的是 h1.name，而不是 &h1.name。因为 name 是一个字符串，所以变量名称代表的是地址或指针。

使用表 5-2 中的信息作为测试数据来运行该程序，则执行结果的前几行应该如下所示：

```
Hurricane: Hazel
Year: 1954, Category: 4
Hurricane: Audrey
Year: 1957, Category: 4
```

练习

假设结构体变量 h1 和 h2 已经由下面的语句给出了定义：

```
struct hurricane
{
   char name[10];
   int year, category;
};

int main(void)
{
   /*  声明变量  */
   struct hurricane h1={"Audrey",1957,4};
   struct hurricane h2={"Frederic",1979,3};
```

写出下面语句执行后的输出结果。

```
1. printf("%s \n%s \n",h2.name,h1.name);
2. printf("Category %d hurricane: %s \n",h1.category,h1.name);
```

338

7.1.3 结构体的运算

前面已经讲过，结构体成员运算符（.）与结构体变量名一同使用便可以访问单独的数据成员。当结构体变量名不和结构体成员运算符一同使用时，它表示的是整个结构体。此时，赋值运算符便可以同结构体变量配合使用，将整个结构体直接赋值给类型相同的另一个结构体。具体实现过程如下面的语句所示：

```
struct hurricane
{
   char name[10];
   int year, category;
};

int main(void)
{
   /*  声明变量  */
   struct hurricane h1={"Audrey",1957,4}, h2;
...
   h2 = h1;
```

但是，关系运算符不可以用在整个结构体上。如果要对两个结构体进行比较，必须要对单独的数据成员进行比较才可以。

下面的程序说明结构体间的运算关系。该程序从文件 storms2.txt 中读入飓风信息，并打印所有种类为 5 的飓风名称。

```
/*-----------------------------------------------------------*/
/*  程序 chapter7_2                                          */
/*                                                           */
/*  该程序从数据文件中读入飓风信息，并打印所有种类为 5 的飓风名称  */

#include <stdio.h>
#define FILENAME "storms2.txt"

/*  定义一个表示飓风信息的结构体  */
struct hurricane
{
   char name[10];
   int year, category;
};

int main(void)
{
   /*  声明变量  */
   struct hurricane h1;
   FILE *storms;
   /*  从文件中读取信息并打印  */
   storms = fopen(FILENAME,"r");
   if (storms == NULL)
      printf("Error opening data file. \n");
   else
   {
      printf("Category 5 Hurricanes: \n");
      while (fscanf(storms,"%s %d %d",h1.name,&h1.year,
            &h1.category) == 3)
         if (h1.category == 5)
            printf("%s \n",h1.name);
      fclose(storms);
   }
   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------*/
```

339

7.2 使用结构体的函数

结构体可以作为参数在函数中使用，也可以用作函数返回值。下面将分别进行讨论。

7.2.1 结构体作为函数参数

整个结构体都可以作为参数传递给函数。当结构体作为参数时，它是一个传值引用。在函数被主程序引用时，实参中的每个数据成员的值均被传入函数中，在函数中可以通过形参的相应数据成员来使用。因此，改变形参中的值并不会对相应的实参产生影响。下面对前面小节中的程序进行修改。在本程序中，将使用函数打印飓风信息。

```
/*-----------------------------------------------------------------*/
/*  程序 chapter7_3                                                */
/*                                                                 */
/*  该程序从数据文件读入飓风信息，并使用函数打印该信息              */

#include <stdio.h>
#define FILENAME "storms2.txt"

/*  定义一个表示飓风信息的结构体  */
struct hurricane
{
   char name[10];
   int year, category;
};

int main(void)
{
   /*  声明变量和函数原型  */
   struct hurricane h1;
   FILE *storms;
   void print_hurricane(struct hurricane h);

   /*  从文件中读入数据并打印  */
   storms = fopen(FILENAME,"r");
   if (storms == NULL)
      printf("Error opening data file. \n");
   else
   {
      while (fscanf(storms,"%s %d %d",h1.name,&h1.year,
            &h1.category) == 3)
         print_hurricane(h1);
      fclose (storms);
   }

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数打印飓风信息                                             */

void print_hurricane(struct hurricane h)
{
   printf("Hurricane: %s \n",h.name);
   printf("Year: %d, Category: %d \n",h.year,h.category);
   return;
}
/*-----------------------------------------------------------------*/
```

340

当需要使用函数修改一个结构体中数据成员的值时，必须要使用该结构体的指针作为函数参数。这样函数便可以直接访问结构体的数据成员。在通过结构体指针获取数据成员时，要使用指针运算符（pointer operator）(->)，而不再是结构体成员运算符。它的使用方法将在7.6节中介绍，届时还将对动态数据结构进行讨论。

7.2.2 返回结构体的函数

函数可以返回一个 struct 类型的值。当函数被调用结束时，整个结构体会返回到被调用的函数中。为了说明这个过程，下面给出一个关于海啸信息的新结构体。信息主要包括海啸发生的日期（月，日，年）、海啸的最大高度（英尺）、遇难人数以及海啸位置。其中，月，日，年和遇难人数用整数表示，最大高度用浮点数表示，位置用字符串表示。这样一来，表示海啸信息的结构体可以定义成如下形式：

```
struct tsunami
{
   int mo, da, yr, fatalities;
   double max_height;
   char location[20];
};
```

再给出一个函数，从键盘读入信息，将其保存在结构体中，并作为返回值返回至 main 函数中。假设此结构体已经在 main 函数中定义，下面给出函数原型的定义语句：

```
struct tsunami get info(void);
```

341

在这个函数中，我们通知用户输入信息，并将此信息以结构体的形式返回。该函数的代码实现如下：

```
/*-----------------------------------------------------------------*/
/*  该函数从用户输入读取信息，并保存信息至 tsunami                  */
/*  结构体中                                                       */

struct tsunami get_info(void)
{
   /*  声明变量  */
   struct tsunami t1;

   printf("Enter information for tsunami in following order: \n");
   printf("Enter month, day, year, number of deaths: \n");
   scanf("%d %d %d %d",&t1.mo,&t1.da,&t1.yr,&t1.fatalities);
   printf("Enter location (<20 characters, no spaces): \n");
   scanf("%s",t1.location);

   return(t1);
}
/*-----------------------------------------------------------------*/
```

修改

1. 编写函数，将用户输入的信息读入结构体 hurricane 的变量中。
2. 编写函数，打印结构体 tsunami 的变量中的海啸信息。

7.3 解决应用问题：指纹分析

本节运用前面介绍的语句来编程解决关于指纹分析的问题。指纹可以分为三种类型：环形、螺旋形和拱形。有大约 60% ~ 65% 的指纹是环型指纹，大约 30% ~ 35% 的指纹是螺旋形，大约 5% 是拱形。在进行未知指纹匹配时，为了减少数据库中需要比对的指纹数量，可以先排除那些类型不符的指纹。例如，一个螺旋形指纹就不需要再同拱形指纹相比较了。实际上，现在已经开发出一些新技术，能够快速识别 10 根手指的指纹类型，并立即从数据库中排除不需比对的指纹。这些技术大体上都基于专利技术——最早的指纹识别技术之一。

下面为每个手指的指纹记录定义一个编码规则。第一个字母表示指纹的左右手分类，右手指纹标识为R，左手指纹标识为L。第二个字母区分不同的手指，比如t（拇指）、i（食指）、m（中指）、r（无名指）和p（小指）。这样就为每个手指定义一个变量，然后根据每个手指的指纹是否是螺旋形分别对它们进行赋值。每个手指对应的变量的赋值情况如下所示：

342

- 如果Rt或Ri是螺旋形，它的值是16。
- 如果Rm或Rr是螺旋形，它的值是8。
- 如果Rp或Lt是螺旋形，它的值是4。
- 如果Li或Lm是螺旋形，它的值是2。
- 其余的值全为0。

根据上面的赋值条件，可以按照下面的式子为一个指纹记录（包含10根手指）计算一个全局分类：

$$全局分类值 = \frac{Ri + Rr + Lt + Lm + Lp + 1}{Rt + Rm + Rp + Li + Lr + 1}$$

可以发现，这个全局分类值不可能为0，同时分母也不可能为0。当全局分类值计算出来后，就会保存在它的指纹记录中。对于一个未知指纹记录，首先计算它的全局分类值，然后就能在数据库中找到一个小范围的种类信息与之匹配。这样一来，匹配指纹的查找工作就会从比对上百万条指纹数据精简到只需要比对几千条指纹数据。在确定了与未知指纹种类值相近的指纹范围后，剩下的工作只需要将未知指纹的细节点同已知指纹的细节点进行一一比对即可。

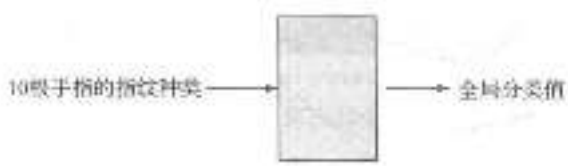
在本节要设计一个C函数，从结构体中得到一组指纹信息，然后计算相应的全局分类值，并将该分类值存储在该结构体中。因此，这个函数可以用来为一个未知指纹设定全局分类值，或者可以被用在循环体中计算数据库中每组指纹的全局分类值。

1. 问题陈述

编写函数计算一个指纹的全局分类值。

2. 输入/输出描述

下图展示了，本程序的输入是10根手指的指纹种类。程序输出是根据输入信息计算出的全局分类值。



3. 手动演算示例

假设10根手指中，螺旋形的指纹出现在右手拇指、右手无名指和左手中指上。除了这三根手指之外，其余手指的值全部为0：

$$Rt = 16$$
$$Rr = 8$$
$$Lm = 2$$

343

根据这些条件，可以手动计算出指纹的全局分类值：

$$\text{全局分类值} = (8 + 2 + 1)/(16 + 1) = 11/17 = 0.65$$

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将方案分解成几个连续的解决步骤。

函数的分解提纲

1）从函数输入中获取指纹数据。

2）根据指纹信息计算出指纹的全局分类值。

3）将全局分类值存储在指纹结构体中。

我们还需要开发一个程序，用来测试这个函数。测试程序的分解提纲如下：

测试程序的分解提纲

1）从用户处读入指纹信息。

2）使用函数计算全局分类值。

3）打印全局分类值。

分解提纲提供了很简单的算法，所以可以直接转化为C程序。

```
/*-----------------------------------------------------------------*/
/*  程序chapter7_4                                                 */
/*                                                                 */
/*  该程序将指纹信息存储在结构体中。随后引用函数计算该指纹的全局分类值  */
#include <stdio.h>
/*  为指纹信息定义一个结构体。指尖顺序依次为右手，拇指到小指；左手，拇指到  */
/*  小指。在程序中用L表示环形，W表示螺旋形，A表示拱形                    */
struct fingerprint
{
  int ID_number;
  double overall_category;
  char fingertip[10];
};
int main(void)
{
   /*  声明变量并初始化  */
   struct fingerprint new_print;
   double compute_category(struct fingerprint f);
   /*  指定新的指纹信息  */
   new_print.ID_number = 2491009;
   new_print.overall_category = 0;
   new_print.fingertip[0] = 'W';
   new_print.fingertip[1] = 'L';
   new_print.fingertip[2] = 'L';
   new_print.fingertip[3] = 'W';
   new_print.fingertip[4] = 'A';
   new_print.fingertip[5] = 'L';
   new_print.fingertip[6] = 'L';
   new_print.fingertip[7] = 'W';
   new_print.fingertip[8] = 'A';
   new_print.fingertip[9] = 'L';
   /*  引用函数来计算全局分类值  */
   new_print.overall_category = compute_category(new_print);
   /*  打印计算出的全局分类值  */
   printf("Fingerprint Analysis for ID: %i \n",
          new_print.ID_number);
   printf("Overall Category: %.2f \n",new_print.overall_category);
```

[344]

```
    /* 退出程序 */
    return 0;
}

/*-----------------------------------------------------------------*/
/* 该函数计算一个给定的全局分类值                                  */

double compute_category(struct fingerprint f)
{
    /* 声明变量并初始化 */
    double Rt=0, Ri=0, Rm=0, Rr=0, Rp=0, Lt=0, Li=0, Lm=0, Lr=0,
           Lp=0, num, den;

    /* 根据螺旋形指纹来设置初值 */
    if (f_fingertip[0] == 'W')
       Rt = 16;
    if (f_fingertip[1] == 'W')
       Ri = 16;
    if (f_fingertip[2] == 'W')
       Rm = 8;
    if (f_fingertip[3] == 'W')
       Rr = 8;
    if (f_fingertip[4] == 'W')
       Rp = 4;
    if (f_fingertip[5] == 'W')
       Lt = 4;
    if (f_fingertip[6] == 'W')
       Li = 2;
    if (f_fingertip[7] == 'W')
       Lm = 2;
    /* 计算全局分类值的分子和分母 */
    num = Ri + Rr + Lt + Lm + Lp + 1;
    den = Rt + Rm + Rp + Li + Lr + 1;

    return num/den;
}
/*-----------------------------------------------------------------*/
```

345

需要注意的是，程序打印出结构体中的全局分类值只是为了确保该值已被存储进结构体中。

5. 测试

测试程序已经将指纹类别进行初始化，以匹配手动演算示例中的数据。得到的输出结果如下所示：

```
Fingerprint Analysis for ID Number Z4910049
Overall Category:  0.65
```

答案与手动演算示例中的结果一致，因此，现在可以改变测试数据来重新测试程序。

修改

根据下面的要求，对本节设计的程序作出适当修改：

1. 修改程序，计算并输出手指上螺旋的数目。
2. 修改程序，计算并输出手指上拱形的数目。
3. 修改程序，计算并输出手指上环形的数目。
4. 修改程序，使其能够同时解决问题1、2和3，并输出三种不同指纹类型的数目。
5. 修改问题4中的程序，分别输出三种不同指纹类型所占的百分比。

7.4 结构数组

在工程应用中，用数组来存储待分析的数据信息中是非常方便的。然而，数组仅可以存储同一数据类型的信息，如整型数组或字符串数组。如果需要利用数组来存储不同数据类型的信息，就可以使用结构数组。例如，如果要将飓风信息存入数组，可以使用结构类型为 hurricane 的数组；如果要将海啸信息存入数组，可以使用结构类型为 tsunami 的数组。如此，便可以通过如下语句定义一个含有 25 个元素的数组，其中每个元素都是结构类型 hurricane（这里也将再次给出 hurricane 的结构体定义）：

[346]

```
struct hurricane
{
  char name[10];
  int year, category;
};
...
struct hurricane h[25];
```

数组中的每个元素都是一个包含三个变量的结构体，如下表所示：

h[0]	?	?	?
h[1]	?	?	?
...	...	...	...
h[24]	?	?	?

为了访问数组中的结构体数据成员，必须要指定数组名、数组下标以及数据成员名称。例如，下面要对数组 h 中的第一个飓风赋值：

```
h[0].name = "Camille";
h[0].year = 1969;
h[0].category = 5;
```

要访问数组中的一个完整结构体，必须要指定数组名和下标。例如，现在调用 7.1 节中定义的输出函数，则可以通过如下语句输出数组中的第一个飓风的信息：

```
print_hurricane(h[0]);
```

输出结果为

```
Hurricane: Camille
Year: 1969, Category: 5
```

下面的程序中，首先将数据文件中的一组飓风信息读入数组。然后确定该数组中飓风的最大等级，并输出最大等级中的所有飓风名称。很显然，在读取数据文件的过程中是无法直接得出结果的。因为只有当浏览完毕文件中的所有信息，才可以确定出飓风的最大等级。在这之后，重新检查一遍文件中的数据才能把所有最大等级的飓风信息显示出来。

[347]

```
/*-----------------------------------------------------------*/
/*  程序 chapter7_5                                          */
/*                                                           */
/*  该程序先从数据文件中读取飓风信息，然后输出文件中最大等级的所有飓风信息 */

#include <stdio.h>
```

```
#define FILENAME "storms2.txt"

/*  定义结构体来表示飓风信息  */
struct hurricane
{
   char name[10];
   int year, category;
};

int main(void)
{
   /*  声明变量和函数原型  */
   int max_category=0, k=0, npts;
   struct hurricane h[100];
   FILE *storms;
   void print_hurricane(struct hurricane h);

   /*  从文件中读取并输出信息  */
   storms = fopen(FILENAME,"r");
   if (storms == NULL)
      printf("Error opening data file. \n");
   else
   {
      printf("Hurricanes with Maximum Category \n");
      while (fscanf(storms, "%s %d %d",h[k].name,&h[k].year,
            &h[k].category) == 3)
      {
         if (h[k].category > max_category)
            max_category = h[k].category;
         k++;
      }
      npts = k;

      for (k=0; k<=npts-1; k++)
         if (h[k].category == max_category)
            print_hurricane(h[k]);

      fclose(storms);
   }

   /*  退出程序  */
   return 0;
}
/*-----------------------------------------------------------------*/
/*  该函数输出相应的飓风信息                                       */

void print_hurricane(struct hurricane h)
{
   printf("Hurricane: %s \n",h.name);
   printf("Year: %d, Category: %d \n",h.year,h.category);
   return;
}
/*-----------------------------------------------------------------*/
```

348

7.5 解决应用问题：海啸分析

海啸是一种破坏力极强的海浪，通常是由地震、海底火山爆发以及海底滑坡等引起的洋流变化所导致的。在浅水层，海啸能够以每小时125英里[1]的速度传播；而在深水层，海啸的传播速度则可以达到每小时400英里。关于海啸的记录最早可以追溯到几百年前发生在智利和秘鲁海岸的海啸事件。例如，在1562年10月28日，智利的一场地震造成了高达52英

[1] 1英里=1.609344千米

尺[1]的海浪。

在相关的记录中还有一些更大规模的海啸事件，其中包括发生在 1899 年 9 月 10 日的阿拉斯加湾海啸。这次海啸的成因主要是地震和海底滑坡，它引起的海浪高达 197 英尺。在 1964 年 3 月 28 日，同样是阿拉斯加湾，这次地震引发了高达 230 英尺的海啸。距离现在更近一些的记录中，1994 年 6 月 3 日在印尼的爪哇岛西部发生地震，引起的海啸高度有 197 英尺。1998 年 7 月 17 日，在新几内亚的巴布亚岛发生地震，引起了 49 英尺的海啸，这次海啸虽然规模不是很大，但是它造成了超过 2 200 人遇难。2011 年 3 月 11 日，日本海岸发生 9.0 级地震，引起了超过 130 英尺的海啸，导致 13 000 多人遇难。

下面要设计一个程序，首先从文件中读取 20 世纪 90 年代以来发生的大规模海啸信息（见表 7-1），并输出统计报告，报告中要分别指出这些海啸的最大高度（以英尺为单位）、平均高度以及所有在平均高度之上的海啸发生地点。

表 7-1　20 世纪 90 年代以来的大规模海啸

日期	位置	最大高度（m）	死亡人数
1992 年 9 月 2 日	尼加拉瓜	10	170
1992 年 12 月 2 日	弗洛雷斯岛	26	1000[①]
1993 年 7 月 12 日	日本的奥尻町	31	239
1994 年 6 月 3 日	爪哇岛东部	14	238
1994 年 11 月 14 日	民都洛岛	7	49
1995 年 10 月 9 日	墨西哥的伦利斯科	11	1
1996 年 1 月 1 日	苏拉威西岛	3.4	9
1996 年 2 月 17 日	伊里安查亚	7.7	161
1996 年 2 月 21 日	秘鲁	5	12
1998 年 7 月 17 日	新几内亚的巴布亚岛	15	2200[①]

①死亡人数为估计值

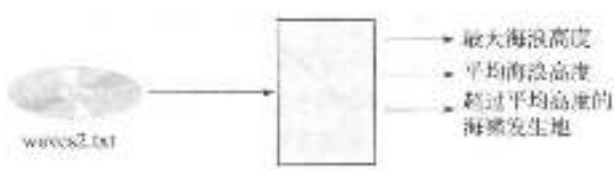
349

1. 问题陈述

输出一份统计报告，报告中要分别指出数据文件 waves2.txt 中记录的海啸的最大高度和平均高度（以英尺为单位），以及所有超过平均高度的海啸的发生地点。

2. 输入 / 输出描述

下面的 I/O 图显示该程序的输入为数据文件，输出为报告信息。



3. 手动演算示例

如果假设数据文件包含的是表 7-1 中的海啸信息，那么海浪最大高度为 31 米。将所有海浪高度相加，再除以 10，以此来计算平均海浪高度；另外要将计算结果的单位由米转换为英尺（1 米 =3.28 英尺）。最后在生成报告时，可以将该计算结果和超过平均高度的海

[1] 1 英尺 = 0.3048 米

啸发生地点一同输出：

```
Summary Information for Tsunamis
Maximum Wave Height (in feet):  101.68
Average Wave Height (in feet):  42.67
Tsunamis with greater than the average height:
Flores_Island
Okushiri,_Japan
Eastern_Java
Papua,_New_Guinea
```

在数据文件中，使用下划线来代替字符串中的空格，以使这些地名信息可以作为一个完整的字符串被读入。

4. 算法设计

在设计具体算法之前，首先根据问题列出分解提纲，将问题分解成几个连续的解决步骤。

分解提纲

1）将海啸数据读入数组，并确定最大海浪高度和平均海浪高度。

2）输出最大海浪高度和平均海浪高度。

350 3）输出所有高于平均海浪高度的海啸的发生地点。

[最终后的伪代码]

```
主函数：if 文件不能打开
            输出错误信息
        else
            将海啸数据读入数组，并确定最大海浪高度和平均海浪高度。
            输出最大海浪高度和平均海浪高度。
            输出所有高于平均海浪高度的海啸发生地点。
```

伪代码中的步骤足够详细，可以将其直接转换为C语句：

```
/*-----------------------------------------------------------------*/
/*  程序 chapter7_6                                                */
/*                                                                 */
/*  该程序从文件中读入海啸数据，然后输出海浪的最大高度、平均高度以及所有  */
/*  大于平均高度的海啸的发生地点                                    */
#include <stdio.h>
#define FILENAME "waves2.txt"

/*  定义结构体以表示海啸信息  */
struct tsunami
{
  int mo, da, yr, fatalities;
  double max_height;
  char location[20];
};

int main(void)
{
   /*  声明变量  */
   int k=0, npts;
   double max=0, sum=0, ave;
   struct tsunami t[100];
   FILE *waves;

   /*  从文件中读取和输出数据  */
```

```
    waves = fopen(FILENAME,"r");
    if (waves == NULL)
       printf("Error opening data file. \n");
    else
    {
       while (fscanf(waves,"%d %d %d %d %lf %s",&t[k].no,&t[k].da,
                     &t[k].yr,&t[k].fatalities,&t[k].max_height,
                     t[k].location) == 6)
       {
          sum = sum + t[k].max_height;
          if (t[k].max_height > max)
             max = t[k].max_height;
          k++;
       }
       npts = k;
       ave = sum/npts;
       printf("Summary Information for Tsunamis \n");
       printf("Maximum Wave Height (in feet): %.2f \n",max*3.28);
       printf("Average Wave Height (in feet): %.2f \n",ave*3.28);
       printf("Tsunamis with greater than average heights: \n");
       for (k=0; k<=npts-1; k++)
          if (t[k].max_height > ave)
             printf("%s \n",t[k].location);
       fclose(waves);
    }
    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------------*/
```

351

5. 测试

使用手动演算示例中的数据作为程序输入，可以得到如下输出结果：

```
Summary Information for Tsunamis
Maximum Wave Height (in feet): 101.68
Average Wave Height (in feet): 42.67
Tsunamis with greater than the average heights:
Flores_Island
Okushiri,_Japan
Eastern_Java
Papua,_New_Guinea
```

修改

根据本节设计的程序来解决下列问题：

1. 修改程序，找出震度最大的海啸发生年份，输出这一年海啸发生的总数量。
2. 修改程序，找到并输出遇难人数最多的海啸发生的日期。
3. 修改程序，找到并输出遇难人数超过 100 的所有海啸发生的地点。
4. 修改程序，输出在 7 月份发生的海啸总数。
5. 修改程序，输出发生在秘鲁（Peru）的海啸数量。假设海啸发生的位置信息还包含秘鲁的城市名称，因此需要搜索含有“Peru”的字符串。

352

*7.6 动态数据结构

静态数据结构，如数组，在程序执行时大小是固定的。在第 5 章介绍过，数组的大小必

须用常量来声明。在声明语句中定义数组时，程序会为数组元素分配一段连续的内存单元。数组中的元素可以通过数组名称（首元素的地址）和下标（距离首元素的偏移量）来引用。在使用静态数据结构时，要求程序员事先了解数据集的大小，以使得初始化时分配的内存空间足够并且不浪费，同时还必须小心不能使数组溢出。相反，动态数据结构（dynamic data structure）是在程序的执行过程中，可以根据需要来收缩和扩展的数据结构。内存按需分配和释放，由于数据不是一定存储在连续的内存空间里，所以要使用指针将数据连接起来。

下面通过链表（linked list）来说明如何使用动态数据结构。链表是由指针连接的节点（node）构成。节点中包含数据项（可以是一个或者多个变量的集合）和指向下一节点的指针。一般假设链表中节点间的数据排列是有序的，比如升序，而链表的一般操作主要是增加新节点或者删除旧节点。链表最简单的图示方法就是画出一组节点，每个节点中包含着信息，同时节点之间通过指针逐个连接起来。图7-1展示了一个具有4个节点的链表，其中包含了有序数据信息10、14、21和35。此外还有一个单独指针，称之为头（head）指针，它指向链表的第一个节点。

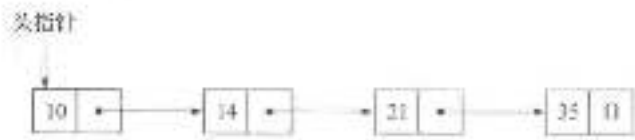


图7-1 链表

为了访问链表，首先使用head指针来引用第一个节点的信息（包含数值10的节点）。然后，由于第一个节点中包含了指向下一节点（包含数值14的节点）的指针，因此可以继续移动到第二个节点。同样，可以使用第二个节点中的指针移动到第三个节点（包含数值21的节点），以此类推。链表的末尾节点包含了一个值为NULL的指针，以表明此时正在访问最后一个节点。通常都使用符号Ω来表示链表的结束（Omega，Ω是希腊字母表中的最后一个字母）。

为了用C语言实现链表，先来看看所需的步骤。链表的每个节点都是一个结构体。在下面的例子中，结构体包含一个整数值和一个指向下一节点的指针。在需要的时候，通过动态存储分配语句来为每个节点分配内存空间。最后一个节点中的指针值为NULL。

如果要在链表中插入一个节点，则需要找到相应的插入位置。首先使用指向第一个节点的指针来找到链表的第一个数据值。如果该数据值小于将要插入的值，则将当前节点中的指针移向下一节点。假设发现要插入的数据值已经存在于链表中，会输出提示信息，而不再插入重复值（根据不同的应用需求，插入重复值可能是合法操作）。有效的数据插入位置有以下4种：首节点之前、两个节点之间、尾节点之后或是在空链表中。因此必须要小心处理这4种情况，以确保节点的指针不会出错。以图7-1中给出的链表基础，图7-2～图7-5分别描述了这4种情况下进行节点插入操作后链表的状况。

353

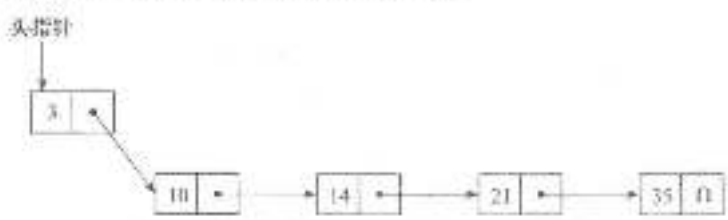


图7-2 在首节点之前插入（需要更新头指针）

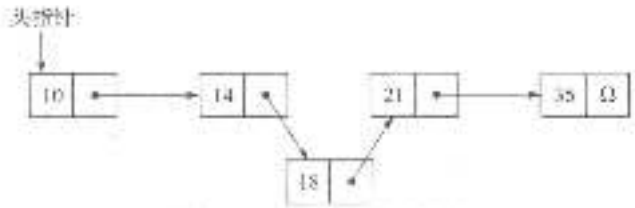


图 7-3 在两节点之间插入

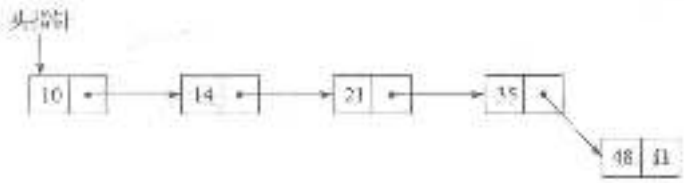


图 7-4 在尾节点之后插入

354

从链表中删除一个节点，需要找到被删除节点的位置。同插入节点一样，再次使用指向链表头部的指针来找到链表的第一个数据，然后使用当前节点中的指针移动到下一节点。如果访问到的节点数据大于被删除节点，则输出“被删除节点在链表中不存在”的提示信息。被删除节点可能是链表的首节点、两个节点之间的节点或者是最后一个节点，因此必须小心处理每种情况，以确保相应节点的指针不会出错。在图 7-2 展示的链表的基础上，图 7-6 ~ 图 7-8 分别描述了这三种节点删除的情况。另外要注意的是，被删除的节点（数据值）不能再被访问。例如，在图 7-6 中，无法再访问到数据 10；在图 7-7 中，无法再访问到数据 21；在图 7-8 中，无法再访问到数据 35。此外，如果链表中只有一个节点，那么该链表在首节点被删除后，就变成了空链表。

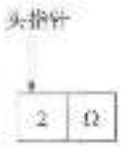


图 7-5 在空链表中插入（需要更新头节点）

有了这些基本介绍，下面就要设计 4 个函数来创建和维护一个链表。第一个函数要判断链表是否为空链表（empty list）；第二个函数输出链表内容；第三个函数在正确位置插入节点；第四个函数负责删除节点。使用下面的结构体来表示链表的节点：

```
struct node
{
   int data;
   struct node *link;
};
```

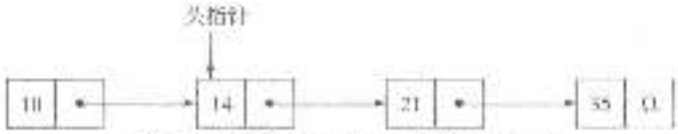


图 7-6 删除首节点（需要更新头指针）

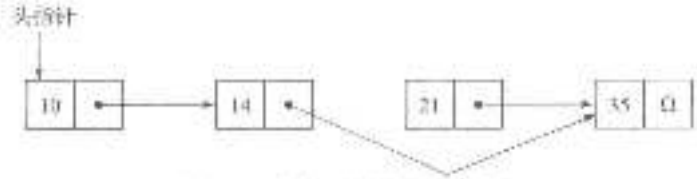


图 7-7 删除两节点之间的节点

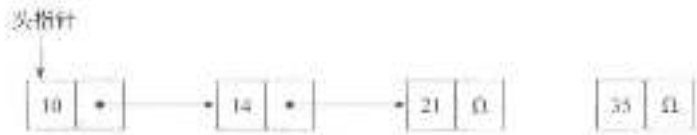


图 7-8 删除最后一个节点

355

结构体 node 包含两部分，一部分是存储数据值的整型变量，另一部分则是指向链表中下一个节点的指针。以下是判断链表是否为空的函数原型：

```
int empty(struct node *head);
```

函数 empty 检测链表头节点。如果链表为空，函数返回 1，否则返回 0。该函数会被其余三个函数使用：

```
/*-----------------------------------------------------------------*/
/*  该函数判断链表是否为空，如果为空则返回整数值 1                  */

int empty(struct node *head)
{
   /*  声明变量  */
   int k=0;

   /*  确定链表是否为空  */
   if (head == NULL)
      k = 1;

   /*  返回一个整数值  */
   return k;
}
/*-----------------------------------------------------------------*/
```

负责输出链表内容的函数原型为：

```
void print_list(struct node *head);
```

函数 print_list 从链表头部开始，输出每个节点中的数据值，直到链表结尾。if 语句中的判定条件是对函数 empty 的引用：

```
if (empty(head))
...
```

前面介绍过，如果 if 语句的判定条件是一个数值，则 0 值被认为是 false，非 0 值被认为是 true。因此，如果链表为空，函数返回 1，此时条件为 true；否则，函数返回 0，此时条件为 false。

```
/*-----------------------------------------------------------------*/
/*  该函数输出链表内容                                              */

void print_list(struct node *head)
{
   /*  声明变量  */
   struct node *next;

   /*  输出链表  */
   if (empty(head))
      printf("Empty list \n")
   else
   {
      printf("List Values: \n");
      next = head;
```

356

```
        while (next->link != NULL)
        {
            printf("%d \n",next->data);
            next = next->link;
        }
        printf("%d \n",next->data);
    }

    /*  无返回值  */
    return;
}
/*-----------------------------------------------------------------*/
```

函数insert从链表头部开始，在正确的位置插入新节点。如果要插入的数据值在链表中已经存在，则输出提示信息，并停止插入节点。前面讲过，一个链表的表头就是指向链表第一个节点的指针。当指针作为函数参数时，这是一个传值调用。当出现在首节点之前，或在空链表中插入新节点时，就需要修改头指针，所以必须要将参数声明为一个指向头指针的指针变量，使其成为传址引用。这样一来，就可以根据需要在函数中更新头指针了。仔细观察下面的语法：

```
/*-----------------------------------------------------------------*/
/*  该函数在链表中插入一个新节点                                    */

void insert(struct node **ptr_to_head, struct node *nw)
{
    /*  声明变量为函数原型  */
    struct node **next;

    /*  检查是否为空链表  */
    if (empty(*ptr_to_head))
        *ptr_to_head = nw;
    else
    /*  遍历链表，找到正确的插入位置  */
    {
        next = ptr_to_head;
        while ( ((*next)->data < nw->data) &&
                ((*next)->link != NULL) )
            next = &(*next)->link;
        /*  检查插入值是否已存在  */
        if ((*next)->data == nw->data)
            printf("Node already in list. \n");
        else
        /*  检查是否在尾节点之后插入  */
            if ((*next)->data < nw->data)
                (*next)->link = nw;
            else
            {
                nw->link = *next;
                *next = nw;
            }
    }
    /*  无返回值  */
    return;
}
/*-----------------------------------------------------------------*/
```

357

删除节点的函数原型是：

```
void remove(struct node **ptr_to_head, int old);
```

remove函数从链表的表头开始，逐一查找数据值为old的节点。如果没有找到该节点，则打印相关信息。如果找到了，便删除该节点，并释放内存。由于当被删除的结点是链表的头结点时需要更新链表的头指针，所以需要向函数传递一个指向头节点的指针。（在这里函数名没有使用delete，是因为很多编译器都将delete作为保留字。）

```
/*-----------------------------------------------------------------*/
/*  该函数从链表中删除一个节点                                     */
void remove(struct node **ptr_to_head, int old)
{
   /*  声明变量和函数原型  */
   struct node *next, *last, *hold, *head;
   /*  检查是否为空链表  */
   head = *ptr_to_head
   if (empty(head))
      printf("Empty list. \n");
   else
   /*  检查是否删除第一个节点  */
   {
      if (head->data == old)
      {
         /*  删除第一个节点  */
         hold = head;
         *ptr_to_head = head->link;
         free(hold);
      }
      else
      /*  遍历链表寻找值为old的节点  */
      {
         next = head->link;
         last = head;
         while ((next->data < old)) &&
               (next->link != NULL))
         {
            last = next;
            next = next->link;
         }
         /*  如果找到该节点，则删除节点  */
         if (next->data == old)
         {
            hold = last;
            last->link = next->link;
            free(hold);
         }
         else
            printf("Value %d not in list. \n",old);
      }
   }
   /*  无返回值  */
   return;
}
/*-----------------------------------------------------------------*/
```

358

下面给出一个main函数来检验这些函数的功能。每次调用insert函数和remove函数之后，都会调动print_list函数来验证它们是否正常工作。

```
/*-----------------------------------------------------------------*/
/*  程序chapter7_7                                                 */
/*                                                                 */
```

```
/*  这程序用来检验链表的插入和删除函数  */
#include <stdio.h>
#include <stdlib.h>
/*  定义一个表示链表节点的结构体  */
struct node
{
   int data;
   struct node *link;
};
int main(void)
{
   /*  声明变量和函数原型  */
   int k=0, old, value;
   struct node *head, *next, *previous, *nw, **ptr_to_head=&head;
   void insert(struct node **ptr_to_head, struct node *nw);
   void remove(struct node **ptr_to_head, int n);
   int empty(struct node *head);
   void print_list(struct node *head);
   /*  生成一个含有5个节点的链表，并打印该链表  */
   head = (struct node *)malloc(sizeof(struct node));
   next = head;
   for (k=1; k<=5; k++)
   {
      next->data = k*5;
      next->link = (struct node *)malloc(sizeof(struct node));
      previous = next;
      next = next->link;
   }
   previous->link = NULL;
   print_list(head);
   /*  允许用户在链表中插入或删除节点  */
   while (k != 2)
   {
      printf("Enter 0 to delete node, 1 to add node, 2 to quit. \n");
      scanf("%d",&k);
      if (k == 0)
      {
         printf("Enter data value to delete: \n");
         scanf("%d,&old);
         remove(ptr_to_head,old);
         print_list(head);
      }
      else
         if (k == 1)
         {
            prinf("Enter data value to add: \n");
            scanf("%d,&value);
            nw = (struct node *)malloc(sizeof(struct node));
            nw->data = value;
            nw->link = NULL;
            insert(ptr_to_head,nw);
            print_list(head);
         }
   }
   /*  退出程序  */
```

359

```
    return 0;
}
/*-----------------------------------------------------------*/
```

试运行程序，得到执行结果如下所示：

```
List Values:
5
10
15
20
25
Enter 0 to delete node, 1 to add node, 2 to quit.
1
Enter data value to delete:
16
List Values:
5
10
15
16
20
25
Enter 0 to delete node, 1 to add node, 2 to quit.
0
Enter data value to delete:
10
List Values:
5
15
16
20
25
```

360

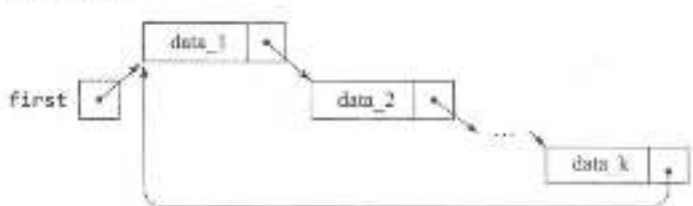
其他动态数据结构

在这里要介绍另外5种功能强大的数据结构。尽管这些数据结构每一种都能单独作为一节的主题来讲述，但是我们把这些内容都放在了这一小节中，因为这些数据结构在空间上与链表的节点非常相似，都是由链表演变而来。

1. 循环链表

如果将链表的最后一个节点指向链表的头节点，便可以生成一个循环链表（circularly linked list），如图7-9所示。要注意的是，在循环链表中几乎不使用NULL常量，这是因为在这里并不存在末尾节点。但是，在一个空的循环链表中，头指针first还是会指向NULL。

在循环链表中的插入和删除操作同普通链表类似。此外，在操作指针first时要格外注意，因为它不仅指向链表的头部，同时还要用作遍历访问结束的标志，标识着一次沿着链表的数据访问又回到了表头。



361

图7-9 循环链表

操作系统是一个非常复杂的程序，其中就有很多应用功能非常适用于使用循环链表实现。例如，假设一个特定程序用来追踪系统用户的交互行为。每当一个新用户登录系统，就将该用户添加到链表；每当用户从系统注销，就将该用户从链表中删除。因为总是将新用户添加在末尾，所以该链表是有序的。当运行交互程序时，计算机在第一个用户的程序中执行若干步骤，然后在下一个用户的程序中执行若干步骤，以此类推，直到最后回到第一个用户。这个过程就在环内持续进行。操作系统中需要存储这些用户信息，并且保持调用这些用户的程序的次序，这样的应用场景非常适合于用循环链表来实现。这种用循环链表实现的数据结构有时也称为轮转（round-robin）数据结构。

2. 双向链表

在链表中，每个节点中的指针都被用来指向下一个节点的头部。而有些应用程序中需要将数据前后相连，以便于在数据链中前后移动，这种类型的链表叫作双向链表（doubly linked list）。在图 7-10 中展示了一个双向链表，可以看到每个节点中同时拥有两个指针，分别是指向前一个节点的前驱指针和指向后一个节点的后继指针。

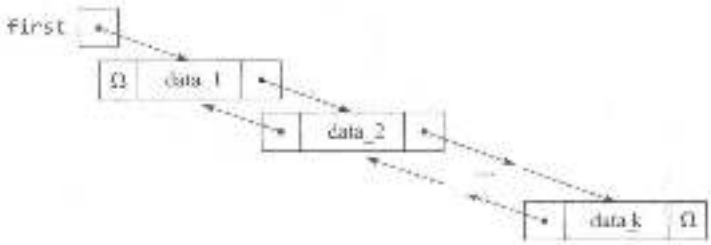


图 7-10　双向链表

尽管在链表中能够实现双向移动的好处非常明显，但与此同时对链表的操作步骤也相应复杂了许多。例如，实现插入操作时需要改变两个前驱指针和两个后继指针。同时，还应该保证链表中首节点的后继指针和末节点的前驱指针均被赋值为 NULL。

双向链表的一大优势就是，在链表中插入或删除节点，不必每次都返回至链表开头。例如，假设现在要向链表第 10 个位置处插入节点，此时不用从表头开始，而是可以将插入项直接同当前指针所指向的节点相比较。如果新节点应该位于当前节点之前，当前指针就沿着前驱指针向链表前部移动，直到找到新节点的合适位置；如果新节点应该位于当前节点之后，就沿着后继指针继续访问链表节点，直到找到新节点的合适位置。在某些特定情况下这种插入方式会非常高效。

3. 堆栈

堆栈（stack）是一种最常用的动态数据结构。它经常被描述为桶状，如图 7-11 所示。向堆栈中添加数据项就类似于将其投入桶里。栈顶元素通常都是最后加入的项。当需要从堆栈中移除数据项时，首先移除的是栈顶元素，或者说是最后加入的项。这个数据结构叫作 LIFO（last in-first out，后进先出）结构。向堆栈中添加项的函数称为压栈函数，从栈中移除项的函数称为出栈函数。图 7-12 展示了对栈进行添加项（压栈）和移除项（出栈）操作时栈内的情况。从图中可以清楚地看到堆栈属于动态数据结构。因此，可以使用一个链表结构来实现，如图 7-13 所示。

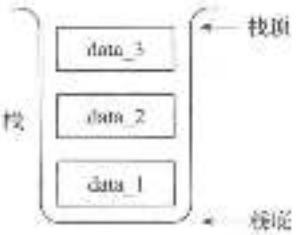


图 7-11　栈结构

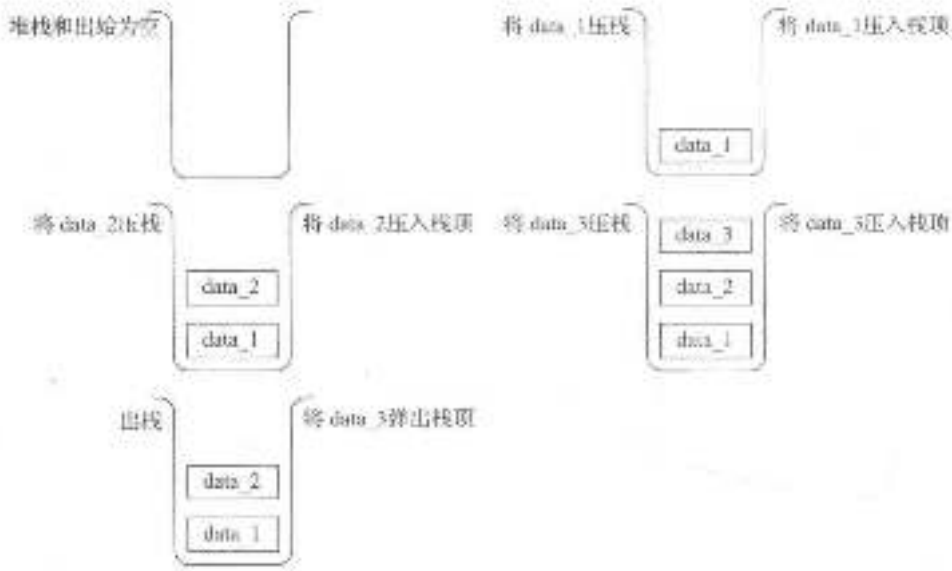


图 7-12 压栈和出栈操作

在进行压栈和出栈操作时必须要密切注意节点间指针和栈顶、栈底指针的指向情况。栈顶指向的是下一次插入操作的位置，所以如果栈顶和栈底指向了同一位置，就说明栈空了。

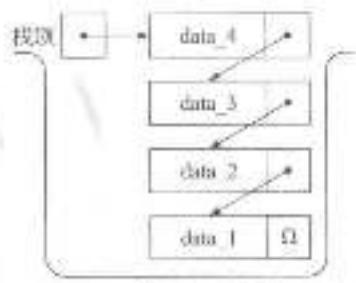


图 7-13 使用链表来实现栈

堆栈的相关应用十分广泛。例如，如果要将一组数据逆序打印出来，可以先顺序获取每个数据，并依次压入栈中，将数据全部压栈完毕之后，再从栈中将其逐一移除。由于最后入栈的元素就是第一个被移除的项，所以将数据依次从堆栈移除的次序就是最初压栈时的逆序。除此之外，在很多应用中需要将数据临时保存，然后再取出最近被存储的数据。编译器就会频繁使用到栈，因为它需要逐条分析程序语句的语法，还要将几条语句共同转化成机器语言或者汇编语言。

4. 队列

队列（queue）的数据结构应该是在生活中非常熟悉的，尽管你可能从没意识到这种结构还有名称。当你每次排队时，不论是在杂货店、音像店，还是快餐馆，你都处于一个队列中。在队列这种数据结构中，数据项从一端加入，并从另一端移除，如图 7-14 所示。队列也叫作 FIFO（first in–first out，先进先出）结构。

在操作系统中经常会有很多等待计算机资源的用户，所以通常会使用队列来对这些用户进行记录和管理。例如，假定当前网络中有一台彩色打印机。如果同时有几个用户需要打印报告，那么操作系统便会将这些用户排成“队列”，以确保按照发出打印请求的顺序，一次只能打印一份报告。

对队列进行操作的函数，必须要执行从队列一端插入数据，从另一端移除数据的操作。因此，一个队列需要两个指针，分别指向队列头部和尾部（有时也叫作头指针和尾指针）。此外，队列操作显然还需要能够检查空队列。队列中的指针同一般的链表类似，但是只能从一端插入，从另一端移除。图 7-15 展示了如何使用链表和指针来实现一个队列。

363

图 7-14 队列结构

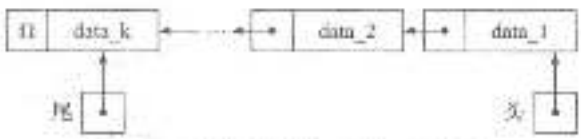


图 7-15 使用链表来实现一个队列

5. 二叉树

最后一个要介绍的动态数据结构是二叉树。与现实生活中的树类似，二叉树（binary tree）从一个独立节点开始不断伸展开，这个起始节点通常称为根节点。根节点有左右两个分支。同时每个分支上的节点又有左右两个分支。一个二叉树的整体结构如图 7-16 所示。

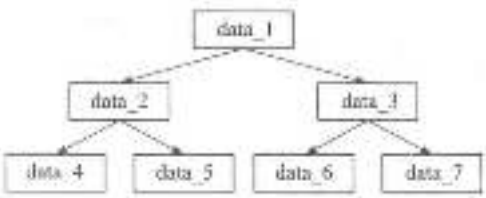


图 7-16 树结构

在处理某些查找问题上二叉树是非常有用的。例如，假设一组有序数据存储在一棵二叉树中，并且较小的值存储在左侧分支，较大的值存储在右侧分支。这样一来，判断某个特定值是否在此数列中就会非常高效。首先查看根节点，同要查找的值相比较。比较的结果就能立即决定接下来应该查找哪一分支，从而将搜索的范围缩小到 1/2。而在分支树上的比较结果又能将搜索范围继续缩小至 1/4 树，以此类推。图 7-17 中展示了一次这样的搜索过程。假设现在要确定数字 189 是否包含在树中。从根节点开始，由于 189 大于 157，可以判定接下来要在右子树中查找。右子树的根节点值为 208，大于 189，所以需要在节点 208 的左子树中查找。而遇到的节点 179 是该分支的末尾，因此可以判定数字 189 并不在此树中。如果要将节点 189 插入二叉树中，那么此处恰好就是插入的正确位置。

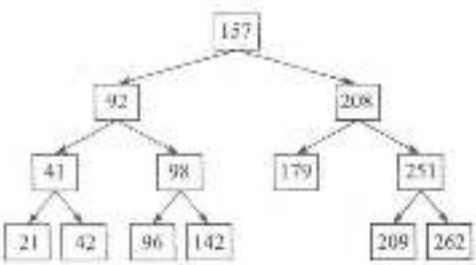


图 7-17 存储在树结构中的有序数列

364
↓
365

搜索二叉树、在树中插入和删除节点等操作需要利用到根节点和左右指针。图 7-18 展示了如何通过使用链表结构来实现一棵二叉树。

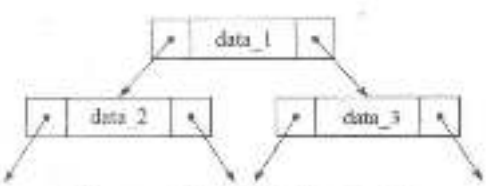


图 7-18 使用链表来实现二叉树

本章小结

结构体的出现使得定义一组数据变得非常便利，这些数据可以为同一类型，也可为不同类型。本章的前半部分给出了一些示例，展示了如何在 C 程序中使用结构体，后半部分介绍了动态数据结构和用链表实

观动态数据结构的算法。此外还讨论了其他几种动态结构，包括循环链表、双向链表、队列、栈和树。

关键术语

binary tree（二叉树）
circularly linked list（循环链表）
data member（数据成员）
doubly linked list（双向链表）
dynamic data structures（动态数据结构）
empty list（空表）
FIFO structure（先进先出结构）
head（表头）
LIFO structure（后进先出结构）
linked list（链表）
node（节点）
pointer operator（指针运算符）
queue（队列）
stack（栈）
structure（结构体）
structure member operator（结构成员运算符）
tag（标签）

C语句总结

定义结构体：

```
struct tsunami
{
   int mo, da, yr, fatalities;
   double max_height;
   char location[20];
};
struct node
{
   int data;
   struct node *link;
};
```

结构体声明：

```
struct tsunami t1;
struct node n1, n2={2,NULL};
```

访问数据成员：

```
t1.mo = 10;
n1.link = &n2;
```

结构体数组：

```
struct tsunami t2[100];
```

注意事项

1. 结构体的定义通常都位于在main函数之外的.h文件里。

调试注意事项

1. 定义结构体不会分配内存空间。如果要分配内存，则应该在结构体定义之后声明变量。
2. 要引用单独的数据成员，需要将结构体成员运算符同该结构体的变量名一同使用。
3. 如果不用结构体成员运算符，单独使用结构体变量名会引用整个结构体。
4. 关系运算符不能用在整个结构体上。

习题

简述题

判断题

判断下列语句的正（T）误（F）。

1. 一个结构体可以拥有不同数据类型的数据成员。 T F

2. 如果要引用结构体的数据成员，应该使用结构体名称，后面再加上圆括号和一个偏移量。 T F

3. 如果要打印结构体 r1 的全部 4 个数据成员，可以使用下面的语句：

```
printf("The values of r1 are:  %f %f %f %f \n",r1);
```

T F 367

4. 要引用一个结构体的数据成员，应该使用结构体名称加上结构体成员运算符。 T F

5. 要访问结构体的数据成员，可以使用一个指向结构体的指针加上指针运算符。 T F

多选题

根据下面的结构体定义来回答 6 ~ 10 题。

```
struct computer
{
   char manufacturer[10];
   double price;
   int speed;
};
struct computer pc1, pc2;
```

6. 要对结构体 pc1 输入数据，以下哪个是正确的函数原型定义？（　　）

(a) void get_pc(computer pc1);　(b) void get_pc(struct computer pc1);

(c) void get_pc(struct computer *pc1);　(d) void get_pc(computer *pc1);

(e) void get_pc(computer *pc1);

7. 打印结构体 pc1，以下哪个是正确的函数原型定义？（　　）

(a) void print_pc(computer pc1);　(b) void print_pc(struct computer pc1);

(c) void print_pc(struct computer.pc1);　(d) void print_pc(struct *pc1);

8. 要对结构体 pc1 的数据成员 price 赋值 $800，下列哪项是正确的赋值语句？（　　）

(a) struct price = 800;　(b) struct computer.price = 800;

(c) struct pc1->price = 800;　(d) pc1->price = 800;

(e) pc1->price = 800;

9. 要定义一个指向结构体 pc1 的指针，下列哪组语句是正确的？（　　）

(a)
```
struct computer pc1, *ptr_pc1;
...
pc1 = ptr_pc1;
```

(b)
```
struct computer pc1, *ptr_pc1;
...
ptr_pc1 = pc1;
```

(c)
```
struct computer pc1, *ptr_pc1;
...
ptr_pc1 = &pc1;
```

(d)
```
struct computer pc1, *ptr_pc1;
...
ptr_pc1 = *pc1;
```

368

内存快照题

根据下面的结构体定义来回答 10 ~ 13 题。

```
struct date
{
    int month, day, year;
};
```

给出下列语句执行后，结构体 start_date 和 end_date 相应的内存快照。假设下面每条语句都是在前面语句的基础上继续执行的。

10. `struct date start_date, end_date;`

11. `start_date.month = 9;`

12. `start_date.year = 2005;`
    `end_date.year = start_date.year + 3;`

13.
```
if (end_date.month < 7)
    end_date.day = 1;
else
    end.date.day = 30;
```

编程题

飓风。在本章定义了一个表示飓风信息的结构体：

```
struct hurricane
{
    char name[20];
    int year, category;
};
```

下面的问题需要处理文件 storms2.txt，该文件包含了 1950 – 2002 年间全美的最强飓风记录。文件中的信息是按年份排序的。

14. 编写程序，从文件 storms2.txt 中读取信息。使用前面给出的结构体，计算并打印出飓风的平均等级。打印信息要按照如下格式输出：

```
Hurricane Summary for Strongest Hurricanes in 1950-2002
Average Category:  xx.x
```

注意：这里不需要使用数组。

15. 编写程序，从文件 storms2.txt 中读取信息。使用前面给出的结构体，打印出每个等级包含的飓风次数。打印信息要按照如下格式输出：

```
Hurricane Summary for Strongest Hurricanes in 1950-2002
Category      Hurricanes
1             x
2             x
3             x
```

369 注意：这里不需要使用数组。

16. 编写程序，从文件 storms2.txt 中读取信息。使用前面给出的结构体，打印发生在 1960 – 1969 年间的飓风信息。打印信息要按照如下格式输出：

```
Strongest Hurricanes between 1960 and 1969
Name  Year  Category
```

注意：这里不需要使用数组。

17. 编写程序，从文件 storms2.txt 中读取信息。使用前面给出的结构体，打印出发生在用户输入的两个年份之间的飓风信息。打印信息要按照如下格式输出：

```
Hurricane Summary for Strongest Hurricanes between x and x
Name  Year  Category
```

注意：这里不需要使用数组。

18. 编写程序，从文件 storms2.txt 中读取信息。使用前面给出的结构体，按照等级由大到小的顺序依次打印飓风信息，如首先打印等级为 5 的飓风信息，随后打印等级为 4 的飓风信息，以此类推。

```
Strongest Hurricanes between 1950 and 2002
Category  Number of Hurricanes
```

直接将信息读入数组中，然后分多次扫描数据以打印信息，但是不要对数组进行排序。

19. 编写程序，从文件 storms2.txt 中读取信息。使用前面给出的结构体，按照字母表顺序依次打印飓风名称：

```
Strongest Hurricanes between 1950 and 2002
Hurricane Name
```

你可能会需要浮习前面有关数组排序的内容（第 5 章）。

20. 编写程序，从文件 storms2.txt 中读取数据。使用前面给出的结构体，按照字母表顺序依次打印飓风名称和其他相关信息：

```
Strongest Hurricanes between 1950 and 2002
Hurricane  Year  Category
```

提示：首先解决 19 题，然后再修改此程序，使得每次交换名称数组的值时，也同时交换年份数组和等级数组中相应位置上的值。

海啸。在本章定义了一个表示海啸信息的结构体：

```
struct tsunami
{
   int mo, da, yr, fatalities;
   double max_height;
   char location[20];
};
```

370

下列问题需要处理文件 waves2.txt，该文件中包含了从 20 世纪 90 年代至今最大的海啸记录。文件中的信息是按照日期排序的。

21. 编写程序，从文件 waves2.txt 中读取信息。使用前面给出的结构体，打印出每年发生的海啸次数。打印信息按照如下格式输出：

```
Information for Large Tsunamis from the 1990s
Year   Number of Tsunamis
xxxx   xx
```

注意：在这里不需要使用数组。

22. 编写程序，从文件 waves2.txt 中读取信息。使用前面给出的结构体，打印出每年的海啸死亡人数。打印信息按照如下格式输出：

```
Information for Large Tsunamis from the 1990s
Year  Number of Fatalities
xxxx  xx
```

注意：在这里不需要使用数组。

23. 编写程序，从文件 waves2.txt 中读取信息。使用前面给出的结构体，打印出每年发生海啸的位置信息。打印信息按照如下格式输出：

```
Information for Large Tsunamis from the 1990s
Year and Locations
1992
     Nicaragua
     Flores_Island
1993
...
```

注意：在这里不需要使用数组。

24. 编写程序，从文件 waves2.txt 中读取信息。使用前面给出的结构体，并通过键盘输入一个年份。打印出该年份中发生的所有海啸信息。打印信息按照如下格式输出：

```
Information for Large Tsunamis from the 1990s
Date        Location   Maximum Wave (m)      Fatalities
```

注意：在这里不需要使用数组。

25. 编写程序，从文件 waves2.txt 中读取信息。使用前面给出的结构体，打印出发生海啸次数最多的年份及相关信息。打印信息按照如下格式输出：

```
Information for Large Tsunamis from the 1990s
Year xxxx had maximum number of xxx tsunamis.
```

371 注意：在这里不需要使用数组。

C++ 编程语言简介

犯罪现场调查：手部识别

手部识别系统通常应用于访问控制。例如，纽约尼斯国际机场是用手部识别来代替传统的检票。很多体育馆也使用手部识别来代替刷卡入馆。这类系统在对安全性要求不高的应用场景中可以大量使用。对于这些大流量的应用，手部识别可算是一项经济性的生物识别技术，因为它可以快速检测，而且计算也不复杂。然而，在识别准确度上，手部识别远不如指纹识别和虹膜识别。因此在一些需要高精度身份认证的应用中，手部识别并不常用。手部识别的一种方法是，被识别者将一只手放入一个装置中，装置内的凹槽相应可以将手指分开并固定。随后识别系统分别测量手指的长、宽、高，以及手指关节间的距离。实际上，大多数手部识别系统会进行约 100 项测量。利用这些测量结果，系统可以将被识别者手部的测量信息同数据库中存储的信息相比较。如果能在数据库中找到一组足够接近的手部信息，则被识别者将被授权进入系统。反之，如果没能找到足够相近的匹配数据，那么可能就需要再提供其他验证信息以获取准入权限。8.5 节设计了一个 C 函数，能够对手部识别系统中获取的手部测量数据进行分析。

372

学习目标

在本章，我们将学到以下解决问题的方法：

- C++ 标准输入 / 输出对象。
- C++ 文件输入 / 输出对象。
- C++ 用户自定义类。

8.1 面向对象编程

AT&T 贝尔实验室的 Bjarne Stroustrup 在 20 世纪 80 年代初开发出了 C++ 编程语言。C++ 就是在 C 语言的基础上增加了额外特性以支持面向对象编程（object oriented programming）。C++ 支持所有 C 语言运算符和控制结构，还支持函数的定义和使用。C++ 语言是 C 语言的扩展，所以大多数的 C 程序都能用 C++ 编译器编译执行。反之则不正确：

C编译器无法识别C++程序所带有的额外特性。

面向对象编程需要的是在思考解决方案时寻求思维方式的转变。在设计面向对象程序时，首先要找到问题中的关键元素。同时，要考虑这些元素是如何定义、创建、修改乃至应用在程序中的。由于我们找出的关键元素无法用已知的数据类型表示，于是便自己定义数据类型。在C++中，使用类来定义新的数据类型。

面向对象编程的主要特性包括类、对象、多态性和继承。类（class）是用户自定义的数据类型，它包含了数据和操作这些数据的函数。对象（object）是自定义类型的变量，或者说是由类生成的实例。类对象可以调用类定义的函数。这些函数称为成员函数（member function），专门对调用函数的对象数据成员进行操作。多态性（polymorphism）是对同一名称赋予多重意义。函数重载就是C++多态性的一个例子。同一个函数名可以被定义成多种形式。程序执行时，系统可以决定调用哪一种函数定义。继承（inheritance）是指一个类可以继承另一个已知类的特性。这个新类（可以叫作子类或派生类）继承了已知类（可以叫作父类或基类）的所有成员数据和函数，并且可以自己定义新的数据成员和成员函数。继承是面向对象设计的核心概念，但在本章暂时不会深入讨论。

373

8.2 C++程序结构

本节主要分析比较C++程序和C程序基础结构的区别。在第1章曾给出一个程序chapter1_1，用来计算并打印两点间的距离。现在将该程序转换成C++程序，代码如下：

```
//-----------------------------------------------------------------
//   程序 chapter8_1
//
//   该程序计算两点间的距离

#include <iostream>
#include <cmath>
using namespace std;

int main(void)
{
   //   声明变量并初始化
   double x1=1, y1=5, x2=4, y2=7,
          side_1, side_2, distance;

   //   计算直角三角形的边
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);

   //   打印距离
   cout.setf(ios::fixed);
   cout.precision(2);
   cout << "The distance between the two points is "
        << distance << endl;

   //   退出程序
   return 0;
}
//-----------------------------------------------------------------
```

C++中有很多新的include文件。其中文件iostream是标准输入/输出流，在下一节中将会详细讨论。

在C++中，可以用/*和*/来标注多行注释，而单行注释可以用双斜杠（//）来分隔。

双斜杠可以用在程序的任一行，并且它右侧的所有内容都会被当作一条注释。 374

8.3 输入和输出

C++ 使用预定义的对象 cin（读作 see-in）来实现标准输入，用预定义的对象 cout（读作 see-out）来实现标准输出。这些对象定义在头文件 iostream 中。如果要在程序中使用任意一个预定义对象，则必须使用下面的预处理命令：

```
#include <iostream>
```

该命令包含了 ostream 和 istream 的类定义，程序中使用的 cout 是 ostream 类的对象，cin 是 istream 类的对象。除此之外，文件中还有一些使用 cin 和 cout 对象所需的其他信息。

使用 math 库的预处理命令是：

```
#include <cmath>
```

除此之外，还有一个指令告诉编译器去使用声明在命名空间 std 中的库文件名，该指令应该同相应的预处理命令一同使用：

```
using namespace std;
```

8.3.1 cout 对象

cout 对象被定义为到标准输出设备的输出流。流（stream）形象地表述了程序生成的一串连续字符被顺序输送至输出设备的过程。流插入运算符（insertion operator，<<）应该与 cout（或其他 ostream 对象）一同使用。与插入运算符相结合的任意值都可能被输送至输出设备中。举例来说，假设标准输出设备是计算机屏幕，那么下列语句会将 4 个值输出至屏幕：

```
cout << "The radius of the circle is " << radius << " centimeters"
     << endl;
```

任何要输出的值都需要在前面加上 << 运算符。在前面的例子中，输出的第一个值是字符串 "The radius of the circle is"，输出的第二个值是变量 radius，输出的第三个值是字符串 " centimeters"，输出的第四个值是预定义的流控制符 endl。endl 向输出流中插入一个换行符，换行符可以启动一个新的行用于后续内容显示，并且可以让输出流中的信息立即显示出来。

下面的例子则是 cout 的另一种用法：

```
double radius=10, area;
const double PI=3.141579;
...
cout << "The radius of the circle is: " << radius << " centimeters"
     << endl
     << "The area is " << PI*radius*radius << " square centimeters"
     << endl;
```

在这个例子中，使用了限定符 const 来声明一个名叫 PI 的常量。这些语句的输出结果为：

```
The radius of the circle is: 10 centimeters
The area is 314.158 square centimeters
```

要注意的是，虽然变量 radius 是 double 数据类型，但显示的 radius 值是 10 而不是 10.0。这是 cout 的默认输出格式，可以通过使用流函数和流控制符来控制 C++ 程序中的输出格式。 375

8.3.2 流函数

前面讲过，cout 是 ostream 类的对象。而流函数是 ostream 类的成员函数，并且可以被 ostream 类对象调用。当类对象调用成员函数时需要使用一个特殊的运算符，叫作点运算符（.）。在下面的例子中会展示一些可以用作格式化输出的流函数和格式标志（format flag）。setf 函数用来设置输出流的格式标志，例如 ios::fixed 就是一种格式标志。当系统设置了 ios::fixed 标志时，precision 函数可以指定小数点右侧打印出多少位；如果没有设置 ios::fixed 标志，precision 函数指定的则是显示的有效数字的个数。表 8-1 展示了几种更常用的格式标志。

```
double radius = 10, area;
const double PI=3.141579;

//设置格式标志
cout.setf(ios::fixed);  // cout 调用 setf 函数
cout.setf(ios::showpoint);
cout.precision(2);      //设置精度
...
cout << "The radius of the circle is: " << radius << " centimeters"
     << endl
     << "The area is " << PI*radius*radius << " square centimeters"
     << endl;
```

上面语句的输出结果如下：

```
The radius of the circle is: 10.00 centimeters
The area is 314.16 square centimeters
```

表 8-1 常用格式标志

标 志	含 义
ios::showpoint	显示小数点
ios::fixed	十进制计数
ios::scientific	科学计数法
ios::right	右对齐打印
ios::left	左对齐打印

练习

假设 int 型变量 sum 的值是 150，double 型变量 average 的值是 12.368。写出下列代码段的输出结果：

1. `cout << sum << average;`
2.
```
cout << sum;
cout << average;
```
3. `cout << sum << endl << average;`
4.
```
cout.precision(2);
cout << sum << endl << average;
```
5.
```
cout.setf(ios::showpoint);
cout.precision(3);
cout << sum << ',' << average;
```
6.
```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout << sum << ',' << average;
```

376

8.3.3 cin 对象

cin 对象被定义为从标准输入设备中读取输入流。举例来说，假设标准输入设备是键盘。那么流提取运算符（>>）与 cin（或其他 istream 对象）一同使用，能够获取输入数值并对变量赋值。该运算符 >> 会自动忽略所有空白（比如空格、缩进和换行符）。下列语句是从键盘输入三个数值：

```
cin >> var1 >> var2 >> var3;
```

cin 语句会等待用户输入。在前面的例子中，从键盘输入的第一个值会被赋给变量 var1，第二个值赋给 var2，第三个值赋给 var3。直到按下回车键，程序才会读取输入值。在 cin 语句中允许输入退格和修改。从键盘输入的值必须用空格分隔，并且对空格的数量不做要求。cin 语句会一直忽略空格，直到它接收到每个变量的值。此外，输入的值必须与 cin 语句中变量的数据类型相匹配。

下面通过一个例子来说明 cin 的用法：

```
int id;
double rate, hours;
char code;
...
cin >> rate >> hours >> id >> code;
cout << rate << endl << hours << endl << id << endl
     << code << endl;
```

假设该键盘的输入流包含如下两行数值：

```
10.5   40
556    r
```

那么输入语句中的变量将被赋予如下值：

rate	10.5
hours	40
id	556
code	'r'

377

cout 语句会在屏幕上打印出如下信息：

```
10.5
40
556
r
```

cin 不需要 scanf 使用的格式说明符，流输入运算符 >> 会根据它后面变量的数据类型来接收并解释输入值。同时，>> 运算符也会忽略所有空格。但是对有些需要输入字符数据的应用来说，空格是不能忽略的。在这种情况下，应该使用流输入（istream）对象的成员函数 get，从输入流中获取单个字符。语句

```
char ch;
cin.get(ch);
```

会从键盘读入下一个字符，并将该字符赋给变量 ch。其中，get 函数并不会忽略掉空格，

它会将空格看作一个合法字符数据。

8.3.4 定义文件流

到目前为止，本节介绍了如何使用 cin 来读取键盘的输入数据，使用 cout 向屏幕打印输出数据。如果要从一个文件中读取数据，或者将信息打印至文件，就必须要定义一个文件流（file stream）对象，并将这个对象与一个文件相关联。

C++ 提供了两个文件流类：ifstream 类用来定义对象以实现文件流输入，ofstream 类用来定义对象以实现文件流输出。ifstream 和 ofstream 这两个类被定义在头文件 fstream 里。下面的声明语句定义了这两个类对象：

```
ifstream indata;    //定义 indata 作为输入文件流对象
ofstream outdata;   //定义 outdata 作为输出文件流对象
```

在文件流对象定义完成后，接下来可以通过成员函数 open 打开指定的文件，文件名为参数，使对象与这个指定文件建立关联。为了说明这个过程，下列语句将数据文件分别同对象 indata 和 outdata 相关联：

```
indata.open("sensor1.txt");      //打开文件 sensor1 作为输入文件
outdata.open("plot1.txt");       //打开文件 plot1 作为输出文件
```

上面的语句定义了对象 indata 和 outdata，并且分别将这两个对象同数据文件相关联，现在就可以按照类似 cin 和 cout 的方式，使用 indata 和 outdata 来实现输入和输出操作了。假设文件 sensor1 中包含了一些实验数据，那么可以使用下面的语句从文件中读入一个值：

```
indata >> x;
```

然后可以将 x 的值连同 x 的自然对数以及 e^x 一同输出至文件 plot1.txt 中。实现该过程的语句如下：

```
outdata << x << " " << log(x) << " " << exp(x) << endl;
```

378 打印的空格是为了将各输出值分隔开。

函数 close 用于文件使用完毕后将其关闭。该函数由文件流对象调用。在这个例子中，可以通过如下语句来关闭这两个文件：

```
indata.close();
outdata.close();
```

当打开数据文件进行输入操作时，最好先提前验证 open 函数是否成功运行。如果 open 函数没能正常打开文件，则不会显示任何错误信息，但是所有读取文件的尝试都会失败。成员函数 fail 可以确定 open 函数是否成功运行，成员函数 eof 则可以确定是否浏览到文件末尾，这些函数的用法将在下节介绍。

8.4 C++ 编程范例

区分 C 和 C++ 的一个好办法是用它们解决同一问题，其中一个程序用 C 语言编写，另一个程序用 C++ 编写，然后将得到的两个程序进行比较。在本节中，我们会针对前面已经用 C 程序解决的几个问题，分别给出对应的 C++ 程序。同时，还会标出相应的 C 程序所在的章节，以方便读者对两种程序进行比较。这些程序的主要差异其实都在输入和输出语句中。

8.4.1 简单计算

在 2.10 节中编写了一个程序来计算功率水平发生变化后飞机的速度和加速度值。现在，将下面的 C++ 程序和 2.10 节的 C 程序 chapter2_4 进行比较：

```
//-----------------------------------------------------------
//  程序 chapter8_2
//
//  该程序计算飞机在指定时间下的速度和加速度值

#include <iostream>
#include <cmath>
using namespace std;

int main(void)
{
   //  声明变量
   double time, velocity, acceleration;

   //  从键盘读入时间值
   cout << "Enter new time value in seconds:" << endl;
   cin >> time;

   //  计算速度和加速度值
   velocity = 0.00001*pow(time,3) - 0.00488*pow(time,2)
              + 0.75795*time + 181.3566;
   acceleration = 3 - 0.000062*velocity*velocity;
   //  输出速度和加速度值
   cout.setf(ios::fixed);
   cout.precision(3);
   cout << "Velocity = " << velocity << " m/s" << endl;
   cout << "Acceleration = " << acceleration << " m/s^2" << endl;

   //  退出程序
   return 0;
}
//-----------------------------------------------------------
```

379

8.4.2 循环

3.5 节中编写了一个函数来将角度值转换为弧度值。现在，将下面的 C++ 程序同 3.5 节的 C 程序 chapter3_4 进行比较：

```
//-----------------------------------------------------------
//  程序 chapter8_4
//
//  该函数使用循环结构输出从角度值到弧度值的转换表格

#include <iostream>
using namespace std;

int main(void)
{
   //  声明常量和变量
   const double PI=3.141593;
   int degrees;
   double radians;

   //  循环输出弧度和角度
   cout.setf(ios::fixed);
```

```
    cout.precision(6);
    cout << "Degrees to Radians" << endl;
    for (degrees=0; degrees<=360; degrees+=10)
    {
       radians = degrees*PI/180;
       cout << degrees << " "
            << radians << endl;
    }
    //  退出程序
    return 0;
}
//-----------------------------------------------------------------
```

8.4.3 函数、一维数组和数据文件

在5.1节中编写了一个程序，从文件中读取100个数据值，然后确定其中的最大值，其中寻找数组最大值这一步骤使用函数来完成。现在，将下面的C++程序同5.1节的C程序chapter5_2进行比较：

[380]

```
//-----------------------------------------------------------------
//  程序chapter8_4
//
//  该程序从文件中读取数据，并利用函数来确定其中的最大值

#include <iostream>
#include <fstream>
using namespace std;
#define FILENAME "lab2.txt"

int main(void)
{
   //  声明变量和函数原型
   const int N=100;
   int k=0, npts=N;
   double y[N];
   double max(double x[], int n);
   ifstream lab;
   //  打开文件，将数据读入数组
   lab.open(FILENAME);
   if (lab.fail())
      cout << "Error opening input file." << endl;
   else
   {
      while (!lab.eof())
      {
         lab >> y[k];
         k++;
      }
      npts = k;
      //  找到并输出最大值
      cout << "Maximum value: "
           << max(y,npts)) << endl;
      //  关闭文件并退出程序
      lab.close();
   }
   //  退出程序
   return 0;
}
```

```
//-----------------------------------------------------------------
//  该函数返回包含n个元素的数组x中的最大值
double max(double x[],int n)
{
   //  声明变量
   int k;
   double max_x;
   //  确定数组中的最大值
   max_x = x[0];
   for (k=1; k<=n-1; k++)
      if (x[k] > max_x)
         max_x = x[k];
   //  返回最大值
   return max_x;
}
//-----------------------------------------------------------------
```

381

8.5 解决应用问题：手部识别

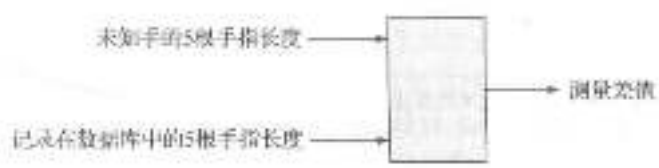
本节将使用前面介绍的C++语句来解决手部识别问题。为简单起见，这里仅使用5个测量值，即右手五根手指的长度。设计函数，函数的输入为待识别的手部信息和数据库中的手部信息。该函数要计算两个手部信息的对应手指长度差的绝对值之和。这个计算结果代表了两只手之间的差异。识别程序会通过该函数将未知的手部信息同数据库中的每组记录一一进行比较，然后从中挑选出最接近的手部信息。如果计算得到的差异小于指定的阈值，则认为该未知身份与数据库信息相匹配，并允许进行后续操作。同时，还要编写程序来测试该函数。

1. 问题陈述

编写函数，计算两个手部测量值间的差异值之和。

2. 输入/输出描述

下面的I/O图显示，该函数的输入是未知手的5根手指长度和记录在数据库中的5根手指长度，函数输出是测量差值。



382

3. 手动演算示例

假设下面就是手部测量值，单位为厘米：

	未知的手部信息	数据库中的记录	差值
大拇指	5.4	6.2	0.8
食指	7.2	7.0	0.2
中指	7.9	8.0	0.1
无名指	7.4	7.4	0.0
小指	5.1	5.8	0.7

计算得到测量差值之和为1.8。

4. 算法设计

首先要设计分解提纲，将解决方案分解成一组可以顺序执行的操作步骤。

函数的分解提纲

1）从函数的输入信息中获取手指数据。

2）计算差值总和。

3）返回差值总和。

同时，还需要设计程序来测试该函数，以下为测试程序的分解提纲。

测试程序的分解提纲

1）指定未知手和数据库记录中手的手指长度。

2）使用函数计算测量差值。

3）输出测量差值。

这些算法比较简单，可以直接将分解提纲转化为 C++ 代码。

```
//-----------------------------------------------------------------
//   程序 chapter8_5
//
//   该程序计算并输出两个手部测量值之间的差值
#include <iostream>
#include <cmath>
using namespace std;
int main(void)
{
   //声明并初始化变量
   double unknown[5]={5.4,7.2,7.9,7.4,5.1},
          known[5]={6.2,7.0,8.0,7.4,5.8};
   double distance(double hand_1[5],double hand_2[5]);

   // 计算并输出差值
   cout << "Distance: " << distance(unknown,known) << endl;

   // 退出程序
   return 0;
}
//-----------------------------------------------------------------
//   该函数计算两个手部测量值之间的差值
double distance(double hand_1[5],double hand_2[5])
{
   //   声明变量
   int k;
   double sum=0;

   //   计算差值的绝对值之和
   for (k=0; k<=4; k++)
      sum = sum + fabs(hand_1[k]-hand_2[k]);

   //   返回测量差值
   return sum;
}
//-----------------------------------------------------------------
```

383

5. 测试

测试程序按照手动演算示例中使用的数据将未知手和已知手的测量值进行了初始化。

以下为程序的输出结果：

```
Distance: 1.8
```

这结果同手动演算示例中的结果完全相同，所以现在可以改变测试程序中的输入，利用其他数据来继续进行测试。可以用班里所有学生的数据对程序进行测试。抽取一个学生的测量数据作为程序中的未知值，并将这个值同全班的测量数据进行比对，看看是不是本人的数据得出的测量差值最小。此外再考虑一个问题，在计算差值的函数中所使用的手指的顺序会对结果造成影响吗？答案肯定是没有影响。但是建议读者在程序中改变手指的顺序并重新进行计算，以此来证明该结论。最后，如果计算中使用的只是差值之和，而不是差值的绝对值之和，又会得到怎样的结论？ [384]

修改

本节主要讨论了关于比较手部测量值的问题。而以下这些问题便是由此产生的。

1. 修改程序，使其从键盘获取未知测量值。
2. 修改程序，使其从数据文件获取已知测量值，并使用循环来对未知测量值和数据文件中的所有测量值进行比较。
3. 修改问题 2 中的程序，使其输出最小差异值。
4. 修改问题 3 中的程序，使其输出最小差异值的数据编号，比如“Known 4 has best match”。
5. 修改问题 4 中的程序，使其输出差异值等于最小值的所有数据记录。

8.6 解决应用问题：地表风向

地表风驱动着海洋的表层流向，而水的密度变化则驱动着海洋的深层流向。地表风可以由卫星测量，它们的风向趋势形成了地球表面的主要风带。在北半球，信风通常是由东北吹向西南；而在南半球，信风则是通常由东南吹向西北（这些风之所以被称为信风，是由于海洋的航运路径是根据它们的风向来确定的）。信风和赤道之间的边界通常被称为“赤道无风带”，因为这片区域几乎无风，这对于帆船来说无疑是令人沮丧的，因为通过赤道无风带会非常缓慢。在本节中，我们要设计一个程序来读取数据文件，文件中包含有某片海域中地表风向的数据。该程序要确定地表风的主要风向，并计算该风向的数据占文件中所有风向记录的百分比。

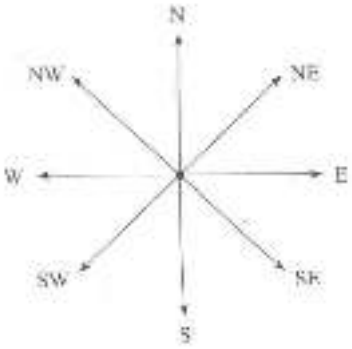


图 8-1 指南针方向

我们将海面划分成为网格，在数据文件中存储每一个格子的风向。例如将一片区域划分成 5×5 的网格大小，网格的每一行都在信息文件 wind1.txt 中占单独一行。图 8-1 展示了指南针的 8 个方向，为了记录方便，按照表 8-2 所示对风向信息进行编码。 [385]

表 8-2 风向编码

风向	编码
N	1
NE	2

（续）

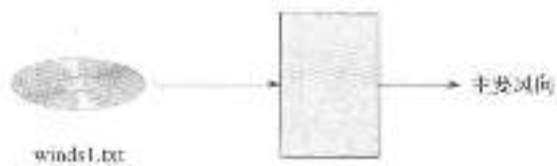
风向	编码
E	3
SE	4
S	5
SW	6
W	7
NW	8

1. 问题陈述

读取一个海岛网格的风向数据，确定并输出主要风向和该风向出现的次数。

2. 输入/输出描述

下面的 I/O 图显示，函数输入为数据文件，输出为报告信息。



3. 手动演算示例

假设网格包含以下信息：

4	4	5	5	4
3	4	4	4	5
4	5	4	4	4
4	4	4	4	4
4	4	4	4	5

在 25 个风向数据中，数字 4 出现了 19 次，占总数据的 76%。因此程序应该具有如下输出结果：

386

```
The wind is blowing from the SE 76% of the time.
```

4. 算法设计

首先，设计分解提纲，将解决方案分解为一组可以顺序执行的操作步骤。

分解提纲

1）将风向数据读入数组，并确定出现次数最多的风向值。

2）计算并输出该风向出现次数的百分比。

[测试后的伪代码]

```
主函数：if文件不能被打开
        输出错误信息
        else
          读取数据并确定每个网格点的风向
          确定出现次数最多的风向
          计算并输出出现次数最多的风向
```

伪代码中的步骤足够详细，可直接将其转换为C++程序：

```
//-----------------------------------------------------------------
//  程序chapter8_6
//
//  该程序从数据文件中读取风向信息，然后确定出现次数最多的风向值，计算并输出
//  具有该风向的点所占的百分比
#include <iostream>
#include <fstream>
using namespace std;

int main(void)
{
   //  声明变量
   int r, c, k, maxk=0;
   int grid[5][5], category[8]={0,0,0,0,0,0,0,0};
   double perc;
   char* direction[8]={"N  ","NE ","E  ","SE ","S  ",
                       "SW ","W  ","NW "};
   ifstream winds;

   //  读取并输出文件中的信息
   winds.open("winds1.txt");
   if (winds.fail())
      cout << "Error opening input file." << endl;
   else
   {
      for (r=0; r<=4; r++)
         winds >> grid[r][0] >> grid[r][1] >> grid[r][2]
               >> grid[r][3] >> grid[r][4];

      //  确定每个风向的出现次数
      for (r=0; r<=4; r++)
         for (c=0; c<=4; c++)
         {
            k = grid[r][c];
            category[k]++;
         }
      //  确定出现次数最多的风向
      for (k=0; k<=7; k++)
         if (category[k] > category[maxk])
            maxk = k;
      //  输出报告
      cout.setf(ios::fixed);
      cout.precision(1);
      perc = (double)category[maxk]/25*100;
      cout << "The wind is blowing from the "
           << direction[maxk-1]
           << perc << "% of the time." << endl;
      //  关闭文件
      winds.close();
   }
   //  退出程序
   return 0;
}
//-----------------------------------------------------------------
```

387

5. 测试

使用手动演算示例中的数据作为输入，程序的输出结果应该如下所示：

```
The wind is blowing from the SE 76 % of the time.
```

修改

本节主要讨论了关于确定风向的问题，否以下这些问题便是由此产生的。

1. 修改程序，输出每种风向出现的次数，输出时需使用风向符号（比如 SE）。
2. 修改程序，如果出现次数最多的方向值不止一个，则逐行打印出相应的统计结果。
3. 修改程序，如果在指南针坐标系中四个象限的风向同时出现在了一次记录中，则输出信息"Possible cyclone or hurricane"（可能会出现龙卷风或飓风）。
4. 修改程序，在屏幕上打印一个网格和风向的示意图。例如，如果风从西面（W）吹来，则输出符号'>'；如果风从东面吹来，则输出符号'<'。为8个风向分别选择对应的符号来表示。
5. 修改程序，从数据文件的第一行读取网格大小（即行数和列数）。假设网格最大为100行×100列。

388

8.7 类

在C++中，类是面向对象编程的基础。类与结构体类似，但不同的是，除了数据成员之外，类还包含成员函数。一个设计良好的类可以直接被用作预定义数据类型。类的实例称为对象。在前面的章节里，已经遇到过使用预定义类和对象的例子。例如，使用cin对象和cout对象来执行标准输入/输出操作。还调用了这些对象的函数，如precision和setf。而在本节，将专门对用户自定义的类进行讨论。

8.7.1 定义类数据类型

类的定义包含两部分：类声明（class declaration）和类实现（class implementation）。下面将分别进行讨论。

在类声明中，类的名称由关键字class来指定。类声明的主体由数据成员（data member）的类型声明语句和成员函数的原型语句组成。假设现在要定义一个数据类型来表示直角坐标系中的点，那么这个数据类型应该表示为一对数字，分别是x坐标和y坐标，如图8-2所示。在设计一个类时，需要考虑用哪些数据成员来表示相应的数据类型，以及对于该数据类型应该定义何种操作。

389

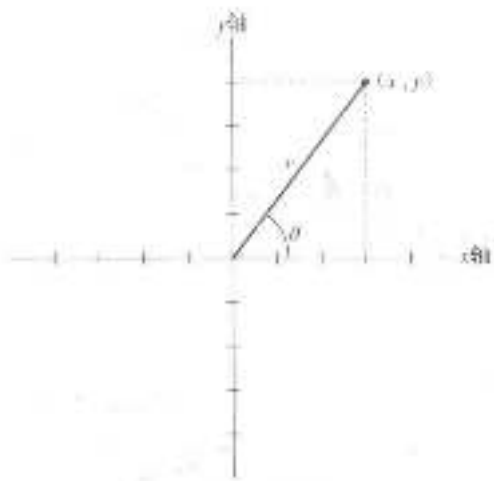


图8-2 数据点的直角坐标

类的设计过程也在类声明中完成，其中包括定义成员函数来实现各种操作。假设我们声明一个类xy_coordinate用于描述坐标系中的点，那么它应该至少由两个数据成员和两个成员函数组成，其中数据成员用来描述坐标点，而成员函数则用来计算半径r和角度θ，如图8-2所示。本节后面的内容里，还将继续对类xy_coordinate加入其他成员函数。类的声明部分通常被保存在头文件中，如下列代码所示：

```
//-----------------------------------------------------------------
//   类的声明（版本1）
//
//   下列语句定义一个类xy-coordinates
//   该声明语句存储在头文件xy_coordinate.h中
```

```
#include <iostream>
#include <cmath>

class xy_coordinate
{
   //  定义公有成员的函数原型
   public:
      void input()
      void print()
      double radius();
      double angle();

   private:
   //  定义私有数据成员
      double x, y;
};
//-----------------------------------------------------------------
```

类 xy_coordinate 具有 2 个数据成员和 4 个成员函数。关键字 public 和 private 负责对控制类成员的访问权限。其中，public 用于指定公有成员（public member），即可以在程序的任何地方被引用的成员。private 用于指定私有成员（private member）。私有成员只能被类 xy_coordinate 的成员函数所引用。为了遵循面向对象的设计原则，建议所有的数据成员都被指定为私有成员。这种受限制的访问模式称为信息隐藏。如果类的数据成员结构和用法发生变化，那么仅需修改成员函数，而无需对用户程序做任何改动。此外，任何私有成员函数只能被同一个类的其他成员函数所引用。由于设计私有成员函数主要是为了帮助其他成员函数完成操作，所以它们通常被称作辅助函数。

类的实现是指实现所有的成员函数定义。在定义一个成员函数时，需要用到范围限定运算符（::）。这个操作符放在类名和函数名之间，用来指定该函数是类的一个成员。在对成员函数进行定义时，首先要指定函数的返回值类型，然后便是类名、范围限定运算符、函数名以及参数列表。在设计成员函数时需注意，所有成员函数都可以直接访问数据成员，数据成员不必出现在参数列表中。此外，辅助函数也可以直接被成员函数调用。类的实现部分也可以被存储在一个单独文件中。下面就是对类 xy_coordinate 的实现过程。需要说明的是，函数 angle 的计算结果是以象限角。 390

```
//-----------------------------------------------------------------
//  类的实现（版本 1）
//
//  下列语句用来定义类 xy_coordinate 的实现。它们被存储在头文件 xy_coordinate.h 中

//  该函数从键盘读取 xy 坐标
void xy_coordinate::input()
{
   cin >> x >> y;
}
//  该函数在屏幕上输出 xy 坐标
void xy_coordinate::print()
{
   cout << "(" << x << "," << y << ")" << "\n";
}

//  该函数计算半径
double xy_coordinate::radius()
{
```

```
//   该函数计算以弧度表示的角度值
double xy coordinate::angle()
{
   //  计算极坐标的角度
      double z, pi=3.141593;
      if (x >= 0)
         z = atan(y/x);
      if (x<0 && y>0)
         z = atan(y/x) + pi;
      if (x<0 && y<=0)
         z = atan(y/x) - pi;
      if (x==0 && y==0)
         z = 0;
      return z;
}
//-----------------------------------------------------------------
```

现在用如下程序来测试这个新的数据类型。

```
//-----------------------------------------------------------------
//   程序 chapter8_7
//
//   该程序用来测试类 xy_coordinate 的功能
#include <iostream>
#include <cmath>
#include "xy_coordinate.h"
using namespace std;
int main(void)
{
   //  声明和初始化变量
   xy_coordinate pt1;

   //  读取输入点
   cout <<  "Enter x and y coordinates:"  << endl;
   pt1.input();

   //  输出直角坐标和极坐标
   cout.setf(ios::fixed);
   cout.precision(2);
   cout << "Coordinate in xy form:" << endl;
   pt1.print()
   cout << "Coordinate in polar form:" << endl;
   cout << "magnitude: " << pt1.radius() << endl;
   cout << "phase (in degrees): << pt1.angle()*180/3.141593 << endl;

   //  退出程序
   return 0;
}
//-----------------------------------------------------------------
```

391

为了执行该程序，需要确保类 xy_coordinate 的头文件和实现文件能够与主程序文件一同编译。将这些自定义类型以文件的形式分离出来，可以帮助用户建立自己的代码库。这些库同标准库一样（如 iostream），可以被许多应用程序使用。在使用用户自定义类型时，应用程序必须要包含声明文件，并将也要连接到对应的实现文件。现在使用这些文件来测试程序。下面是该程序的一个输出样例：

```
Enter x and y coordinates:
4 4
Coordinate in xy form:
(4.00,4.00)
Coordinate in polar form:
magnitude: 5.66
phase (in degrees): 45.00
```

8.7.2 构造函数

在定义一个变量时，通常会希望为该变量赋一个初值，比如

```
double sum=0;
```

如果在定义变量时没有为其赋初值，那么该变量的值是未知的，直到为它分配一个有效数值。构造函数（constructor function）是在声明类对象时被自动调用的特殊成员函数。它主要用来对将要定义的对象的数据成员进行初始化。当设计一个类时，应该具有一整套完整的构造函数。构造函数有三个特殊属性：

- 在声明一个类的对象时，构造函数被自动调用。
- 构造函数的名称就是类名。
- 构造函数没有返回值，返回值类型也不能用 void 来修饰。

392

为了说明构造函数的用法，现在为 xy_coordinate 类定义两个构造函数来对其成员进行初始化，其中一个将 x 和 y 初始化为 0，另一个可以在声明语句中为 x 和 y 赋值。在下列代码中，公有成员声明的前两条语句便是这两个构造函数的声明语句：

```
//-----------------------------------------------------------------
//   类的声明(版本2)
//
//   下列语句实现 xy_coordinates 类的定义
//   假设这些声明语句存储在头文件 xy_coordinate.h 中
//   update 是两个构造函数之外的补充函数

#include <iostream>
#include <cmath>
using namespace std;

class xy_coordinate
{
   //  声明两个构造函数和6个公有成员的函数原型
   public:
      xy_coordinate();
      xy_coordinate(double a, double b);
      void input();
      void print();
      double radius();
      double angle();

   //  声明私有数据成员
   private:
      double x, y;
};
//-----------------------------------------------------------------
```

每当使用下面语句定义一个类对象时，默认构造函数（default constructor）会被自动调用：

```
xy_coordinate pt1;
```

这行代码声明了类 xy_coordinate 的一个对象 pt1，并且 pt1 的数据成员被默认构造函数 xy_coordinate() 初始化。而当使用如下语句定义一个对象时，此时带参的构造函数便被自动调用：

```
xy_coordinate pt2(3,4);
```

对象 pt2 的数据成员被构造函数 xy_coordinate(double a,double b) 通过其参数

393 列表来完成初始化。因此，x 的值为 3，y 的值为 4。

构造函数属于用户自定义函数。下面是刚刚添加的两个构造函数的代码实现。这些代码需要加入到类的实现文件中。

```
//-----------------------------------------------------------------
//  该构造函数将 x 和 y 初始化为 0
xy_coordinate::xy_coordinate()
{
   x = 0;
   y = 0;
}

//  该构造函数将 x 和 y 初始化为参数值
xy_coordinate::xy_coordinate(double a, double b)
{
   x = a;
   y = b;
}
//-----------------------------------------------------------------
```

8.7.3 类运算符

赋值运算符可以用于同一个类的不同对象间的赋值操作。如果 pt1 和 pt2 都是类 xy_coordinate 的对象，那么下面的语句是合法的：

```
pt1 = pt2;
```

这条语句执行后，对象 pt1 的每一个数据成员都会被赋值为对象 pt2 中相应的数据成员。但是，C++ 中的算术运算符和关系运算符不能自动处理用户自定义的类型。以下的语句是非法的：

```
if (pt1 == pt2)        {非法比较}
```

但是，在类中可以定义一组专门用于对类对象进行操作的运算符。

运算符重载（overload）在 C++ 中具有重要作用，它可以使得用户自定义的数据类型同预定义数据类型一样方便使用。以 C++ 中定义的算术运算符为例，这些运算符被定义为可以对所有的预定义数据类型进行运算。然而，对于用户自定义的数据类型，算术运算符无法支持这样的类对象运算。但是在定义一个类时，可以为类对象定义一组专用的运算符，称为运算符函数。运算符函数和其他成员函数的定义方式基本一致，只是需要加上关键字 operator。跟在关键字 operator 后面的是重载函数名，也就是 C++ 预定义的若干运算符中的一个。要记住，只有预定义运算符才能重载。例如，无法定义一个新的运算符 ** 来进行幂运算，因为此运算符并不是 C++ 的预定义运算符。在下一节，我们将会借助复数类的示例
394 着重介绍重载运算符的使用，如为复数类重载算术运算符 +、-、*、/。

8.8 数值方法：复根

在科学和工程领域中需要使用复数来解决许多问题，尤其是在物理和电气工程中。因此，在 C++ 中定义一个复数类会非常有用。（一些编译器本身添加了复数类，但由于它不在标准库中，所以有时是无效的。）复数的形式为 $a + bi$，其中 i 是 $\sqrt{-1}$，a 和 b 是实数。复数的实部由 a 表示，虚部由 b 表示，因此可以使用一个有序实数对来表示复数。可以看到，复数的表示方法和上节中的 xy 坐标表示法非常相似。实际上，如果将 x 轴作为实轴，y 轴作为

虚轴，就能转换直角坐标得到复数表示法，如图 8-3 所示。

如果要从键盘输入一个复数，则需要输入两个数值，因此输入函数其实和 xy 坐标是一样的。但复数的输出则不同，因为显示过程中要有对 i 的处理。例如，用数对（3, 5）表示的复数需要输出为 3+5i。可是如果虚部是负数，那么输出应该是 3-2i，而不是 3+-2i。这种细节差异需要在负责输出的成员函数中小心处理。

两个复数之间进行算术运算，其运算结果也为复数。复数的算术运算法则不同于整数和实数。表 8-3 列出了复数的基本运算法则。

谈到复数，经常会研究它的幅度和相位，分别用 r 和 θ 表示，如图 8-3 所示。幅度和相位的定义同直角坐标系完全相同，因此也可以使用相似的代码来实现。

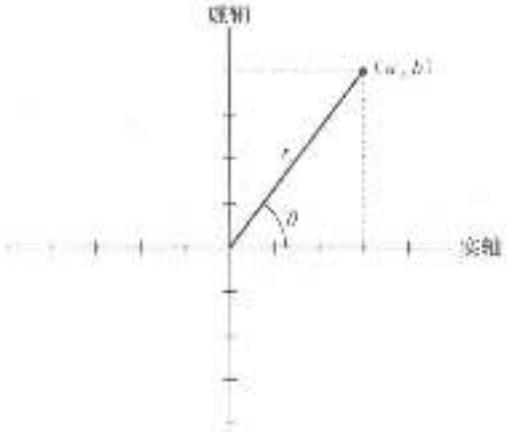


图 8-3 复数图解

395

表 8-3 复数算术运算

运算	结果
$c_1 + c_2$	$(a_1 + a_2) + (b_1 + b_2)i$
$c_1 - c_2$	$(a_1 - a_2) + (b_1 - b_2)i$
$c_1 * c_2$	$(a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)i$
c_1/c_2	$\frac{a_1a_2 + b_1b_2}{a_2^2 + b_2^2} + \frac{a_2b_1 - b_2a_1}{a_2^2 + b_2^2}i$
$(c_1 = a_1 + b_1i, c_2 = a_2 + b_2i)$	

8.8.1 复数类定义

接下来要讨论复数类的定义和实现。然后，在本节的最后编写程序来计算并输出二次方程式的复数根。

在类的声明中，复数的实部和虚部全部定义为私有成员，同时定义了一些公有成员函数，包括从键盘输入复数、在屏幕上输出复数，以及定义算术运算符。类的声明和实现将在下面的代码中表示。

请仔细阅读这些语句，并观察其中的实现步骤。但是要强调一点，这仅仅是对 C++ 能初步介绍，所以只针对这些例子给出了实现细节，并没有对类的声明和实现的原理做详细讨论。

复数类的声明和实现：

```
//-----------------------------------------------------------------
//   类的声明
//
//   定义一个复数类
//   (声明部分存储在头文件 complex.h 中)

#include <iostream>
#include <cmath>
using namespace std;
```

```
class complex
{
   //  声明公有成员函数原型
   public:
      complex();
      complex(double a, double b);
      void print();
      void input();
      double magn(complex);
      double angle(complex);
      complex operator+(complex);
      complex operator-(complex);
      complex operator*(complex);
      complex operator/(complex);
   //  声明私有成员
   private:
      double real, imag;
};
//-----------------------------------------------------------------
//  类的实现
//
//  复数类的实现部分

//  该函数是默认构造函数，对没有赋初值的复数进行初始化
complex::complex()
{
   real = 0;
   imag = 0;
}

//  该函数是将复数初始化为给定值的构造函数
complex::complex(double a, double b)
{
   real = a;
   imag = b;
}
//  该函数输出复数
void complex::print()
{
   if (imag > 0)
      cout << real << "+" << imag << "i" << endl;
   else
      if (imag == 0)
         cout << real << endl;
      else
         cout << real << imag << "i" << endl;
}

//  该函数读取复数的两个值
void complex::input()
{
   cin >> real >> imag;
}
//  该函数定义一个复数的和
complex complex::operator+(complex c)
{
   // 定义复数加法
   complex temp;
   temp.real = c.real + real;
```

396

```
    temp.imag = c.imag + imag;
    return temp;
}
//  该函数定义两个复数的差
complex complex::operator-(complex c)
{
    //  定义复数减法
    complex temp;
    temp.real = real - c.real;
    temp.imag = imag - c.imag;
    return temp;
}
//  该函数定义复数的乘积
complex complex::operator*(complex c)
{
    //  定义复数乘法
    complex temp;
    temp.real = (real*c.real - imag*c.imag);
    temp.imag = (imag*c.real + real*c.imag);
    return temp;
}
//  该函数定义复数的商
complex complex::operator/(complex c)
{
    //  定义复数除法
    complex temp;
    temp.real = (real*c.real + imag*c.imag)/
                (pow(c.real,2) + pow(c.imag,2));
    temp.imag = (imag*c.real - real*c.imag)/
                (pow(c.real,2) + pow(c.imag,2));
    return temp;
}
//-----------------------------------------------------------------------
```

197

现在用一个简单的例子来测试前面设计的复数类：在构建自己的类时，每写出一个成员函数都要立刻进行测试。调试一个复杂类的完整定义是非常困难的。

```
//-----------------------------------------------------------------------
//  程序 chapter8_8
//
//  该程序展示一个复数类的使用方法和运算操作

#include <iostream>
#include "ccomplex"
including namespace std;

int main(void)
{
    //  声明和初始化变量
    complex c1(4,1), c2(-3,-2), c3;
    //  输出初始值
    cout << "c1:"
    c1.print();
    cout << "c2:"
    c2.print();
    cout << "c3:";
    c3.print();
```

198

```
    //  计算并输出新值
    c3 = c1 + c2;
    cout << "c1+c2 = "
    c3.print();
    c3 = c1 - c2;
    cout << "c1-c2 = "
    c3.print();
    c3 = c1*c2;
    cout << "c1*c2 = "
    c3.print();
    c3 = c1/c2;
    cout << "c1/c2 = "
    c3.print();

    //  退出程序
    return 0;
}
//-----------------------------------------------------------
```

以下为程序的输出结果：

```
c1: 4+1i
c2: -3-2i
c3: 0
c1+c2 = 1-1i
c1-c2 = 7+3i
c1*c2 = -10-11i
c1/c2 = -1.07692+0.384615i
```

8.8.2 二次方程的复根

具有实系数的二次方程式的一般形式为：

$$ax^2 + bx + c = 0$$

该方程有两个根，可以通过下列公式计算：

$$\text{root}_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$\text{root}_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

399

平方根符号下的项，b^2-4ac，叫作判别式。如果判别式大于等于0，则方程的两个根都是实数，并且可以由如下表达式求出：

$$\text{root}_1 = -\frac{b}{2a} + \frac{\sqrt{\text{discriminant}}}{2a}$$

$$\text{root}_2 = -\frac{b}{2a} - \frac{\sqrt{\text{discriminant}}}{2a}$$

如果判别式小于0，则方程的两个根都是复数，并且可以由如下表达式求出：

$$\text{root}_1 = -\frac{b}{2a} + \frac{\sqrt{-\text{discriminant}}}{2a}i$$

$$\text{root}_2 = -\frac{b}{2a} - \frac{\sqrt{-\text{discriminant}}}{2a}i$$

下面给出一个程序，从键盘分别读取二次方程中 a, b, c 的值。然后计算并打印方程的两个根。如果该方程有实根，则将结果打印为实数；如果方程无实根，则将结果打印为复数。代码实现如下：

```
//-----------------------------------------------------------
//   程序 chapter8_9
//
//   该程序计算并打印出一个二次方程的根。程序中使用用户自定义的 complex 类

#include <iostream>
#include <cmath>
#include "ccomplex"
using namespace std;

int main(void)
{
   //   声明并初始化变量
   double a, b, c, term1, disc;
   complex root1, root2;

   //   从键盘读取 a, b, c 的值
   cout << "Enter real values a, b, c:" << endl;
   cin >> a >> b >> c;

   //   计算二次方程的根
   term1 = -b/(2*a);
   disc = b*b - 4*a*c;
   if (disc >= 0)
   {
      root1.real = term1 + sqrt(disc)/(2*a);
      root2.real = term1 - sqrt(disc)/(2*a);
   }
   else
   {
      root1.real = term1;
      root1.imag = sqrt(-disc)/(2*a);
      root2.real = term1;
      root2.imag = -sqrt(-disc)/(2*a);
   }

   //   打印方程的根
   cout << "Roots:" << endl;
   root1.print();
   root2.print();

   //   退出程序
   return 0;
}
//-----------------------------------------------------------
```

400

下面给出程序的两组运行示例：

```
Enter real values a, b, c:
1 -3 -4
Roots:
4
-1

Enter real values for a, b, c:
1 0 4
```

```
Roots:
0+2i
0-2i
```

401

修改

根据本节设计的 complex 类和上面设计的计算二次方程的根的程序来回答下列问题。

1. 用一个具有重根的方程来测试该程序，观察计算结果是否正确。
2. 修改程序，根据原理判定方程根的个数和状态，并根据判定结果打印出相应的提示信息，如"real roots""double roots""complex roots"。
3. 修改程序，在打印方程的根时，保留小数点后两位。

本章小结

C++是C语言的超集。本章介绍了C++程序的基本结构，同时还给出了实现标准I/O和文件I/O的C++语句。面向对象编程的最大特点便是对类和对象的使用。类是用户自定义的数据类型，它包含了数据成员和成员函数，而对象则是类的实例。对象通过调用成员函数来操作类的数据成员。本章给出了一些类定义的说明示例，同时还开发了程序，通过类实现直角坐标系中的点和复数的计算。

关键术语

class（类）
class declaration（类的声明）
class implementation（类的实现）
constructor function（构造函数）
data member（数据成员）
default constructor（默认构造函数）
dot operator（点运算符）
extraction operator（提取运算符）
format flag（格式标志）
inheritance（继承）
insertion operator（插入运算符）
member function（成员函数）
object（对象）
object-oriented programming（面向对象编程）
overload（重载）
polymorphism（多态）
private member（私有成员）
public member（公有成员）
scope resolution operator（范围限定运算符）
stream（流）
white space（空白字符）

C++语句总结

包含了标准输入/输出和文件输入/输出信息的预处理命令：

```
#include <iostream>
#include <fstream>
using namespace std;
```

单行注释：

```
//程序chapter8_1
```

从键盘输入：

402

```
cin >> data;
```

输出至屏幕：

```
cout << data;
```

类的声明：

```
class class_name
{
public:
   function prototypes;
private:
   declarations;
   function prototypes;
};
```

成员函数定义：

```
return type class_name::function_name(parameter list)
{
   declarations;
   statements;
}
```

声明一个对象（即一个类的实例）：

```
xy_coordinate pt1;
```

初始化一个对象：

```
xy_coordinate pt2(3,4);
```

注意事项

1. 一个类的所有数据成员都应该被指定为私有成员。
2. 一个设计良好的类应该拥有多个构造函数。

调试注意事项

1. 应该在每个成员函数实现完成的时候就对其进行检测。如果不加检测就写完整个类的实现，可能会对后面的调试工作造成困难。

习题

简述题

判断题

判断下列陈述的正（T）误（F）。

1. 对象是类的实例。	T	F
2. 对象 cin 通过标准输入设备向程序读入数据。	T	F
3. 函数重载是多态性的一种。	T	F
4. 一个类的所有数据成员必须是同一数据类型。	T	F
5. 成员函数通过点运算符来调用。	T	F
6. 定义成员函数时需要使用点运算符。	T	F
7. 对象是一个类的成员函数。	T	F
8. 成员函数可以访问所有私有数据成员。	T	F
9. 构造函数属于成员函数。	T	F
10. 构造函数不能被重载。	T	F

403

多选题

选出符合下列问题的正确答案。正确选项可能不止一个。

11. 下列哪条 C++ 语句能够正确打印出两个整型变量 a 和 b 的标准输出？（　　）

(a) `cout << a, b;`　　(b) `cout >> a, b;`

(c) `cout << a << b;`　　(d) `cout >> a >> b;`

(e) `cout(a,b);`

12. 下列哪条 C++ 语句可以通过标准输入向变量 a 和 b 赋予正确的值？（　　）

(a) `cin << a, b;`　　(b) `cin >> a, b;`

(c) `cin >> a >> b;`　　(d) `cin << a << b;`

(e) `cin(a,b);`

13. 专门对一个类的数据成员进行初始化的函数叫作（　　）。

(a) 访问函数　　(b) 构造函数

(c) 对象函数　　(d) 类函数

(e) 以上都不对

编程题

14. 定义一个表示日期的类。日期通过三个整型变量来定义：month、day 和 year。类中还应包含能实现以下功能的成员函数：
    - 输入一个日期；
    - 按照 month/day/year（10/1/1999）的格式打印日期；
    - 按照 month day, year（October 1, 1999）的格式打印日期；
    - 初始化一个日期类对象。

15. 定义一个表示时间的类。时间通过三个整型变量来定义：hours、minutes 和 seconds。使用 24 小时制来计时，机器显示。将时间表示为从 00:00:00（即 12 am）到 23:59:59（即 11:59:59 pm）。类中包含能实现以下功能的成员函数：
    - 输入时间；
    - 按 12 小时制打印时间值；
    - 计算两个时间之差；
    - 初始化时间对象。

16. 定义一个有理数类。有理数是由两个整数及一个除号所组成的数，如 1/2、2/3 和 4/5 等。一个有理数类可以通过两个整型变量 numerator（分子）和 denominator（分母）来定义。类中应包含以下运算操作的成员函数：

404 ↑ 405

```
a/b + c/d = (a·d + b·c) / (b·d)        [加法]
a/b - c/d = (a·d - b·c) / (b·d)        [减法]
(a/b) · (c/d) = (a·c)/(b·d)            [乘法]
(a/b) / (c/d) = (a·d) / (c·b)          [除法]
```

ANSI C 语言标准库

本书已经对 ANSI 标准 C 语言库做了详细讨论。接下来，本附录将对标准 C 库中的每个头文件定义进行简要介绍。受篇幅所限，这里不能给出每个函数实现的相关细节，只能帮助读者来确定在开发过程中该使用哪些函数；更详细的信息可以从其他参考资料中获得。在开始介绍这些库之前，我们假定读者已经对指针和字符串等各种变量类型非常熟悉。

<assert.h>

头文件 <assert.h> 给出了 assert 函数定义，该函数可以通过检测程序来提供相应的诊断信息。该诊断信息是系统相关的，被存储在标准错误文件中，用户可以在程序执行完毕后获得。

<ctype.h>

头文件 <ctype.h> 定义了一些字符检测和字符转换的函数。（相关内容见第 2 章。）在讨论函数原型语句之前，首先给出如下定义：

数字	字符 0123456789 中的一个
十六进制数	可以是一个十进制数字，或者是字符 ABCDEFabcdef 中的一个
大写字母	字符 ABCDEFGHIJKLMNOPQRSTUVWXYZ 中的一个
小写字母	字符 abcdefghijklmnopqrstuvwxyz 中的一个
字母字符	一个大写字母或小写字母
字母数字字符	一个数字或字母
标点字符	字符 !"#%&' ();<=>?[\]*+,-./:^ 中的一种
可显示字符	一个字母数字字符或标点字符
可打印字符	一个可显示字符或空格字符
移动控制符	控制符 FF（换页），NL（换行），CR（回车），HT（水平制表符），和 VT（垂直制表符）
空白字符	空格字符或移动控制符中的一种
控制符	BEL（响铃），BS（退格）或者移动控制符中的一种

下面介绍该头文件中的每个函数原型，并简要介绍函数的功能：

```
int isalnum(int c);
```

当且仅当输入的字符为数字或大写、小写字母时，函数返回一个非零值（true）

```
int isalpha(int c);
```

当且仅当输入的字符为大写或小写字母时，函数返回一个非零值（true）

```
int iscntrl(int c);
```

当且仅当输入的字符为控制符时，函数返回一个非零值（true）

```
int isdigit(int c);
```

当且仅当输入的字符为数字时，函数返回一个非零值（true）

```
int isgraph(int c);
```

当且仅当输入的字符是一个可显示字符时，函数返回一个非零值（true）

```
int islower(int c);
```

当且仅当输入的字符是小写字母时，函数返回一个非零值（true）

```
int isprint(int c);
```

当且仅当输入的字符是一个打印字符时，函数返回一个非零值（true）

```
int ispunct(int c);
```

当且仅当输入的字符是一个标点字符时，函数返回一个非零值（true）

```
int isspace(int c);
```

当且仅当输入的字符是一个空白字符时，函数返回一个非零值（true）

```
int isupper(int c);
```

当且仅当输入的字符是一个大写字母时，函数返回一个非零值（true）

```
int isxdigit(int c);
```

当且仅当输入的字符是一个十六进制数时，函数返回一个非零值（true）

```
int tolower(int c);
```

将一个大写字母转换成小写字母

```
int toupper(int c);
```

将一个小写字母转换成大写字母

<errno.h>

头文件 <errno.h> 中定义了宏指令 EDOM、ERANGE，以及一个外部函数 errno。EDOM 和 ERANGE 是两个与系统相关的非零整型常量。函数 errno 的作用是记录系统中发生的错误情况，其使用法也是系统相关的。

<float.h>

头文件 <float.h> 中定义了若干个关于浮点值的限制与特性的宏指令。需要说明的是，浮点数的存储使用的是标准化科学计数法，即使用指数和尾数两部分分别存储，其中尾数的值大于等于 1 且小于 10。这些宏指令及其定义如下所示：

```
int FLT_ROUNDS;
```

指定浮点数的舍入模式

```
int FLT_RADIX;
```

浮点数所表示的基数

```
int FLT_MANT_DIG;
int DBL_MANT_DIG;
int LDBL_MANT_DIG;
```

对 float 型、double 型或 long double 型数值的科学计数法中尾数部分的位数

```
int FLT_DIG;
int DBL_DIG;
int LDBL_DIG;
```

对 float 型、double 型或 long double 型数值，小数点后保留的数字位数

```
int FLT_MIN_EXP;
int DBL_MIN_EXP;
int LDBL_MIN_EXP;
```

对 float 型、double 型或 long double 型数值，基数为 FLT_RADIX 时，指数的最小负整数值 408

```
int FLT_MIN_10_EXP;
int DBL_MIN_10_EXP;
int LDBL_MIN_10_EXP;
```

对 float 型、double 型或 long double 型数值，基数为 10 时指数的最小负整数值

```
int FLT_MAX_EXP;
int DBL_MAX_EXP;
int LDBL_MAX_EXP;
```

对 float 型、double 型或 long double 型，基数为 FLT_RADIX 时指数的最大正整数值

```
int FLT_MAX_10_EXP;
int DBL_MAX_10_EXP;
int LDBL_MAX_10_EXP;
```

对 float 型、double 型或 long double 型，基数为 10 时指数的最大正整数值

```
float FLT_MIN;
double DBL_MIN;
long double LDBL_MIN;
```

对 float 型、double 型或 long double 型，可以表示的最小正值

```
float FLT_MAX;
double DBL_MAX;
long double LDBL_MAX;
```

对 float 型、double 型或 long double 型，可以表示的最大正值

```
float FLT_EPSILON;
double DBL_EPSILON;
long double LDBL_EPSILON;
```

对 float 型、double 型或 long double 型在 1 和大于 1 的最小值之间的差值

<limits.h>

头文件 <limits.h> 提供了若干个关于整型值的限制与特性的宏指令。这些宏指令及其定义如下所示：

```
int CHAR_BIT;
```

定义最小的数据类型（除位特数据除外）包含的比特数

```
int CHAR_MIN;
int CHAR_MAX;
```

定义 char 类型的最大值和最小值 409

```
int INT_MIN;
int INT_MAX;
```

定义 int 类型的最大值和最小值

```
int LONG_MIN;
int LONG_MAX;
```

定义 long int 类型的最大值和最小值

```
int MB_LEN_MAX;
```

定义多字节字符的最大字节数

```
int SCHAR_MIN;
int SCHAR_MAX;
```

定义 signed char 类型的最大值和最小值

```
int SHRT_MIN;
int SHRT_MAX;
```

定义 short int 类型的最大值和最小值

```
int UCHAR_MAX;
```

定义 unsigned char 类型的最大值

```
int UINT_MAX;
```

定义 unsigned int 类型的最大值

```
int ULONG_MAX;
```

定义 unsigned long int 类型的最大值

```
int USHRT_MAX;
```

定义 unsigned short int 类型的最大值

<locale.h>

头文件 <locale.h> 定义了两个函数、一种数据类型以及若干关于数值格式化的宏指令。它可以通过数字格式规范化来解决国际通用标准化的问题；此外还包括有关货币符号和日期格式化等工作。

<math.h>

头文件 <math.h> 中定义了多种数学相关的函数。这些函数在工程计算问题中经常用到。这些函数已经在第 2 章做了详细介绍，具体函数功能如下所示：

```
double acos(double x);
```

计算 x 的反余弦值。其中，x 的定义域为 [-1, 1]；函数返回值的值域是 [0, π]（以弧度值表示）

```
double asin(double x);
```

计算 x 的反正弦值。其中，x 的定义域为 [-1, 1]；函数返回值的值域是 [-π/2, π/2]（以弧度值表示）

```
double atan(double x);
```

计算 x 的反正切值。函数返回值的值域是 [-π/2, π/2]（以弧度值表示）

```
double atan2(double y, double x);
```

计算 y/x 的反正切值。函数返回值的值域是 [-π, π]（以弧度值表示）

```
int ceil(double x);
```

返回一个不小于 x 的最小整数

```
double cos(double x);
```

计算 x 的余弦值，其中 x 以弧度值表示

```
double cosh(double x);
```

计算 x 的双曲余弦值，等价于 $(e^x+e^{-x})/2$

```
double exp(double x);
```

计算 e^x 的值，其中 e 是自然对数的底数，约等于 2.718 282

```
double fabs(double x);
```

计算 x 的绝对值

```
int floor(double x);
```

返回一个不大于 x 的最大整数

```
double log(double x);
```

计算 ln x，也就是 x 的自然对数值（以 e 为底）；如果 x ≤ 0 则会报错 410

```
double log10(double x);
```

计算 $\log_{10}x$，也就是 x 的常用对数（以 10 为底）；如果 x ≤ 0 则会报错

```
double pow(double x, double y);
```

计算 x 的 y 次幂（次方），即 x^y；当 x=0 且 y ≤ 0，或者 x<0 且 y 不是整数时会报错

```
double sin(double x);
```

计算 x 的正弦值，其中 x 为弧度值

```
double sinh(double x);
```

计算 x 的双曲正弦值，其等价于 $(e^x-e^{-x})/2$

```
double sqrt(double x);
```

计算 x 的平方根，其中 x ≥ 0

```
double tan(double x);
```

计算 x 的正切值，其中 x 为弧度值

```
double tanh(double x);
```

计算 x 的双曲正切值，其等价于 (sinh x)/(cosh x)

<setjmp.h>

头文件 <setjmp.h> 中定义了一个宏指令、一个函数和一个变量类型，该变量类型和相应的函数能够绕过正常的函数调用和返回规则。一般来讲，并不推荐使用这些操作，故在此不做详细讨论。

<signal.h>

头文件 <signal.h> 中定义了一个数据类型、两个函数和几个宏指令来处理程序执行期间报告的各种信号。有些信号表示系统中可能发生了严重错误。由于信号处理是不可移植

的，故在此不对该头文件进行详细讨论。通常来讲，系统提供的默认信号处理机制已足够应付正常的操作。

<stdarg.h>

头文件<stdarg.h>定义了一个变量类型和三个宏，这些宏指令可以在参数个数不知的情况下获取函数中的参数。尽管该头文件实现的功能十分强大，但在工程应用中很少使用。

<stddef.h>

头文件<stddef.h>定义了各种变量类型和宏指令，并且它们之间没有太多关联。由于这些变量类型和宏在工程应用中并不常用，故在此不做详细介绍。

<stdio.h>

头文件<stdio.h>定义了多种数据类型、宏指令和函数，来执行输入和输出。在这些数据类型中，FILE类型在工程应用中是最常用的，因为它可以用于操作各种数据文件。在第2章和第3章中已经讨论过了大多数的输入/输出函数；除此之外还有以下函数：

```
void clearerr(FILE *stream);
```

清除指定流的文件结束符和错误标识符

```
int fclose(FILE *stream);
```

411

关闭stream所指向的文件，并且清空缓冲区

```
int feof(FILE *stream);
```

检查stream指向的流文件是否已经到达文件尾部

```
int ferror(FILE *stream);
```

检测stream指向的流文件的错误标识符

```
int fflush(FILE *stream);
```

刷新给定流stream的输出缓冲区

```
int fgetc(FILE *stream);
```

从指定的流stream获取下一个字符，并把文件读写指针位置标识符向前移动

```
int fgetpos(FILE *stream, fpos_t *pos);
```

获取流stream的当前文件读写指针位置，并把它写入pos

```
char *fgets(char *s, int n, FILE *stream);
```

从指定的流读取一行，并把它存储在s所指向的字符串内。当读取到n-1个字符时，或读取到换行符时，或者到达文件末尾时，会停止

```
FILE *fopen(const char *filename, const char *mode);
```

使用指定的模式mode打开filename所指向的文件

```
int fprintf(FILE *stream, const char *format, ...);
```

发送格式化输出到流stream中，函数返回输出的字符个数

```
int fputc(int c, FILE *stream);
```

把参数 c 指定的字符写入指定的流 stream 中，并将读写位置标识符向前移动

```
int fputs(const char *s, FILE *stream);
```

把参数 s 指定的字符串写入指定的流 stream 中，但不包括空字符

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

从给定流 stream 中读取数据到 ptr 所指向的数组中，写入的数据元素数目最多为 nmemb，其中每个元素大小为 size 字节

```
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

把一个新文件名 filename 与给定的打开的流 stream 相关联，同时将流中的旧文件关闭

```
int fscanf(FILE *stream, const char *format, ...);
```

从流 stream 中读取格式化输入；函数返回输入值的个数

```
int fseek(FILE *stream, long int offset, int whence);
```

设置给定流 stream 的文件读写位置为指定位置，由函数参数 pos

```
int fsetpos(FILE *stream, const fpos_t *pos);
long int ftell(FILE *stream);
```

返回给定流 stream 的当前文件位置

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
        FILE *stream);
```

把 ptr 所指向数组中的数据写入到给定流 stream 中；其中写入元素数量最多为 nmemb，元素大小为 size 个字节

```
int getc(FILE *stream);
```

从指定的流 stream 中获取下一个字符，并将文件读写位置标识符向前移动

```
int getchar(void);
```

从标准输入流 stdin 获取一个字符 412

```
char *gets(char *s);
```

从标准输入流读入字符到 s 指向的数组中，当读取到换行符或到文件末尾时停止；在数组中换行符被空字符替代

```
void perror(const char *s);
```

将参数表达式 errno 中的错误代号映射成一条错误信息，并存储在 s 所指向的字符串中

```
int printf(const char *format, ...);
```

发送格式化输出到标准输出流 stdout，使用 format 指定的输入格式；函数返回输出字符的个数

```
int putc(int c, FILE *stream);
```

把参数 c 指定的字符写入 stream 指向的输出流文件中

```
int putchar(int c);
```

把参数c指定的字符写入标准输出stout中

```
int puts(const char *s);
```

将参数s指向的字符串写入标准输出流中，并且字符串中的空字符用换行符来替代

```
int remove(const char *filename);
```

删除给定的文件名filename

```
int rename(const char *old, const char *new);
```

将原来由old指向的文件改为由new来指向，其中old和new中分别包含了该文件的旧文件名和新文件名

```
void rewind(FILE *stream);
```

设置文件读写位置到给定流stream的开头

```
int scanf(const char *format, ...);
```

从标准输入流读取格式化输入，使用format指定的输入格式；函数返回为输入值的个数

```
void setbuf(FILE *stream, char *buf);
```

指定流stream的缓冲类型

```
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
```

指定流stream的缓冲类型

```
int sprintf(char *s, const char *format, ...);
```

发送格式化输出到参数s指向的数组，使用format指定的输入格式；在输出字符的末尾附加一个空字符；函数返回输出字符的个数，其中不包括空字符

```
int sscanf(const char *s, const char *format, ...);
```

从参数s指向的字符串读取格式化输入，使用format指定的输入格式；函数返回输入值的个数

```
FILE *tmpfile(void);
```

创建一个暂时二进制文件，并且当文件关闭时会自动删除

```
char *tmpnam(char *s);
```

产生一个合法的文件名，且不能与任何已存在的文件名重复

```
int ungetc(int c, FILE *stream);
```

将参数c指定的字符压入给定的流stream中，以便它是下一个被读取到的字符

```
int vfprintf(FILE *stream, const char *format, va_list arg);
```

发送格式化输出到流文件stream，使用format指定的输出格式和可变参数列表中的参数值；函数返回为输出字符的个数

413

```
int vprintf(const char *format, va_list arg);
```

发送格式化输出到标准输出流stream，使用format指定的输出格式和可变参数列表中包含的参数值；函数返回输出字符的个数

```
int vsprintf(char *s, const char *format, va_list arg);
```

发送格式化输出到参数 s 指向的数组中。使用 format 指定的输出格式和可变参数列表中包含的参数值；对在写入字符的末尾附加一个空字符；函数返回输出字符的个数，但不包括空字符

<stdlib.h>

头文件 <stdlib.h> 定义了 4 个数据类型、一些宏指令和通用工具函数。类型 div_t 和 ldiv_t 是两个结构体，用来存储商和余数。下面首先介绍这些宏指令：

NULL

二进制零的指针值

EXIT_FAILURE
EXIT_SUCCESS

分别用来表示 exit 函数执行失败和成功时要返回的状态值

RAND_MAX

表示 RAND 函数返回的最大值

MB_CUR_MAX

表示在多字节字符集中的最大字符数

下面展示一些在工程应用中常用的函数，包括函数原型语句和函数定义（其中的很多函数已经在第 3 章和第 6 章讨论过）：

```
void abort(void);
```

使一个异常程序终止

```
int abs(int k);
long int labs(long int k);
```

计算整数 k 的绝对值

```
int atexit(void (*func)(void));
```

当函数正常终止时，调用指定的函数 func

```
double atof(const char *s);
int atoi(const char *s);
long int atol(const char *s);
double strtod(const char *s, char **endptr);
long int strtol(const char *s, char **endptr, int base);
unsigned long int strtoul(const char *s, char **endptr, int base);
```

把参数 s 所指向的字符串转换为一个浮点数（float 型）/ 整数（int 型）/ 长整数（long int 型）/ 双精度浮点数（double 型）/ 无符号长整数（unsigned long int 型）

```
void *bsearch(const void *key, const void *base, size_t n, size_t
              size, int(*compar)(const void *,const void *));
```

在由 n 个对象组成的数组中进行二分查找，查找 key 所指向的值 414

```
void *calloc(size_t n, size_t size);
```

为一个具有 n 个对象的数组分配空间，每个空间大小为 size

```
div_t div(int numer, int denom);
ldiv_t ldiv(long int numer, long int denom);
```

计算 numer 除以 denom 得到的商和余数

```
void exit(int status);
```

使程序正常终止

```
void free(void *ptr);
```

释放由 ptr 所指向的空间

```
void *malloc(size_t size);
```

为一个对象分配大小为 size 的空间

```
void qsort(void *base, size_t nmemb, size_t size, int
           (*compar)(const void*, const void *));
```

使用快速排序将一个具有 n 个对象的数组排列成升序

```
int rand(void);
```

返回一个从 0 到 RAND_MAX 之间的伪随机数

```
void *realloc(void *ptr, size_t size);
```

改变由 ptr 所指向的对象的大小

```
void srand(unsigned int seed);
```

设置随机数种子，对函数 RAND 使用的随机数发生器进行初始化

<string.h>

头文件 <string.h> 定义了一个无符号整型数据类型 size_t；一个宏 NULL，它的值为二进制零。另外，头文件中还定义了一些函数来实现各种字符串操作（这些函数在第 6 章已经讨论过）：

```
void *memchr(const void *s, int c, size_t n);
```

在参数 s 所指向的字符串的前 n 个字符中搜索字符 c 第一次出现的位置

```
int memcmp(const void *s, const void *t, size_t n);
```

对两个字符串 s 和 t 的前 n 个字符进行比较，当 s>t 时返回一个正整数；当 s=t 时返回零；当 s<t 时返回一个负整数

```
void *memcpy(void *s, const void *t, size_t n);
```

将参数 t 所指向的字符串的前 n 个字符复制到 s 所指向的字符串中

```
void *memmove(void *s, const void *t, size_t n);
```

同样是将 t 所指向字符串的前 n 个字符复制到 s 所指向的字符串中，不同之处是该函数使用一个临时存储空间中转，即使 t 和 s 的空间出现了重叠也不会出错

```
void *memset(void *s, int c, size_t n);
```

复制字符 c 到参数 s 所指向字符串的前 n 个字符

```
char *strcat(char *s, const char *t);
```

把参数 t 所指向的字符串连接到 s 所指向字符串的末尾；函数返回字符串 s 的指针

```
char *strchr(const char *s, int c);
```

返回参数s所指向的字符串中第一次出现字符c的位置指针

```
int strcmp(const char *s, const char *t);
```

逐个元素地比较字符串s和字符串t，当s>t时，返回一个正整数；当s=t时，返回零；当s<t时，返回一个负整数

415

```
int strcoll(const char *s, const char *t);
```

默认情况下该函数同strcmp函数功能相同。对于设置了LC_COLLATE语言环境的情况下，则根据设定的适当排序方式进行字符串比较

```
char *strcpy(char *s, const char *t);
```

将参数t所指向的字符串复制到s所指向的字符串中；函数返回指向字符串s的指针

```
size_t strcspn(const char *s, const char *t);
```

检查s指向的字符串开头连续有几个字符不包含字符串t中的字符；函数返回未包含t中字符的最大字符数

```
size_t strlen(const char *s);
```

返回参数s所指向的字符串的长度

```
char *strncat(char *s, const char *t, size_t n);
```

把字符串t中的最多n个字符追加到字符串s的结尾；函数返回指向字符串s的指针

```
int strncmp(const char *s, const char *t, size_t n);
```

将字符串s和字符串t逐个字符进行比较（最多比较前n个字符）。当s>t时，函数返回一个正整数；当s=t时，函数返回零；当s<t时，函数返回一个负整数

```
char *strncpy(char *s, const char *t, size_t n);
```

将字符串t中的最多n个字符复制到字符串s中；如果t的字符数少于s，则s中的剩余位置用空字符填充；函数返回指向字符串s的指针

```
char *strpbrk(const char *s, const char *t);
```

返回字符串t中的任意字符在字符串s中第一次出现的位置指针；也就是说，依次检查s中的字符，当被检查字符在字符串t中也包含时，则停止检查，并返回该字符的位置

```
char *strrchr(const char *s, int c);
```

在参数s所指向的字符串中搜索最后一次出现字符c的位置；函数返回该位置的指针

```
size_t strspn(const char *s, const char *t);
```

返回字符串s中第一个不在字符串t中出现的字符下标

```
char *strstr(const char *s, const char *t);
```

在字符串t中搜索第一次出现字符串s的位置（不包含空结束字符）；函数返回该位置的指针

<time.h>

头文件<time.h>定义了两个宏、四种数据类型以及各种操作日期和时间的函数。数据类型clock_t和time_t是表示时间的数据类型；结构体tm包含了一个日历时间，被表示为秒（tm_sec）、分（tm_min）、时（tm_hour）、日（tm_mday）、月（从一月开始，tm_mon）、

年（从1900年开始，tm_year），星期（从周日开始，tm_wday），日期（从一月一日开始，tm_yday），和夏令时（tm_isdst）；结构体中这些值的顺序是依赖系统的。下面介绍相关的宏指令：

CLOCKS_PER_SEC

每秒钟的处理器时钟个数

NULL

416 表示一个空指针常量的值，为二进制零

接下来介绍头文件中的函数原型及功能描述：

```
char *asctime(const struct tm *timeptr);
```

该函数将结构体timeptr所表示的日期和时间转换成一个字符串并返回

```
clock_t clock(void);
```

返回当前处理器的时间

```
char *ctime(const time_t *timer);
```

该函数将结构体timer所表示的时间转换成一个字符串并返回

```
double difftime(time_t time1, time_t time0);
```

函数返回time1和time0间相差的秒数

```
struct tm *gmtime(const time_t *timer);
```

该函数将结构体timer所表示的时间转换为格林威治标准时间并返回

```
struct tm *localtime(const time_t *timer);
```

该函数将结构体timer所表示的时间转换为本地时间

```
time_t mktime(struct tm *timeptr);
```

将结构体timeptr的时间值转换成一个系统时间

```
time_t time(time_t *timer)
```

返回当前的日历时间

```
size_t strftime(char *s, size_t maxsize, const char *format,
                const struct tm *timeptr);
```

417 根据format中定义的格式化规则，格式化结构体timeptr中的时间，并把它存储在字符串s中

ASCII 字符编码表

下面的表格列出了 128 个 ASCII 字符，以及它们的整数值和二进制数值。其中，整数 1 – 31 对应的字符对计算机系统具有特殊含义。例如，整数 7 表示的字符 BEL 会产生键盘铃声。

表中的字符是按照对应的整数序列排序的，其中有一些有趣的特征。例如，数字的顺序要小于大写字母，而大写字母的顺序又小于小写字母。同时，特殊字符并不是都排在一起——有些字符排在数字之前，有些字符排在大写字母和小写字母之间。下面给出 ASCII 字符码表。

字符	整数值	二进制数值
NUL（二进制零）	0	0000000
SOH（标题始）	1	0000001
STX（正文始）	2	0000010
ETX（正文尾）	3	0000011
EOT（传送止）	4	0000100
ENQ（查询）	5	0000101
ACK（信号确认）	6	0000110
BEL（响铃）	7	0000111
BS（退格）	8	0001000
HT（水平制表）	9	0001001
LF（换行）	10	0001010
VT（垂直制表）	11	0001011
FF（换页）	12	0001100
CR（回车）	13	0001101
SO（移出）	14	0001110
SI（移入）	15	0001111
DLE（数据链路转义）	16	0010000
DC1（设备控制 1）	17	0010001
DC2（设备控制 2）	18	0010010
DC3（设备控制 3）	19	0010011
DC4（设备控制 4– 停机）	20	0010100
NAK（信息否认）	21	0010101
SYN（同步）	22	0010110
ETB（组传送止）	23	0010111
CAN（取消）	24	0011000
EM（介质结束）	25	0011001
SUB（替换）	26	0011010

（续）

字符	十进制数值	二进制数值
ESC（转义）	27	0011011
FS（文件分隔符）	28	0011100
GS（分组符）	29	0011101
RS（记录分隔符）	30	0011110
US（单元分隔符）	31	0011111
SP（空格）	32	0100000
!	33	0100001
"	34	0100010
#	35	0100011
$	36	0100100
%	37	0100101
&	38	0100110
'（右单引号）	39	0100111
(	40	0101000
)	41	0101001
*	42	0101010
+	43	0101011
,（逗号）	44	0101100
-（连字线）	45	0101101
.（圆点）	46	0101110
/	47	0101111
0	48	0110000
1	49	0110001
2	50	0110010
3	51	0110011
4	52	0110100
5	53	0110101
6	54	0110110
7	55	0110111
8	56	0111000
9	57	0111001
:	58	0111010
;	59	0111011
<	60	0111100
=	61	0111101
>	62	0111110
?	63	0111111
@	64	1000000
A	65	1000001
B	66	1000010
C	67	1000011

418

（续）

字符	整数值	二进制数值
D	68	1000100
E	69	1000101
F	70	1000110
G	71	1000111
H	72	1001000
I	73	1001001
J	74	1001010
K	75	1001011
L	76	1001100
M	77	1001101
N	78	1001110
O	79	1001111
P	80	1010000
Q	81	1010001
R	82	1010010
S	83	1010011
T	84	1010100
U	85	1010101
V	86	1010110
W	87	1010111
X	88	1011000
Y	89	1011001
Z	90	1011010
[	91	1011011
\	92	1011100
]	93	1011101
^（插入符，| 寿山符）	94	1011110
_（下划线），	95	1011111
`（左单引号）	96	1100000
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
e	101	1100101
f	102	1100110
g	103	1100111
h	104	1101000
i	105	1101001
j	106	1101010
k	107	1101011
l	108	1101100

419

（续）

字符	数值	二进制数值
m	109	1101101
n	110	1101110
o	111	1101111
p	112	1110000
q	113	1110001
r	114	1110010
s	115	1110011
t	116	1110100
u	117	1110101
v	118	1110110
w	119	1110111
x	120	1111000
y	121	1111001
z	122	1111010
{	123	1111011
|	124	1111100
}	125	1111101
~	126	1111110
DEL（删除/擦除）	127	1111111

[420]

使用 MATLAB 绘制文本文件中的数据点

为了更好地理解工程问题并提出解决方案，需要将涉及的数字信息进行可视化。因此，能够从数据文件中轻松得到数据在直角坐标系中的简单分布是解决工程问题的一项必备能力。

在该附录中，首先给出一个可以生成数据文件的简单 C 程序，然后会向大家展示如何使用 MATLAB 来得到数据分布图。之所以选择 MATLAB 来生成附录 C 和书中其他章节中的数据图形，是因为它是非常强大的交互计算软件环境，非常擅长于处理数值计算、数据分析和绘制图形类型的问题。

在后面的例子中，会通过 C 程序生成文本文件，然后使用 MATLAB 来绘制信息图像。当然，文本文件也可以由文字处理软件生成，然后采用相同的步骤用 MATLAB 绘制图像。如果数据文件是由文字处理软件生成，那么要注意选择将文件保存为文本文件格式，而不是保存为文字处理软件的文件格式。

下面的程序会生成一个包含 100 行信息的数据文件。每一行都包含相应的时间值和阻尼正弦函数值

$$f(t) = e^{-t}\sin(2\pi t)$$

其中 $t = 0.0, 0.1, 0.2, \cdots, 9.9$ 秒。程序中使用的打开数据文件、向数据文件写入信息和关闭数据文件等操作的相关语句已经在第 3 章里介绍过。

使用 C 程序生成数据文件

```
/*                                                          */
/*   程序 chapterc_1                                        */
/*                                                          */
/*   该程序根据阻尼正弦函数生成数据文件                      */

#include <stdio.h>
#include <math.h>
#define PI 3.141593
#define FILENAME "dsine.txt"

int main(void)
{
   /* 声明变量 */
   int k;
   double t, f;
   FILE *data_out;

   /* 生成数据文件 */
   data_out = fopen(FILENAME,"w");
   for (k=1; k<=100; k++)
   {
      t = 0.1*(k-1);
      f = exp(-t)*sin(2*pi*t);
      fprintf(data_out,"%.1f %.3f \n",t,f);
   }
```

```
    /*     关闭数据文件并退出程序  */
    fclose (data_out);
    return 0;
}
/*-----------------------------------------------------------------*/
```

由 C 程序生成的数据文件

由样例程序生成的数据文件中，每行包含两个数值。该文件的前几行和后几行数据如下所示：

```
0.0  0.000
0.1  0.532
0.2  0.779
...
9.9  0.000
```

使用 MATLAB 生成数据分布图像

用 MATLAB 绘制数据信息的分布图像只需要两条语句。在执行下面的两条语句之前，要先将文件保存在 MATLAB 工作目录（文件夹）中。第一条语句将文件加载到 MATLAB 工作区，第二条语句生成数据 xy 坐标在直角坐标系中绘制相应的曲线：

```
>>load dsine.txt
>>plot(dsine(:,1),dsine(:,2))
```

生成的图像如图 C-1 所示。

除了显示图像，还要添加标注信息。通过下面的语句，可以显示出图像标题、标注坐标轴并且增加背景网格：

```
>>load dsine.txt
>>plot(dsine(:,1),dsine(:,2)),
>>title('Damped Sine Function'),
>>xlabel('Time, s'),ylabel('f(t)'),grid
```

图 C-2 显示了带标注的分布图像。上述语句同样是假设数据文件存储在 MATLAB 工作目录中。

[422]

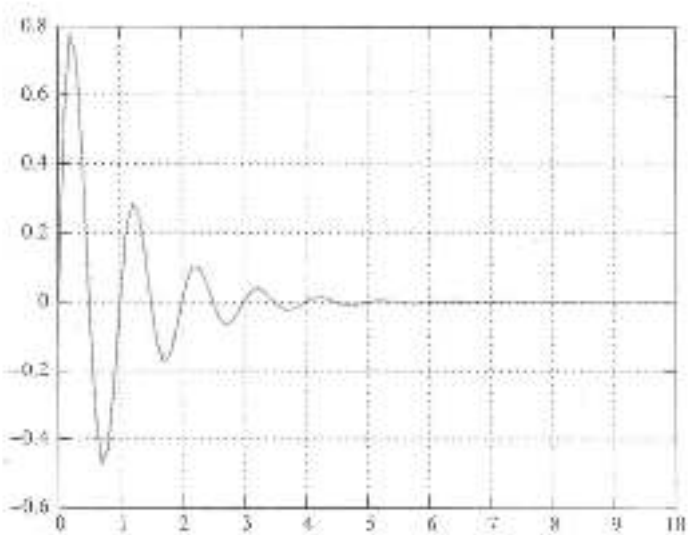


图 C-1　阻尼正弦函数图

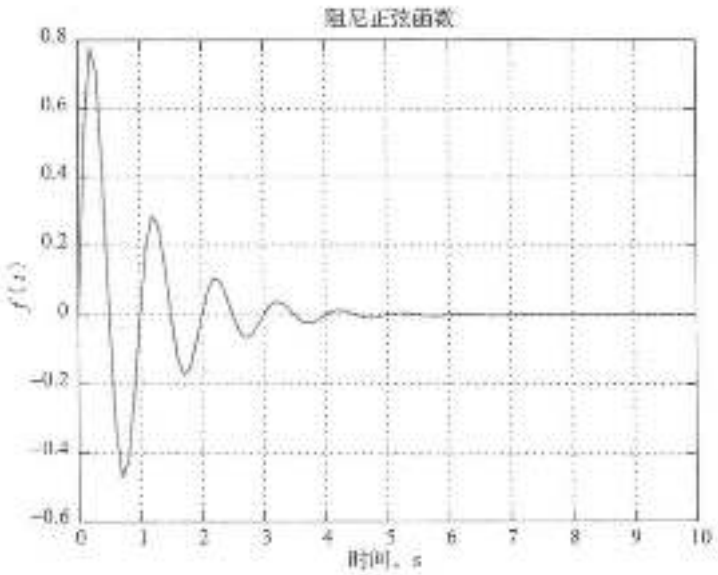


图 C-2 带加标注的阻尼正弦函数图

423

“练习”的完整答案

2.2 节

1. 合法
2. 合法
3. 合法
4. 合法
5. 合法
6. 非法字符（-），合法形式为 tax_rate
7. 合法
8. 非法字符（^），合法形式为 sec_sqrt
9. 合法
10. 非法，关键字，合法形式为 break_1
11. 非法字符（#），合法形式为 num_123
12. 非法字符（&），合法形式为 x_and_y
13. 合法
14. 非法，关键字，合法形式为 void_term
15. 非法字符（(，)），合法形式为 fx
16. 合法
17. 合法
18. 非法字符（.），合法形式为 w1_1
19. 合法
20. 合法
21. 非法字符（/），合法形式为 m_per_s

2.2 节

1. $3.500\,4 \times 10^{1}$
2. 4.2×10^{-4}
3. -5.0×10^{4}
4. $3.157\,23 \times 10^{0}$
5. -9.99×10^{-2}
6. $1.000\,000\,28 \times 10^{7}$
7. 0.000 010 3
8. −105 000
9. −3 552 000
10. 0.000 667
11. 0.090 2

12. -0.022

424

2.2 节

```
1. #define LIGHT_SPEED 2.99792e08
2. #define CHARGE_E 1.602177e-19
3. #define N_A 6.022e23
4. #define G_MSS 9.8
5. #define G_FISS 32
6. #define MASS 5.98e24
7. #define MOON_RADIUS 1.74e06
8. #define UNIT_LENGTH 'm'
9. #define UNIT_TIME 's'
```

2.3 节

1. 6
2. 4.5
3. 3
4. 3.0

2.3 节

```
1. distance = x0 + v0*t + 0.5*a*t*t;
2. tension = (2*m1*m2*g)/(m1 + m2);
3. P2 = P1 + rho*v2*v2*(A2*A2 - A1*A1)/(2*A1*A1);
```

4. centripetal $= \dfrac{4\pi^2 r}{T^2}$

5. potential energy $= \dfrac{GM_E m}{r}$

6. change $= GM_E m\left(\dfrac{1}{R_E} - \dfrac{1}{R_E + h}\right)$

2.3 节

1. x [3] y [4] z [8]
2. x [3] y [4] z [12]
3. x [6] y [4]
4. x [2] y [0]

2.4 节

```
1. Sum =    65; Average =    12.4
2. Sum =    65
   Average = 12.3680
3. Sum and Average

   65 12.4
```

425

```
4. Character is b; Sum is A
5. Character is 98; Sum is 65
6.   12.37 is the average;
        65 is the sum
7.   12.37 is the average;   65 is the sum
```

2.6 节

1. 75.86°　89.35°　111.25°　109.92°

2. 0.67　1.60　1.71　1.87

3. 如图 2-5 所示，对应 110° 有 5 个时间值。这些值计算如下：

   1.98　2.84　3.39　4.42　4.67

2.8 节

1. −3	2. −2	3. 0.125	4. 3.16
5. 25	6. 11	7. −1	8. 32

2.8 节

```
1. velocity = sqrt(pow(v0,2) + 2*a*(x - x0));
2. length = k*sqrt(1 - pow(v/c,2));
3. center = 38.1972*(r*r*r - s*s*s)*sin(a)/((r*r - s*s)*a);
```

4. $$\text{frequency} = \frac{1}{\sqrt{\frac{2\pi c}{L}}}$$

5. $$\text{range} = \frac{v_0^2}{g}\sin 2\theta$$

6. $$v = \sqrt{\frac{2gh}{1 + \frac{I}{mr^2}}}$$

2.8 节

```
1. cothx = cosh(x)/sinh(x);
2. secx = 1/cos(x);
3. cscx = 1/sin(x);
4. acothx = 0.5*log((x + 1)/(x - 1));
5. acoshx = log(x + sqrt(x*x - 1));
6. acscx = asin(1/x);
```

426

2.9 节

```
1. x
2. A
3. a
4. ac
   c
```

3.2 节

1. 正确	2. 正确	3. 正确	4. 正确

5. 正确　　6. 正确　　7. 错误　　8. 错误

3.3 节

```
1. if (time > 15)
      time += 1;
2. if (sqrt(poly) < 0.5)
      printf("poly = %f \n",poly);
3. if (abs(volt_1-volt_2) > 10)
      printf("volt_1: %f, volt_2: %f \n",volt_1,volt_2);
4. if (den < 0.05)
      result = 0;
   else
      result = num/den;
5. if (log(x) >= 3)
   {
      time = 0;
      count--;
   }
6. if (dist<50.0 && time>10)
      time += 2;
   else
      time += 2.5;
7. if (dist >= 100)
      time += 2;
   else
      if (dist > 50)
         time += 1;
      else
         time += 0.5;
```

427

3.3 节

```
1. switch (rank)
   {
      case 1: case 2:
         printf("Lower division \n");
         break;
      case 3: case 4:
         printf("Upper division \n");
         break;
      case 5:
         printf("Graduate student \n");
         break;
      default:
         printf("Invalid rank \n");
         break;
   }
```

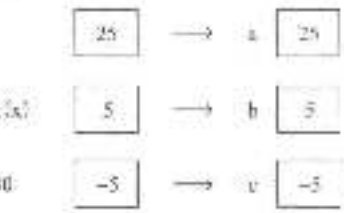
3.5 节

1. 18　　2. 18　　3. 17

4. 9　　5. 无限循环　　6. 15

4.2 节

1. 实参



2. 2

4.2 节

1. 外部标识符：无

2. 局部变量及其范围：

main 函数：

seed, n, k, component_reliability, a_series, a_parallel, series_success, parallel_success, num1, num2, num3, rand_float

rand_float 原型语句：

a, b

rand_float 函数：

a, b

3. 外部标识符：无

4. 局部变量及其范围：

main 函数：

n, k, a0, a1, a2, a3, a, b, step, left, right

check_roots 原型语句：

left, right, a0, a1, a2, a3

check_roots 函数：

left, right, a0, a1, a2, a3, f_left, f_right

poly 原型语句：

x, a0, a1, a2, a3

poly 函数：

x, a0, a1, a2, a3

4.9 节

1.
```
#define area_sq(side) ((side)*(side))
printf("area: %f \n",area_sq(side1));
```

2.
```
#define area_rect(side1,side2) ((side1)*(side2))
sum = area_rect(sidea,sideb) + area_rect(sidec,sided);
```

```
3. #define area_par(base,height) ((base)*(height))
   area1 = area_par(b,h1);

4. #define area_trap(base,height1,height2)
           (0.5*(base)*((height1)+(height2)))
   area += area_trap(base,left,right);

5. #define vol_sph(radius) (4.0/3.0*3.141593*pow((radius),3))
   printf("volume: %f \n",vol_sph(5.5));

6. #define vol_pyr(area,height) (1.0/3.0*(area)*(height))
   vol1 = vol_pyr(0.5*baseb*baseht,pyrht);

7. #define vol_cone(radius,height)
           (1.0/3.0*3.141593*pow((radius),3)*(height))
   vol2 = vol_cone(diameter/2,ht);

8. #define vol_cube(side) ((side)*(side)*(side))
   vol3 = vol_cube(sqrt(base_area));

9. #define vol_par(length,width,height)
           ((length)*(width)*(height))
   vol4 = vol_par(side1,side1,side1);
```

429

5.1 节

1.

-5	4	3	0	0	0	0	0	0	0

2.

'a'	'b'	'c'

3.

?	-5.5	5.5	5.5

4.

-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4

5.1 节

1. 3 8　　2. 8 30
   15 21
   30 41

5.1 节

1. 9.8　2. 9.8　3. 3.2　4. 1.5

5.4 节

1. 9　2. 6　3. 5.36　4. 2.32
5. 2.5　6. 5.75

5.8 节

1.

1
4
6

2.

5	2
−2	3
0	0
0	0
0	0
0	0

430

3.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

4.

1	0	0
0	1	0
0	0	1

5.

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

6.

1	−1	1	−1	1
1	−1	1	−1	1
1	−1	1	−1	1
1	−1	1	−1	1
1	−1	1	−1	1

5.8 节

1. 9
2. 4
3. −6
4. 3

5.8 节

1. 13
2. 2
3. 18
4. 22

5.10 节

1. 5

2. −8

3. $\begin{bmatrix} 5 & -1 & 3 \\ 3 & -3 & 2 \end{bmatrix}$ 431

4. $\begin{pmatrix} -4 & 12 \end{pmatrix}$

5. $\begin{bmatrix} -2 & -2 & 4 \\ 7 & -9 & 10 \end{bmatrix}$

6. $\begin{bmatrix} 24 \\ 20 \\ 18 \end{bmatrix}$

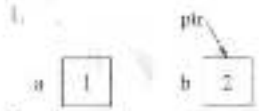
5.11 节

1. x = 2, y = 1

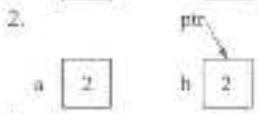
2. x = 3, y = −1, z = 2

6.1 节

1.



2.

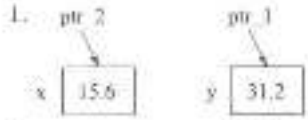


3.

4.

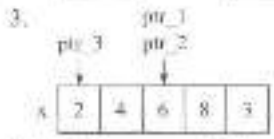
6.2 节

1.



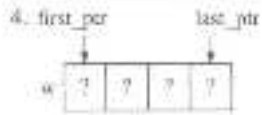
2.

3.



432

4.



6.2 节

		偏移量
q[0]	2	0

g[1]	4	1
g[2]	5	2
g[3]	8	3
g[4]	10	4
g[5]	32	5
g[6]	78	6

1. 2
2. 4
3. 3
4. 32
5. 2
6. 8
7. 4
8. 32

6.2 节

1.

		偏移量
d[0][0]	1	0
d[0][1]	6	1
d[1][0]	0	2
d[1][1]	0	3
d[2][0]	0	4
d[2][1]	0	5
d[3][0]	0	6
d[3][1]	0	7

2.

		偏移量
g[0][0]	5	0
g[0][1]	2	1
g[0][2]	-2	2
g[0][3]	3	3
g[1][0]	1	4
g[1][1]	2	5
g[1][2]	3	6
g[1][3]	4	7
g[2][0]	0	8
g[2][1]	0	9
g[2][2]	0	10
g[2][3]	0	11

433

3.

		偏移量
h[0][0]	0	0
h[0][1]	0	1
h[0][2]	0	2
h[1][0]	0	3
h[1][1]	0	4
h[1][2]	0	5
h[2][0]	0	6
h[2][1]	0	7
h[2][2]	0	8

6.2 节

		偏移量
x[0][0]	1	0

x[0][1]	8	1
x[0][2]	7	2
x[0][3]	6	3
x[1][0]	2	4
x[1][1]	4	5
x[1][2]	-1	6
x[1][3]	0	7

1. 1　　3. 3

2. 7　　4. 14

6.2 节

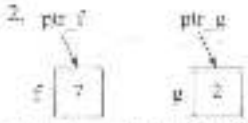
```
1. int a[4][6], sum=0, *ptr=&a[1][0];
   ...
   for (k=0; k<=5; k++)
      sum += *(ptr+k);
2. int a[4][6], sum=0, *ptr=&a[0][2];
   ...
   for (k=0; k<=3; k++)
      sum += *(ptr-k*6);
3. int a[4][6], *ptr=&a[0][0];
   ...
   max = a[0][0];
   for (k=0; k<=17; k++)
      if (max < *(ptr-k))
         max = *(ptr-k);
4. int a[4][6], *ptr=&a[0][0];
   ...
   min = a[0][2];
   for (i=0; i<=3; i++)
      for(j=2; j<=5; j++)
         if (min > *(ptr+i*6+j))
            min = *(ptr+i*6+j);
```

434

6.4 节

1. 非法，参数不是指向整数型的指针

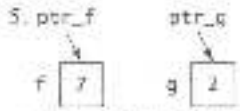
2.



3. 非法，参数不是指针

4. 非法，参数不是指向 f 和 g 的指针

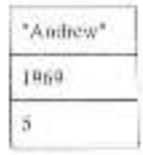
5.



6. 非法，参数不是指针

7.1 节

1.



2.

3.

7.1 节

1. Audrey
   Frederic
2. Category 4 Hurricane: Audrey

8.3 节

1. 15012.368
2. 15012.368
3. 150
   12.368
4. 150
   12
5. 150,12.4
6. 150,12.368

435

“修改”的部分答案

2.4 节

1.
```
/*-----------------------------------------------------------*/
/*  程序 chapter2-mod                                        */
/*                                                           */
/*  该程序以 0 为除数来查看系统的反应                          */

#include <stdio.h>

int main(void)
{
   /*  声明和初始化变量  */
   double a=5, b=0, c;

   /*  执行以 0 为除数的除法  */
   c = a/b;

   /* 输出 c 的值  */
   printf("c = %f \n",c);

   /* 退出程序  */
   return 0;
}
/*-----------------------------------------------------------*/
```

4.5 节

```
/*-----------------------------------------------------------*/
/*  程序 chapter4_5mod                                       */
/*                                                           */
/*  该程序生成并输出在用户输入范围内的 10 个随机浮点数          */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   /*  声明变量和函数原型  */
   unsigned int seed;
   int k;
   double a, b;
   double rand_float(double a, double b);

   /*  得到种子值和问题范围  */
   printf("Enter a positive integer seed value: \n");
   scanf("%u",&seed);

   srand(seed);
   printf("Enter limits a and b (a<b): \n");
   scanf("%f %f",&a,&b);

   /*  生成并输出 10 个随机数  */
       printf("Random Numbers: \n");
   for (k=1; k<=10; k++)

      printf("%f ",rand_float(a,b));

   /* 退出程序  */
```

```
    return 0;
}
/*-----------------------------------------------------------------*/
                    (4.5.2 节的 rand_float 函数)
/*-----------------------------------------------------------------*/
```

5.6 节

```
1. /*-----------------------------------------------------------------*/
   /*  程序 chapter5_mod                                              */
   /*                                                                 */
   /*  该程序初始化数组，然后使用选择排序将数组重新排列              */

   #include <stdio.h>

   int main(void)
   {
   /*  声明变量和函数原型  */
      int k;
      double x[10]={4,8,-2,16,19,6, 4,0,20,3};
      void sort(double x[], int n);

      /*  输出初始序列  */
      printf("Original Order \n");
      for (k=0; k<=9; k++)
         printf("%.1f ",x[k]);
      printf("\n");

      /*  排序  */
      sort(x,10);

      /*  输出新序列  */
      printf("New Order \n");
      for (k=0; k<=9; k++)
         printf("%.1f ",x[k]);
      printf("\n");

      /*  退出程序  */
      return 0;
   }
   /*-----------------------------------------------------------------*/
                              (5.6 节的 sort 函数)
   /*-----------------------------------------------------------------*/
```

[437]

章末简述题的完整答案

第 1 章

1. F
2. T
3. F
4. T
5. F
6. F
7. F
8. F
9. F
10. T
11. T
12. F
13. F
14. T
15. T
16. F
17. F
18. F
19. F
20. T
21. (d)
22. (b)
23. (c)
24. (a)
25. (a)
26. (b)
27. (e)
28. (c)
29. (d)
30. (c)
31. (a)
32. (b)
33. (c)
34. 程序
35. 硬件
36. 中央处理单元
37. 输出设备
38. 系统软件
39. 算法
40. 编译
41. 电子表格
42. 语法
43. 操作系统
44. 算术逻辑单元
45. 调试
46. 逻辑错误
47. ANSI C
48. 微处理器
49. 超级计算机
50. 机器语言

第 2 章

1. T
2. F
3. F
4. T
5. F
6. 错误

```
int i, j, k;
```

7. 正确
8. 错误

```
double D1, D2, D3;
```

9. 错误

```
float a1, a2;
```

10. 正确
11. (d)
12. (h)
13. (c)
14. (c)
15. (e)

16. x1 | 2 |
17. a | 3.8 |
18. x | 2.8 |
19. n | 2 |

18. value_1=5.783
19. value_4= 6.65e01
20. value_5= 17750

第 3 章

1. T
2. F

3. T
4. F
5. T
6. T
7. 错误，逗号处应该是分号
8. 错误，应该是 k==1
9. 错误，控制表达式应该是整数
10. (c)
11. (b)
12. (a)
13. (a)
14. (c)
15. (c)
16. (c)
17. a

7500

第 4 章

1. T
2. F
3. T
4. T
5. (b)
6. (a)
7. (d)
8. 1
9. 0
10. 0
11. 如果输入数为负，结果就是系统相关的，因此，函数只能使用正整数

第 5 章

1. T
2. F
3. F
4. F
5. (a)
6. (a)
7. (a)
8. (b)
9. k

4

t

5	8	11	14	17

10. r

4

c

5

s

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7

11. 5
12. 6

第 6 章

1. T
2. T
3. F
4. F
5. (a)
6. (b)
7. (d)
8. (b)

9.

name	20.5
x	20.5
n	14

10. 5

11. 12

12. 5

13. 5

第 7 章

1. T

2. F

3. T

4. T

5. T

6. (b)

7. (b)

8. (d)

9. (a)

10.

start_datemonth	?	month
	?	day
	?	year
end_datemonth	?	month
	?	day
	?	year

11.

start_datemonth	9	month
	?	day
	?	year
end_datemonth	12	month
	?	day
	?	year

12.

start_datemonth	9	month
	?	day
	2005	year
end_datemonth	12	month
	?	day
	2006	year

13.

start_datemonth	9	month
	?	day
	2005	year
end_datemonth	12	month
	30	day
	2008	year

第 8 章

1. T
2. T
3. T
4. F
5. T
6. T
7. F
8. T
9. T
10. F
11. (c)
12. (c)
13. (b)

441

章末编程题的部分答案

第2章

```
/*-----------------------------------------------------------*/
/*   程序 chapter2_prob39                                    */
/*                                                           */
/*   该程序计算氨基酸的分子量                                */

#include <stdio.h>
#define O 15.9994
#define C 12.011
#define N 14.00674
#define S 32.066
#define H 1.00794

int main(void)
{
    /*  声明变量  */
    int num_o, num_c, num_n, num_s, num_h;
    double weight;

    /* 提示用户输入原子的个数  */
    printf("Enter number of oxygen atoms: \n");
    scanf("%i",&num_o);
    printf("Enter number of carbon atoms: \n");
    scanf("%i",&num_c);
    printf("Enter number of nitrogen atoms: \n");
    scanf("%i",&num_n);
    printf("Enter number of sulfur atoms: \n");
    scanf("%i",&num_s);
    printf("Enter number of hydrogen atoms: \n");
    scanf("%i",&num_h);

    /*  计算分子量  */
    weight = O*num_o + C*num_c + N*num_n +
             S*num_s + H*num_h;

    /*  输出分子量  */
    printf("amino acid molecular weight = %.5f \n",
           weight);

    /*  退出程序  */
    return 0;
}
/*-----------------------------------------------------------*/
```


第3章

```
/*-----------------------------------------------------------*/
/*   程序 chapter3_prob37                                    */
/*                                                           */
/*   该程序输出一个表格，显示20年来每年年末重新绿化土地的英亩数  */

#include <stdio.h>
#define UNCUT 2500
#define RATE 0.02

int main(void)
```

```
{
   /*  声明变量  */
   int year;
   double old_forest=UNCUT, new_forest;

   /*  输出报告  */
   printf("Reforestation Summary \n");
   printf("Year    Total Acres Forested \n");
   for (year=1; year<=20; year++)
   {
      new_forest = old_forest*RATE;
      old_forest += new_forest;
      printf("%4i    %10.2f \n",year,old_forest);
   }

   /*  退出程序  */
   return 0;
}
/*----------------------------------------------------------*/
```

第 4 章

```
/*----------------------------------------------------------*/
/*  程序 chapter4_prob14                                    */
/*                                                          */
/*  该程序模拟投掷两个六边骰子，计算出点数和为 8 占投掷总次数的百分比 */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   /*  声明变量和函数原型  */
   unsigned int seed;
   int rolls, k, die_1, die_2, sum=0;
   int rand_int(int a, int b);
   /*  得到种子值  */
   printf("Enter a positive integer seed value: \n");
   scanf("%u",&seed);
   srand(seed);

   /*  提示用户输入掷骰子的总次数  */
   printf("Enter number of rolls of dice: \n");
   scanf("%i",&rolls);

   /*  模拟掷骰子  */
   for (k=1; k<=rolls; k++)
   {
      die_1 = rand_int(1,6);
      die_2 = rand_int(1,6);
      if (die_1+die_2 == 8)
         sum++;
   }

   /*  计算并输出和为 8 占总次数的百分比  */
   printf("Number of rolls: %i \n",rolls);
   printf("Percent with sum of eight: %.2f \n",
         sum*100.0/rolls)

   /*  退出程序  */
   return 0;
}
/*----------------------------------------------------------*/
            包含 4.5.1 节的函数 rand_int。
/*----------------------------------------------------------*/
```

443

444

第 6 章

```
/*                                                          */
/*   程序 chapter6_prob18                                   */
/*                                                          */
/*   该函数确定数组中正数、负数和 0 的个数                   */

void signs(int x[], int npts, int *npos,
           int *nzero, int *nneg)
{
    /*  声明变量  */
    int k;

    /*  将总数设置为 0  */
    *npos = *nzero = *nneg = 0;

    /*  更新相应的总数目  */
    for (k=0; k<=npts-1; k++)
    {
       if (x[k] < 0)
          *nneg += 1;
       else
          if (x[k] > 0)
             *npos += 1;
          else
             *nzero += 1;
    }

    /*  无返回值  */
    return;
}
/*                                                          */
```

445

术 语 表

ANSI C 美国国家标准协会标准，提供了系统无关的、明确的C语言定义。

ASCII 美国标准信息交换码。

bug 程序中的错误。

EBCDIC 扩展二进制编码的十进制交换码。

EOF字符 在文本文件末尾，指示已经到达文本最后的特殊字符。

FIFO结构 队列的另一种叫法，或者称为先进先出结构。

for循环 执行指定次数的循环。

LIFO结构 堆栈的另一种说法，或称为后进先出结构。

while循环 只要条件为真就执行的循环。

编译错误 程序编译时发生的错误提示。

编译器 将程序从高级语言翻译成机器语言的程序。

变量 一些被赋予了名称的内存存储单元，在程序执行时其内容可能会也可能不会发生改变。

标签 和定义结构相关联的名称。

标识符 一个特定格式的名称字符串，用于引用在内存单元中的存储值。

标准C库 可以由C程序访问的常量和函数库。

标准差 方差的平方根。

表达式 由常量、变量和操作符组成的式子，并且最终可以计算出唯一值作为结果。

参数 函数的输入。

操作系统 在用户和硬件之间提供接口的软件。

测试数据 用于检测程序正确性的数据。

常量 像3.141593这样在程序执行时不会发生改变的值。

超平面 由超过三个变量的方程所表示的空间。

成员函数 和类相关联的函数。

程序 描述由计算机执行操作的指令集。

程序走查 为了检查一个解决复杂问题而设计的算法或程序是否正确，向一组新人逐步介绍详细的解决步骤，以获得他们的反馈和建议的方法。

抽象 这个概念是指，程序员使用一个模块来完成指定任务，而不需要了解该模块中的详细步骤。

初值 第一次赋给变量的值（经常包含在声明语句中）。

处理器 控制计算机其他部分的组件。

传值调用 在函数调用时，将实参的值传递给对应的形参的调用方式。

串联 首尾相连。

存储类型 确定变量使用范围的类型名称。

错误条件 在程序预期的执行过程中不应该发生的条件。

大小写敏感 小写字母和大写字母被认为是不同的字符。

单目运算符 对单个数值进行操作的运算符（比如负号）。

地址 唯一定义一个存储单元的正整数。

传址调用 在函数调用时，将实参地址作为对应的参数传递给形参的调用方式。

地址运算符 确定一个标识符所在内存地址的单目运算符。

递归 通过调用自身来求解问题的方法。

点操作符 类对象调用成员函数时使用的操作符。

点积 两个向量相应位置元素的乘积之和。

电子表格 可以将信息以包含行和列的表格形式显示的软件工具。

电子副本 存储在计算机或者其他形式（如CD等）中，可以由计算机读取的信息。

调试 识别并消除程序中的漏洞或错误的过程。

调试器 帮助识别并消除程序中漏洞或错误的程序。

调用 调用或引用一个函数（或模块）。

迭代 一个循环语句中的循环体执行一次。

动态内存分配 允许程序员在程序运行时指定内

存分配的技术。

动态数据结构 当数据增加时，通过使用动态内存分配使空间增大的数据结构。

队列 节点从一边加入，从另一边移除的数据结构，也称为先入先出（FIFO）结构。

对象 类的实例。

多态性 借助同一个标识符在运行时调用多种方法的能力。

多重赋值 在一条语句中连续为多个变量赋值的方法。

二叉树 具有左分支和右分支的节点链表。

二进制 有两种状态，通常用0和1表示。

二进制码 由0和1组成的码。

二维数组 能够可视化表示为具有行和列的表或是网格的数据结构。

范围 程序中能够有效引用函数和变量的部分。

范围限定运算符 符号::用在类名和函数名之间，用来表明该函数是类的成员。

方差 计算一组数与其平均值的差值，然后再计算这些差值的平方和的均值。

方阵 具有相同行数和列数的矩阵。

非奇异 指一组方程仅有唯一解的情况。

斐波那契数列 一个序列，它从1开始，接着每个随后的值都是前面两个值的和。

分解提纲 解决问题的必要步骤的概述。

分治 将大问题分解为小问题并求解的策略。

浮点型值 可以表示整数和非整数的值的一种数据类型。

符号常量 通过预处理命令分配给标识符的常量。

复合语句 由括号括起来的一组语句。

重用性 软件按照模块化的思路设计和开发，以便于代码能在解决多个问题的过程中重复使用。

赋值语句 为标识符赋值的语句。

高级语言 不针对于特定类型CPU的指令语言，程序语句通常类似英语。

高斯消元法 求解联立方程组的一种数值方法。

格式化标志 C++中格式化输出的标志。

个人计算机（PC） 基于微处理芯片设计而成的，体积较小、价格不昂贵的计算机。

根 能使$f(x)=0$的x值。

公有成员 用户程序中随时都可以引用的类的成员函数。

构造函数 提供类对象自动初始化的类成员函数。

关键字 对于C编译器有特定意义的命令。

关系运算符 比较两个表达式的运算符。

过零点 函数与x轴的交叉点。

函数 在程序执行中可以调用的独立模块，最多可以返回一个数值。

函数原型 标识调用函数的必要信息的语句。

函数组合 就是函数嵌套。

宏 预处理命令，可以用于定义一些简单的函数。

后缀 位于标识符之后的位置。

汇编程序 将汇编语言程序转换为二进制的程序。

汇编语言 针对特定类型的CPU的机器指令而设计的编程语言，但是其用使用的词汇类似于英语的单词。

混合运算 两个不同类型值之间的运算。

机器语言 将指令写为二进制中的语言。

计算机 能够运行程序指定操作的机器，其中程序是由一系列指令构成的。

计算机仿真 使用随机数来模拟特定事件的计算机程序。

继承 类从现存的类中继承其属性的能力。

寄存器类 一个被频繁访问的变量，在存储时放在寄存器中以提高执行效率。

间接引用 引用存储在指针中的地址所指向的内存单元中的值的操作。

间接引用 引用存储在指针中地址对应的内存单元中数值的操作。

阶乘 一个正整数的函数，计算一个整数与在它和1之间所有整数的乘积。

节点 由数据和指向另一个节点的指针组成的结构。

结构 一些不同数据类型或相同数据类型的变量组成的集合。

结构成员操作符 用来分隔结构变量名和数据成员名的句点。

结构化程序 由简单控制结构组织问题解决方法的程序。

结构图 显示程序模块结构的图。

结合性 在表达式中对各种运算符分组的性质。

结束信号 在数据文件尾部用于表明已访问到文件末尾的值。

截断 将数值部分丢弃。

精度 科学计数法中由用户指定的小数位数。

静态类型 一种变量类型的名称，在整个程序执行期间保留为这种类型的变量分配的内存。

局部变量 有效范围在所定义函数之内的变量。

矩阵 排列在具有行和列的矩形网格中的一组数。

矩阵乘法 由两个矩阵计算生成一个新矩阵的运算。

均方值 一组数的平方加和的平均值。

科学计数法 将值表示为10以内的尾数乘以10的幂次方的计数法，如3.1×10^2。

空白字符 空格字符或者以下之一的字符：FF(换页)，NL(换行)，CR(回车)，HT(水平制表符)，VT(垂直制表符)。

空链表 用于指向第一个节点的表头指针是一个空指针的情况。

空指针 通常用于函数的返回值的指针类型，该函数没有指明指针指向的变量的类型。

空字符 值为二进制0的常量。

控制表达式 用在switch语句中作条件判断的表达式。

控制字符 以下字符之一：FF（换页），NL（换行），CR（回车），HT（水平制表符），VT（垂直制表符），BEL（响铃），BS（退格）。

控制字符串 在输出语句中指定输出格式的字符串。

库函数 通常是由编译环境提供的具有一定通用性的函数。

垃圾值 在变量未被初始化之前，内存单元中保留的前一程序的无意义数据。

类 由数据和函数组成的抽象数据类型。

类声明 声明类的名称，数据成员和函数成员的C++语句。

类实现 给出类中所有函数成员定义的C++语句。

类型说明符 区分C语言存储的各种数值形式的术语。

联立方程组 具有公共解的一组方程。

联立线性方程组 有公共解的一组线性方程。

链表 每个数据成员都包含信息，并通过指针连接到下一数据成员的数据结构。

链接/加载 为目标程序准备好执行所需条件的过程。

流 一串顺序的字符。

流插入操作符 和cin对象一同使用的字符>>。

流程图 一种用于描述算法步骤的示意图。

流提取运算符 与cout对象一同使用的字符<<。

逻辑错误 解决问题时逻辑步骤中出现的错误。

逻辑运算符 用来比较条件的运算符。

面向对象的程序设计 使用类进行编程的方法。

模块 一组功能相对独立的语句，用于实现某个特定的操作或者某种数值的计算方法。

模块化 将问题的解决方案分解成一组模块分别处理的方法。

模块图 显示程序模块结构的图。

默认构造函数 当对象被定义但没有被初始化时调用的构造函数。

目标程序 机器语言程序。

内存 计算机存储信息的部分。

内存快照 在程序执行时，显示指定时刻存储单元内容的图。

内核 操作系统中用来管理硬件和软件应用程序之间接口的组成部分。

内积 即点积。

排序 将一组数升序或降序排列的方法。

偏移量 表示当前数组元素在内存中的位置与数组第一个元素的距离。

平均值 一组数的平均值。

前缀 位于标识符之前的位置。

强制类型转换 在使用数值计算之前将其转换为另一种数据类型的过程。

求和记号 描述一组数字之和的数学符号。

驱动程序 为测试函数提供简单接口的程序。

取模 计算一个整数除以另一个整数得到的余数的运算。

全局变量 在主函数或其他用户自定义函数之外定义的变量。

缺省标签 在switch语句中指明没有其他语句执行时要执行语句的标签。

软件 描述计算机执行步骤的一组程序。

软件工具 用来执行一些常见的实用操作的程序，比如生成报告或图形。

软件生命周期 大型软件项目开发过程中的多个阶段。

软件维护 对现有软件进行必要的修复和增强工作，包括需要修正软件中发现的错误，更新软件以使它可以与新的硬件和软件适配等。

软件原型 不包含最终系统所有功能的软件包。

但是具备绝大多数的用户接口，用于对软件进行初步的评估。

刚条件 表示联立方程组缺少唯一解的一种说法。

三角函数 计算三角或逆三角函数值的函数。

上溢 由于算术运算的结果太大，以至于不能存储在分配给它的内存中而导致错误。

哨兵标记 在文件结束处，表明已访问到文件末尾的值。

声明 定义存储在内存中的变量的语句。

实参 当函数被调用时用来赋值给形参的值。

实时程序 使用汇编语言编写，执行速度非常快的程序。

实用程序 能实现常见功能的程序，如将硬盘的文件复制到CD。

输入输出图 定义程序输入和输出的简单框图。

数据成员 和结构体相关的变量。

数据库管理工具 操作和管理海量数据的软件工具。

数据文件 包含程序要读取数据的文件或存储生成数据的文件。

数学函数 计算常见数学函数值的函数，如计算x的平方根。

数组 是一种数据结构，允许使用一个公有名称表示一组元素，并用下标来区分各元素。

双目运算符 具有两个运算变量的运算符，比如加法。

双曲线函数 自然对数或自然指数函数。

双向链表 每个节点都包含一个正向指针和一个反向指针的特殊链表。

顺序 使步骤相继顺序执行的一种控制结构。

顺序搜索 从列表第一个值开始，顺序查找指定值的搜索算法。

顺序序列 从低到高的字符串顺序。

私有成员 仅能由类中的其他成员函数引用的成员函数。

算法 问题解决方法的步骤概要。

算术逻辑单元（ALU） 执行算术和逻辑运算的计算机组件。

随机数 由统计特征定义的数字，而无法用确定的公式描述。

随机数种子 用来初始化随机数序列的值。

缩略赋值 使用缩略形式的赋值语句。

提示信息 程序输出在屏幕上的信息，一般用于提示用户需要输入数据。

条件 能被判定为正确或错误的表达式。

条件运算符 具有三个参数的三目运算符，三个参数为：条件，当条件正确时执行的语句和当条件错误时执行的语句。

头指针 在链表中指向第一个节点的指针。

外部类 全局类对象的类定义，一般这些类对象的作用域是整个程序。

网络 将计算机连接起来，使其可以共享资源和信息。

微处理器 包含在单个集成电路芯片中，比邮票还小的CPU。

伪代码 描述算法步骤的一组类似自然语言的语句。

位 二进制数字0或1。

文件打开模式 指明数据文件状态的字符。

文件结束指示符 在文件结尾指示已经到达文件末尾的特殊字符。

文件指针 指向数据文件的指针变量。

文字处理软件 能够进行文本输入和格式化处理的软件工具，可以用于书写报告，也可以用于编写计算机程序。

问题求解过程 解决新问题的方法。

系统相关 有些特性或功能不是在所有计算机系统上都有效，或者效果都一致。

下标 用来区分数组元素的整数。

下溢 由于算术运算计算出的结果太小，以至于未能存储在分配给它的内存中而导致的错误。

线性插值 一种数值技术，通过假设函数值落在两点之间的直线上来估计函数值。

线性回归 用于确定与一组数据的匹配度最高的直线方程的数值技术。

线性模型 一维数据在一条直线上的模型。

向量 仅有一行或一列的矩阵。

行列式 由矩阵元素计算出的特定值。

形式参数 函数定义中表示输入值的标识符，简称为形参。

选择 一种控制结构，当条件为真时需要执行一组步骤；当条件为假时执行另一组步骤。

选择标签 在switch-case语句中，标识一个选择结构的表达式。

选择结构 在switch-case语句中，当符合某个条件时连续执行的一组语句。

选择排序算法 一种排序算法，需要在数组中多次执行，每次将最小值交换到指定位置。

循环 重复执行的一组语句。

循环控制变量 控制 for 循环的变量。

循环链表 最后一个节点指向第一个节点的链表。

验证和确认 这两个过程旨在验证程序是否正确实现其目标，并且这些目标能否解决手头的问题。

一维数组 一种存储一组数据的数据结构，可以被可视化表示为一行或一列数据。

引用调用 在函数调用时，将实参地址作为对应的参数传递给形参的调用方式。

硬件 像键盘、鼠标和硬盘一类的计算机设备。

优先级 表达式中执行运算的顺序。

右对齐 使值的右侧没有空格的对齐方式。

语法 语言的语法规则。

语句 程序中的注释或指令。

预处理命令 向编译器发出指令的语句。

元素 数组中的值。

源程序 高级语言的程序代码。

云计算 一种能够远程访问大规模计算和存储的信息技术。

增量查找 估计函数根的一种数值方法。

栈 最后添加的元素会被最先移除的链表，也被称为后进先出（LIFO）结构。

折半查找 是一种搜索算法，每次比较都将元素的范围折半，以减少需要查找的次数。

振幅 信号上下波动范围的绝对值。

执行 由程序描述的步骤执行过程。

指数计数法 使用字母 e 将科学计数法中尾数和指数分开的计数法，如 3.1e02。

指针 存储另一个变量的地址的变量。

指针运算符 当以类对象的指针或结构的指针来访问其数据成员时使用的运算符，作用与成员运算符类似。

中位数 如果一组排好序的数据有奇数个数字，则中位数为其中间值，否则，是中间两个数的平均值。

中央处理器（CPU） 控制器和 ALU 的组合。

重复 包含一组需要多次执行的步骤的控制结构，只要条件为真，就重复执行这组步骤。

重载 根据不同的数据类型，为运算符赋予不同的含义。

逐步提炼 将问题解决方法分解成一组更小的执行步骤的过程。

注释 程序中的语句但不是指令，用来在程序中标注和解释执行步骤。

转换标识符 在显示输出用的字符串中，用于描述输出值格式的标识符。

转换操作符 单目运算符，在进行下一步的计算之前转换一个值的数据类型。

转义字符 在控制串中使用的反斜杠（\）。

转置 将初始矩阵行和列互相对换而产生另一个矩阵。

桌面印刷 由高质量的打印机和功能强大的文字处理软件制作专业文档的过程。

字段宽度 用于控制输出值在屏幕上占据字符数的最小数目。

字符 一种表示数字或其他文字符号信息的数据类型。

字符串 以空字符结尾的字符数组。

[字符串] 分析 逐个检查数组或字符串中的每个字符。

字符函数 具有字符参数或者返回字符值的函数。

字节 由 8 个位（或者 8 位二进制数）组成的内存单元。

自顶向下设计 从解决方案的整体框架描述开始，然后逐步提炼和精简解决步骤的一种设计方法。

自定义函数 由程序员编写的函数。

自动类型 表示局部变量的类。

最小二乘法 将模型和给定函数或给定点集之间的方差最小化的技术。

左对齐 使值的左侧没有空格的对齐方法。

索　引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

() parenthesis（圆括号），37；
cast operator（强制类型转换运算符），38
[] subscript brackets（下标括号），214
{} braces（花括号），27
<> library header file indicators（库头文件标识符），26
/* begin comment（开始注释界定符），26
*/ close comment（结束注释界定符），26
++ autoincrement（自动增量），41
-- autodecrement（自动减量），41
+ unary plus（单目加），37；
binary
addition（二进制加），37
- unary minus（单目减），37；
binary
subtraction（二进制减），37
! unary not（单目非），89
& address operator（地址运算符），48，290
* dereference operator（解引用运算符），292; multiplication
operator（乘法运算符），37
/ division operator（除法运算符），37
\ backslash（反斜杠），46
% modulus operator（取模运算符），37；
conversion specifier（转换说明符），44，47
< less than（小于），88
<= less than or equal to（小于等于），88
> greater than（大于），88
>= greater than or
equal to（大于等于），88
== is equal to（相等条件判断），88
!= is not equal to（不相等条件判断），88
&& logical and（逻辑与），89
|| logical or（逻辑或），89
?: conditional operator（条件运算符），94
= equals（赋值），36，37
+= abbreviated addition（相加并赋值的缩略写法），42
-= abbreviated
subtraction（相减并赋值的缩略写法），42
*= abbreviated
multiplication（相乘并赋值的缩略写法），42
/= abbreviated division（相除并赋值的缩略写法），42
%= abbreviated modulus（取模并赋值的缩略写法），37
>> extraction operator（提取运算符），377
<< insertion operator（插入运算符），375
// C++ comment（C++中的注释），374
, comma operator（逗号运算符），107
; semicolon（分号），27，92
. dot operator（点运算符），376
\a alert or bell
character（警报或响铃），46
\b backspace character（退格），46
\f formfeed（换页），46
\n newline（换行），46
\r carriage return（回车），46
\t horizontal tab（水平制表符），46
\v vertical tab（垂直制表符），46
\\ backslash character（反斜杠字符），46
\? question mark（问号），46
\' single quote（单引号），46
\" double quote（双引号），46
#define directive（宏定义指令），34
#include directive（包含指令），26
1GL（第一代计算机语言），13
2GL（第二代计算机语言），13
3GL（第三代计算机语言），13
4GL（第四代计算机语言），14
5GL（第五代计算机语言），14

A

Abbreviated
  assignment（缩略赋值），42
  operator（缩略运算符），42
Abort（中止），102
abs function（绝对值函数），61
Absolute value（绝对值），66
Abstraction（抽象），150
acos function（反余弦（acos）函数），62
Actual parameter（实际参数），158
Addition
  matrix（加法矩阵），263
  symbol（加法符号），37
Address（地址），289
  arithmetic（地址的算术运算），295
  operator（地址运算符），48，290
Advanced composite materials（先进复合材料），6
Aerospace engineering examples（航空航天工程实例），78，143，208，282
Algorithm（算法），18
  search（搜索算法），244
  sort（排序算法），242
Allocation of memory（内存分配），11，321
Alphanumeric character（字母数字字符），407
Alternative solutions（替代解决方案），85
ALU（算术逻辑单元），11
Amino acid（氨基酸），77
American National Standard Institute（ANSI，美国国家标准协会），16
  C Standard Library（美国国家标准协会的标准C语言库），13，407
American Standard Code for Information Interchange（ASCII，美国信息交换标准编码），41，418
Analysis（分析），10
ANSI C（美国国家标准协会C语言），13
  Standard library（美国国家标准协会C语言标准库），13，407
Application satellites（应用卫星），5
Apollo spacecraft（阿波罗宇宙飞船），4
Area（面积），76，191，192
Argument（参数），61
Arithmetic（算术运算）
  floating point（浮点数运算），30
  integer（整数运算），30
  logic unit（ALU，算术逻辑单元），11
  mixed（混合运算），37
  operations（算术操作），36
  operator（算术运算符），37
  operator precedence（算术运算符优先级），38，39
  pointer（指针的算术运算），295
Array（数组），
  declaration（数组声明），214，249
  element（数组元素），214
  five-dimensional array（五维数组），279
  four-dimensional array（四维数组），278
  function argument（数组做函数参数），218
  higher-dimensional arrays（高维数组），277
  initialization（数组的初始化），214
  one-dimensional（一维数组），213
  storage order（数组元素的存储顺序），300
  subscript（数组下标），214
  three-dimensional（三维数组），277
  two-dimensional（二维数组），248，300
ASCII code（ASCII编码），33
  table（ASCII码表），418
asin function（反正弦函数），62
Assembler（汇编器），15
Assembly language（汇编语言），13
assert.h standard header file（assert.h标准头文件），407
Assignment（赋值）
  abbreviated（缩略赋值），42
  multiple（多重赋值），35
  operator（赋值运算符），35
  statement（赋值语句），35
Associativity（结合性），39
atan function（反正切函数），62
atan2 function（反正切函数），62
Atomic weights（原子量），77
  elements（元素的原子量），226
Atmospheric layers（大气层），131，132
Autodecrement（自动减量），41
Autoincrement（自动增量），41
auto storage class specifier（自动存储类型说明

符），161
Automatic storage class（自动存储类型），161
Average（均值），232

B

B language（B 语言），13
Back substitution（回代），271
Backslash character（反斜杠字符），46
BCPL language（BCPL 语言），13
Best fit（最佳适配），129
Binary（二进制），13
  code（二进制编码），33
  language（二进制语言），13
  operator（双目运算符），37
  search（二分搜索），246
  tree（二叉树），364
Biomedical engineering example（生物医学工程实例），144
Biometric，4-color insert（生物特征识别），彩页
Bit（比特），13
Blank line（空行），28
Block of statements（语句块），91
break statement（break 语句），96，107
Bug（程序错误），14
Byte（字节），321

C

C
  language（C 语言），13，14
  program structure（C 程序结构），28
C++
  language（C++ 语言），13，14
  program structure（C++ 程序结构），374
CAD/CAM（计算机辅助设计/计算机辅助制造），5
Call（调用）
  by address（传址调用），160，220
  by reference（传值调用），160
  by value（传值调用），160，220
calloc function（清零的动态内存分配函数），321
Case
  label（分支语句的选择标签），96
  sensitive（区分大小写），29
  structure（选择结构），96
Cast operator（强制类型转换运算符），38
CAT scan（计算机 X 射线轴向分层造影扫描），6
cell function（cell 函数），61
Central processing unit（CPU，中央处理单元），11
char data type（字符数据类型），34
Character（字符），65
  comparison（字符比较），33，66
  constant（字符常量），33
  data（字符数据），33
  function（字符函数），66
  input/output（字符输入/输出），65
  string（字符串），314
  variable（字符变量），33
Chemical engineering examples（化学工程实例），143，226，285
cin object（cin 对象），377
Circularly linked list（循环链表），361
class（类），373，389
  declaration（类声明），389
  implementation（类实现），389
cloud computing（云计算），10
Code（编码）
  ASCII（ASCII 编码），33
  binary（二进制编码），33
  EBCDIC（扩充二进制编码的十进制编码），33
Coercion of arguments（函数参数的强制类型转换），159
Cofactor（余子式），286
Collating sequence（整理序列），244
Column（列）
  pivoting（列主元），272，286
  vector（列向量），260
Combination（合并），211
Comma operator（逗号运算符），107
Comment（注释），26
Communication skills（沟通技巧），9
Compile（编译），14
Compiler（编译器），14
  error（编译错误），14
Complex（复数）
  class（复数类），396
  data（复数数据），396

Composite materials（复合材料），6
Composition（构成），61
Compound statement（复合语句），91
Computer（计算机），10
  aided design（CAD，计算机辅助设计），5，13
  aided manufacturing（CAM，计算机辅助制造），5
  engineering examples（计算机工程实例），206，284，332
  hardware（计算机硬件），10
  language（计算机编程语言），13
  organization（计算机组成），11
  simulation（计算机仿真），181
  software（计算机软件），10，11
Computerized axial tomography（CAT，计算机x射线轴向分层造影），6
Concatenate（连接），316
Condition（判定条件），83
Conditional（条件）
  expression（条件表达式），88
  operator（条件运算符），94
Constant（常量），29
Constructor function（构造函数），392
  default（默认构造函数），393
continue statement（continue语句），107
Control（控制）
  character（控制字符），67
  string（控制字符串），44
  structure（控制结构），82
Controlling expression（控制表达式），96
Conversion（转换）
  input specifier（输入转换说明符），47
  output specifier（输出转换说明符），44
cos function（余弦函数），62
cosh function（双曲余弦函数），64
Counter controlled loop（用计数器控制的循环），104
cout object（cout对象），375
CPU（中央处理单元），11
Crime scene investigation, 4-color insert（犯罪现场调查，彩页，2，24，48，97，163，236，318，342，382
Critical path analysis（关键路径分析），145
Cryptography（密码学），284
ctype.h standard header file（ctype.h标准头文件），73，407
Cubic spline interpolation（三次样条差值），52
Currency conversions（货币换算），142
Custom header file（自定义头文件），235

D

Data（数据）
  file（数据文件），87，116
  member（数据成员），335，389
  type（数据类型），31
  window（数据窗口），309
Database management（数据库管理），13
Debug（错误修正），14
Debugging（调试），14
Debugging notes（调试说明），27
Declaration（声明），27
Decomposition outline（分解提纲），18，82
Decrement operator（减量运算符），41，
Default（默认）
  constructor function（默认构造函数），393
  define directive（默认宏定义指令），34
  define disector（默认宏解析器），34
  label（分支语句的默认标签），96
Dereference operator（（依据地址的）取值运算符），292
Design/process/manufacture path（设计/处理/制造过程），9
Desktop publishing（个人出版（桌面印刷）），12
Determinant（行列式），261，286
Directive（指令），34
Divide and conquer（分治），82
Division（除法），37
  by zero（被零除），14
DNA analysis（DNA分析），288，318
do statement（do语句），103
do/while loop（do/while循环），103
Dot（点）
  operator（点运算符），376
  product（点积），260
double
  data type（double数据类型），27，32
  limits（double类型的数据范围），32
Double precision（双精度），32

Doubly linked list（双链表），362
Driver（驱动程序），152
Dynamic（动态）
  data structure（动态数据结构），353
  memory allocation（动态内存分配），321

E

EBCDIC（扩充二进制编码的十进制编码），33
El Nin-o（厄尔尼诺现象），303
El Nin-o-Southern Ocillation(ENSO，厄尔尼诺－南方涛动），303
Electrical engineering examples（电气工程实例），272，282，283
Electronic copy（电子画本），11
Element（元素），214
Empty（空的）
  list（空链表），355
  statement（空语句），92
End-of-file indicator（文件结束标识符），124
endl（endl 标识符），375
Engineering（工程）
  achievements（工程成果），3
  Problem Solving methodology（工程问题求解方法论），16
ENSO（厄尔尼诺－南方涛动），303
Environmental engineering examples（环境工程实例），131，144，145，309
EOF（文件结束），65
  character（文件结束符），65
Error condition（错误条件），86
errno.h standard header file（errno.h 标准头文件），408
Equator（赤道），169
Escape（转义）
  character（转义字符），46
  sequence（转义序列），46
Execution（执行），14
Exosphere（外大气层，外逸层），132
exp function（exp 函数），61
Exponent（指数），30
  overflow（指数上溢），41
  underflow（指数下溢），41
Exponential notation（指数计数法），30
Exponentiation（幂运算），61
Expression（表达式），35
Extended Binary Coded Decimal Interchange Code（EBCDIC，扩充二进制编码的十进制编码），33
extern storage class specifier（extern 存储类型说明符），161
External storage class（外部存储类型），161
Extraction operator（提取运算符），377
Extrapolation（推断法），131

F

fabs function（fabs 函数），61
Face recognition（人脸识别），80，97
Factorial（阶乘），200，211
fclose function（fclose 函数），118
Fibonacci sequence（斐波那契数列），202
Field width（字段宽度），45
FIFO（先进先出），363
Fifth generation language（5GL，第五代计算机语言），14
File（文件），116
  close（文件关闭），118
  header（头文件），27，235
  input（输入文件），117，119
  open mode（文件打开模式），117
  output（输出文件），126
  read（读文件），117
  pointer（文件指针），117
  stream（文件流），378
  write（写文件），126
FILE data type（FILE 文件数据类型），117
Finger print recognition（指纹识别），334，342
First-in-first-out（FIFO，先进先出），363
Five-dimensional array（五维数组），279
flight simulator（飞行模拟器），208
float
  data type（float 数据类型），31，32
  limits（float 类型数据范围），32
float.h standard header file（float.h 标准头文件），71，408
Floating-point（浮点），30
  conversion to integer（浮点型－整型转换），36

declaration（浮点类型声明），32
limits（浮点类型数据范围），32
number（浮点数），30
overflow（浮点型上溢），41
precision（浮点型精度），31
underflow（浮点型下溢），41
value（浮点型数值），31
floor function（floor 函数），61
Flowchart（流程图），82
fopen function（fopen 函数），117
for loop（for 循环），104
Forensic anthropology（法医人类学），24，48
Forestry（林业），144
Formal parameter（形式参数），158
Format flag（格式化标志），376
Four-dimensional array（四维数组），278
Fourth generation language（4GL，第四代计算机语言），14
fprintf function（fprintf 函数），118
free function（free 函数），321
fscanf function（fscanf 函数），118
fstream.h header file（fstream.h 头文件），378
Function（函数），149，156
argument（函数参数），61
elementary math（基本数学函数），61
hyperbolic（双曲函数），64
library（函数库），61
macro（宏函数），196
parameter（函数参数），61
programmer-defined（自定义函数），152
prototype（函数原型），157
recursive（函数递归），199
string（字符串函数），315
trigonometric（三角函数），62

G

Garbage value（垃圾值），29
Gauss elimination（高斯消元法），270，285
General form（一般形式），28
General structure of
C program（C 程序的一般结构），28
C++ program（C++ 程序的一般结构），374
Generations of computer languages（历代计算机语言），14，15
Genetic engineering（基因工程），7
examples（基因工程实例），77
get function（get 函数），378
getchar function（getchar 函数），66
Global（全局）
Positioning System (GPS)（全球定位系统），5，169，256
variable（全局变量），161
Glossary（术语表），446
GPS（全球定位系统），5，169，256
Graph character（图形特征），407
Graphics tool（图形化工具），13
Great circle（最大圆），169

H

Hand，recognition（手部识别），372，382
Hard copy（硬拷贝），11
Hardware（硬件），10
Head（头指针），353
Header file（头文件），27，235，407
assert.h，407
custom（自定义头文件），235
ctype.h，73，407
complex.h，397
errno.h，408
float.h，71，408
limits.h，71，408
locale.h，410
math.h，60，410
setjmp.h，411
signal.h，411
stat_lib.h，235
stdarg.h，411
stddef.h，411
stdio.h，26，43，411
stdlib.h，61，414
string.h，315，415
time.h，416
xy_coordinate.h，389
Helper function（辅助函数），390
Hexadecimal digit（十六进制数），67
High-level language（高级语言），13

Higher-dimensional arrays（多维数组），277
Hurricane（飓风），221，369
Hyperbolic function（双曲函数），64
Hyperplane（超平面），267

I

I/O diagram（I/O 图表），16
Iceberg（冰山），169
Identifier（标识符），29
if statement（if 语句），90
if/else statement（if/else 语句），92
ifstream class（ifstream 类），378
Ill conditioned（病条件），272
include statement（include 语句），26
Increment operator（自增运算符），41
Incremental search（增量搜索），188
Indirection operator（间接运算符），292
Infinite loop（无限循环），102
Information hiding（信息隐藏），390
Inheritance（继承），374
Initial value（初始值），27
Inner product（内积），260
Input statement（输入语句），47
Input/Output（IO，输入/输出），16
Insertion operator（插入运算符），375
Instrument reliability（仪器可靠性），180
int data type（int 数据类型），31，32
Integer（整数），31
  declaration（整数类型声明），32
  limits（整数类型数值范围），32
Interdisciplinary team（跨学科团队），9
Interference pattern（干涉模式），109
Internet（互联网），10
Inverse（反函数）
  hyperbolic functions（反双曲函数），65
  trigonometric functions（反三角函数），62
Invoke（调用），152
Ionosphere（电离层），132
IOS::fixed，376，377
IOS:left，377
IOS::right，377
IOS::scientific，377
IOS::showinput，376，377
iostream.h header file（iostream.h 头文件），363
Iris recognition（虹膜识别），148，163
isalnum function（isalnum 函数），66
isalpha function（isalpha 函数），66
iscntrl function（iscntrl 函数），67
isdigit function（isdigit 函数），66
isgraph function（isgraph 函数），67
islower function（islower 函数），66
isprint function（isprint 函数），67
ispunct function（ispunct 函数），67
isspace function（isspace 函数），67
istream class（istream 类），375
isupper function（isupper 函数），66
isxdigit function（isxdigit 函数），67
Iteration（迭代），107

J

Java，13，14
Jumbo jet（喷气式飞机），6

K

Kernel（内核），12
Keyword（关键字），29，30
  input（关键字输入），73

L

La Nin-a（拉尼娜现象），303
Language（语言），13
  assembly（汇编语言），13
  binary（二进制语言），13
  computer（计算机语言），13
  high-level（高级语言），13
  generation（历代编程语言），13
  natural（自然语言），14
  machine（机器语言），13
Lasers（激光），8
Last-in-first-out（LIFO，后进先出），363
Latitude（纬度），169
Least（最小）
  common multiple（最小公倍数），110
  squares（最小二乘法），129
Left justified（左对齐），45

LIFO，363
Library function（库函数），152
limits.h standard header file（limits.h标准头文件），409
Linear（线性）
  equation（线性方程），129
  interpolation（线性插值），52，53，282
  modeling（线性建模），128
  regression（线性回归），128
  simultaneous equations（联立线性方程组），265
Linked list（链表），353
Linking loading（链接加载），14，15
Local variable（局部变量），161
locale.h standard header file（locale.h标准头文件），410
log function（log函数），61
log10 function（log10函数），61
Logarithm（对数），61，78
Logic error（逻辑错误），14
Logical expression（逻辑表达式），88
Logical operator（逻辑运算符），89
  and（逻辑运算符与），89
  or（逻辑运算符或），89
  precedence（逻辑运算符优先级），89
long
  data type（long数据类型），31
  limits（long类型数据范围），32
Long-time power（长时功率），309
long double data type（long double数据类型），31
Longitude（经度），169
Loop（循环），84
  control variable（循环控制变量），105
  structures（循环结构），101
Lowercase（小写），67

M

Machine language（机器语言），13
Macro（宏），196
main function（main函数），27，149
Magnitude（振幅），236
malloc function（malloc函数），321
Mantissa（尾数），30
Manufacturing engineering examples（制造工程示例），145，180，207
Mathematical function（数学函数），60
math.h standard header file（math.h标准头文件），26，60，410
Mathematical tool（数学工具），13
MATLAB（MATLAB软件），13，421
Matrix（矩阵），260
  addition（矩阵的加法），263
  multiplication（矩阵的乘法），263
  square（矩阵的乘方），260
  subtraction（矩阵的减法），263
  transposition（矩阵的转置），262
Maximum（最大值），232
Mean（平均值），232
Mechanical engineering examples（机械工程示例），67，190，256
Median（中位值），232
Member function（成员函数），374
Memory（内存），11
  address（内存地址），289
  allocation（内存分配）11，321
  RAM（随机存储器），11
  ROM（只读存储机），11
  snapshot（内存快照），29
Mesosphere（中间层），132
Microprocessor（微处理器），4，11
Minimum（最小值），232
Minor（较小者），286
Mixed operation（混合运算），37
Modularity（模块化），150
Module（模块），149
  chart（图表），151
Modulus（系数），37
  operator（运算符），37
Molecular weight（分子量），226
Moon landing（登月行动），4
Motion control character（移动控制字符），407
Multiple assignment（多重赋值），42
Multiplication（乘法）
  matrix（矩阵），263
  symbol（符号表示），37

N

National Academy of Engineering（美国国家工程院），3
Natural language（自然语言），14
Nested if/else statements（if/else 语句的嵌套），93
Network（网络），10
New line indicator（换行符），44
Node（结点），353
Noise signals（噪声信号），282
Nonsingular（非奇异），267
Normalization technique（规一化技术），287
NULL（空指针 NULL），117，295
Null
  character（空字符），117
  pointer（空指针），295
Numeric conversion（数值转换），36
  data type（数据类型），31
  integer（整数类型），31
  floating-point（浮点类型），31
Numerical techniques（数值方法），52，128，186，265，395

O

Object（对象），373
  declaration（声明），389
  initialization（初始化），390
  program（程序），14
Object-oriented programming（面向对象的编程方法），373
Ocean
  engineering examples（海洋工程示例），56，108，169，221，303，349，369，370
Offset（偏移量），299
ofstream class（ofstream 类），378
One-dimensional array（一维数组），213
open function（打开函数 open），378
open-rotor engine（开式转子发动机），67
Operating system（操作系统），12
Operator（运算符）
  abbreviated assignment（缩写赋值运算符），42
  address（地址运算符），48，290
  arithmetic（算术运算符），36
  binary（双目运算符），37
  cast（类型转换运算符），38
  comma（逗号运算符），107
  conditional（条件运算符），94
  decrement（递减运算符），41
  dereference（取值运算符），292
  increment（递增运算符），41
  indirection（间接运算符），292
  logical（逻辑运算符），89
  overload（运算符重载），394
  priority（运算符优先级），38，39，297
  relational（关系运算符），88
  sizeof（sizeof 运算符），321
  unary（单目运算符），37
Optical fiber（光纤），8
Ordered list（有序列表），245
ostream class（ostream 类），375
Output statements（输出语句），44
Overflow（溢出），41
Overload（重载），394
Ozone measurements（臭氧层测量），131

P

Parameter（参数），61
  actual（实际参数），158
  formal（形式参数），158
  list（参数列表），158
Parsing（解析），228
Permutations（排列组合），211
Personal computer（PC，个人计算机），11
Pivot value（枢轴值），325
Pivoting（主元消元法），272
  column（列主元消元法），272，286
  row（行主元消元法），272，285
Plotting data（图形绘制数据），421
Pointer（指针），292
  file（文件指针），117
  operator（指针运算符），341
  void（空指针），321
Polymorphism（多态性），374
Polynomial（多项式），186
  root（多项式的根），186

Postfix（后缀），41
pow function（指数运算函数 pow），61
Power（功率），236
  long-time（长时功率），309
  short-time（瞬时功率），309
Power plant data（电厂数据），283
Precedence（优先级），38
  tables（优先级表），39，43，90，219，297
Precision（精确度），45
precision function（precision 函数），376
Prefix（前缀），41
Preprocessor directive（预处理），26
Prime Meridian（本初子午线），169
printf function（printf 函数），27，44
Printing character（可打印字符），71
Priority of operators（运算符优先级），38，39
Private member（私有成员），390
Problem Solving Applied（解决应用问题），48，56，67，97，108，131，163，169，180，190，221，226，236，256，272，303，309，318，342，349，382，385
Problem-solving
  methodology（问题解决方法论），16
  process（问题解决进程），16
Processor（处理器），10
Program（程序），10
  compilation（编译），14
  execution（执行），14
  structure（结构），25
  walkthrough（程序走查），87
Programmer-defined（自定义）
  class（自定义类），373，389
  function（自定义函数），152
  structure（自定义结构），335
Programer-written function（自定义函数），152
Prompt（提示信息），48
Prototype（原型），157
Pseudocode（伪代码），82
Pseudo-random number（伪随机数），175
Public member（公有函数），390
Punctuation character（标点字符），407
putchar function（putchar 函数），65

Q

Quadratic equation（二次方程式），399
Queue（队列），363
Quicksort algorithm（快速排序算法），325

R

RAM（随机存储器），11
rand function（rand 函数），175
RAND_MAX（RAND_MAX 常量），179
Random access memory（RAM，随机存储器），11
Random number（随机数），175
seed（随机数种子），176
Range（范围），32
Read-only memory（ROM，只读存储器），11
Real roots（实根），186
Real-time program（实时程序），13
realloc function（realloc 函数），321
Rectangular coordinates（直角坐标），170
Recursion（递归），199
Recursive function（递归函数），199
Reference（引用）
  by address（地址引用），220
  by value（值引用），220
Refinement（提炼），82
  in flowchart（提炼后的流程图），82
  in pseudocode（提炼后的伪代码），82
Register storage class（寄存器存储类型），162
Relational（关系运算）
  operator（关系运算符），88
  precedence（关系运算优先级），89
Reliability（可靠性），180，207
Repetition structure（重复结构），82，84
return statement（return 语句），28
Reusability（可重用性），150
Richter scale（里氏震级），309
Right justified（右对齐），45
ROM（只读存储器），11
Root（根），186，209
  polynomial（多项式的根），186
Root-finding technique（求根公式），188
Rounding（取整），61

Row vector（行向量），260
Run-time error（运行时错误），14

S

Saffir-Simpson scale（萨菲尔辛普森飓风等级），221
Salinity（盐度），56
Satellites（卫星），5
scanf function（scanf 函数），47
Scientific notation（科学计数法），30
Scope（范围），160
  resolution operator（范围限定运算符），390
screen output（屏幕输出），47
Sea（海洋）
  state（海况），108，109
  surface temperature（海面温度），303
Seawater（海水）
  composition（海水成分），56
  freezing temperature（海水冰结温度），56
Search algorithm（查找算法），244
  binary（二分查找），246
  sequential（顺序查找），244
Seismic event（地震事件），309
  detection（地震事件探测），309
Seismometer（地震仪），309
Selection（选择）
  sort（选择排序），242
  statements（选择语句），90
  structure（选择结构），82，83
Sentinel（哨兵）
  controlled loop（哨兵控制回路），119
  signal（哨兵标记），119
Sequence（序列）
  escape（转义序列），46
  structure（顺序结构），82
Sequential search（顺序查找），244
setf function（setf 函数），376
setjmp.h standard header file（setjmp.h 标准头文件），411
short
  data type（short 数据类型），31，32
  limits（short 类型数值范围），32
Short-time power（短时功率），309
signal.h standard header file（signal.h 标准头文件），411
Simulation（仿真），181，206，208
Simultaneous linear equations（联立线性方程组），265
  graphical interpretation（线性方程组的图像阐释），265
sin function（sin 函数），62
sinh function（sinh 函数），64
Sinusoid（正弦曲线），109
sizeof operator（sizeof 运算符），321
Smartphone（智能手机），11
Societal context（社会环境），10
Soft copy（软拷贝），11
Software（软件），10，11
  life cycle（软件生命周期），15
  maintenance（软件维护），15
  prototype（软件原型），15
  tool（软件工具），12
Solutions（参考答案）
  End-of-Chapter problems（章末问题的参考答案），438，442
  Modify! problems（"修改"习题参考答案），436
  Practice! problems（"练习"习题参考答案），424
  Short Answer problems（"简述题"参考答案），438
  Programming problems（"编程题"参考答案），442
Sorting algorithm（排序算法），242
Sounding rocket（探空火箭），143
Source program（源程序），14
Speech analysis（语音分析），212
Spherical coordinates（球面坐标），170
Spreadsheet（电子表格），12
srand function（srand 函数），176
sqrt function（sqrt 函数），61
Square matrix（方阵），260
Stable system（稳定系统），190
Stack（栈），363
Standard（标准）
  C library（标准 C 语言库），26，407
  deviation（标准差），233
  I/O（标准输入/输出），26
Statement（语句），27
Static Storage class（静态存储类型），162
Statistical measurements（统计测量），231
stdarg.h standard header file（stdarg.h 标准

头文件)，411
stddef.h standard header file (stddef.h标准头文件)，411
stdio.h standard header file (stdio.h标准头文件)，26，43，411
stdlib.h standard header file (stdlib.h标准头文件)，61，414
Stepwise refinement (逐步提炼)，82
Storage class (存储类型)，161
  automatic (自动存储类型)，161
  external (外部存储类型)，161
  register (寄存器存储类型)，162
  static (静态存储类型)，162
strcat function (strcat函数)，316
strchr function (strchr函数)，316
strcmp function (strcmp函数)，316
strcpy function (strcpy函数)，315
strcspn function (strcspn函数)，316
Stream (流)，375
Stratosphere (平流层)，132
String (串)，314
string.h standard header file (string.h标准头文件)，415
strlen function (strlen函数)，315
strncat function (strncat函数)，316
strncmp function (strncmp函数)，316
strncpy function (strncpy函数)，315
strpbrk function (strpbrk函数)，316
strrchr function (strrchr函数)，316
strspn function (strspn函数)，316
strstr function (strstr函数)，316
struct statement (struct语句)，336
Structure (结构)，335
  chart (结构图)，151
  declaration (结构体声明)，336
  member operator (访问结构成员的运算符)，336
Structured program (结构化程序)，82
Subscript (下标)，214
Subtraction (减法)
  matrix (矩阵减法)，263
  symbol (减法符号)，37
Summation notation (求和符号)，129
Surface winds (地表风)，385
Suture packaging (缝合线包装)，144
switch statement (switch语句)，95
Symbolic constant (符号常量)，34
Syntax (语法)，13
Synthesis (合成)，10
System (系统)，190
  dependent (系统相关)，32
  of equations (方程组)，267
  limitations (系统约束)，71

T

Tag (标签)，336
tan function (tan函数)，62
tanh function (tanh函数)，64
Temperature (温度)
  conversion (温度转换)，76
  distribution (温度分布)，285
Terrain navigation (地形导航)，256
Test data (测试数据)，87
Thermosphere (电离层)，132
Three-dimensional array (三维数组)，277
Threshold (阈值)，309
Timber management (木材管理)，144
time.h standard header file (time.h标准头文件)，416
tolower function (tolower函数)，66
Top-down design (自顶向下设计)，81
toupper function (toupper函数)，66
Trailer signal (尾标记)，119
Trajectory (弹道)，143
Transpose ((矩阵的)转置)，262
Trigonometric function (三角函数)，62
Troposphere (对流层)，131
Truncate (对流层)，37
Tsunami (海啸)，349，370
Two-dimensional array (二维数组)，248，300
Type specifier (类型说明符)，32

U

Unary operator (单目运算符)，37
Underflow (下溢)，41
Uniform random number (均匀分布随机数)，175

Unit conversion（单位换算），77，142
Unmanned aerial vehicle（UAV，无人机），256
Unordered list（无序列表），244
unsigned qualifier（无符号限定符 unsigned），32
Uppercase（大写），67
Using directive（使用（命名空间）指令），375
Utility（实用程序），12
Utterance（语音），236

V

Validation（验证），87
Variable（变量），29
  automatic（自动变量），161
  external（外部变量），161
  global（全局变量），161
  local（局部变量），161
  scope（变量范围），160
  static（静态变量），162
Variance（方差），233
Velocity computation（速度计算），67
Vector（向量），260
  column（列向量），260
  row（行向量），260
Verification（确认），87
void（空）
  data type（空数据类型），27
  function（返回 void（空）的函数），27，157
  pointer（void（空）类型的指针），321
Volumes（体积），77

W

Wave characteristics（波的特性），108
Wavelength（波长），108
Weather balloon（探空气球），145
while loop（while 循环），102
White space（空白字符），67，377
Wind（风）
  direction（风向），385
  tunnel（风洞），78
Word processor（文字处理软件），12
World marketplace（全球市场），10

Z

Zero crossings（过零点），236

推荐阅读

深入理解计算机系统（原书第3版）
作者：兰德尔E.布莱恩特 大卫R.奥哈拉伦
译者：龚奕利 贺莲
中文版：978-7-111-54493-7，139.00元

计算机系统概论（第2版）
作者：Yale N. Patt Sanjay J. Patel
译者：梁阿磊 吴贵忠 林俊
中文版：7-111-21556-1，49.00元
英文版：7-111-19766-6，66.00元

数字设计和计算机体系结构（第2版）
作者：David Harris Sarah Harris
译者：陆俭峰
英文版：978-7-111-44810-5，129.00元
中文版：2016年4月出版

计算机系统：核心概念及软硬件实现（原书第4版）
作者：J. Stanley Warford
译者：龚奕利
书号：978-7-111-50783-3
定价：79.00元

推荐阅读

C++语言程序设计三部曲

C++语言之父经典名著最新版本，全面掌握标准C++11及其编程技术的权威指南

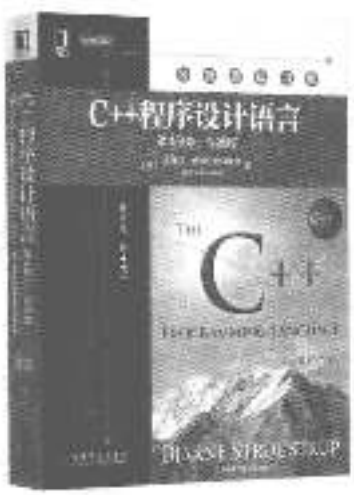
系统学习C++语言的最佳读物

C++语言导学

作者：本贾尼·斯特劳斯特卢普 译者：杨巨峰 等
ISBN：978-7-111-40812-4 定价：39.00元

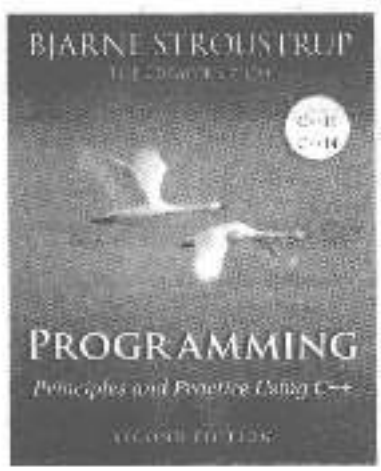
C++程序设计语言（第1~3部分）（英文版·第4版）

作者：本贾尼·斯特劳斯特卢普
ISBN：978-7-111-52368-4 定价：163.00元



C++程序设计语言（第4部分：标准库）（英文版·第4版）

作者：本贾尼·斯特劳斯特卢普
ISBN：978-7-111-52487-4 定价：89.00元



C++程序设计原理与实践（第2版）

中文版即将全新出版

推荐阅读

算法导论（原书第3版）

作者：Thomas H Cormen 等 ISBN：978-7-111-40701-0 定价：128.00元

算法基础：打开算法之门

作者：Thomas H. Cormen ISBN：978-7-111-62306-4 定价：59.00元

数据结构与算法设计

作者：三晓东 ISBN：978-7-111-37924-9 定价：29.00元

数据结构、算法与应用——C++语言描述（原书第2版）

作者：Sartaj Sahni ISBN：978-7-111-49600-7 定价：79.00元

数据结构与算法分析——C语言描述（原书第2版）

作者：Mark Allen Weiss ISBN：978-7-111-12748-X 定价：35.00元

数据结构与算法：Python语言描述

作者：裴宗燕 ISBN：978-7-111-62119-1 定价：45.00元

工程问题C语言求解 原书第4版

Engineering Problem Solving with C Fo

本书与一般C语言编程书籍最大的不同在于以工程问题为引导培养程序设计思维，跟随作者的脚步，你时而化身一名聪明的探员，思考如何解决犯罪现场调查中的指纹识别问题；时而成为一名无人机设计师，研究处理各种地面形态和拓扑结构的导航算法；时而扮演一名航空科学家，收集探测火箭的轨迹数据并分析性能……如果你毫无编程经验，书中详尽的C语言基础知识将带你轻松入门；如果你面临的是实际工程应用，书中经过实践验证的方法论将开拓你对计算思维的理解。

本书特点

- 有趣的工程问题。本书以“犯罪现场调查”为主题案例，其他精选的工程问题包括冰山追踪、仪器可靠性、病情分析及氨基酸分子量等，包罗万象，妙趣横生。
- 经典的解决方案。“五步法”涵盖从问题陈述到测试的全过程，此外，书中还使用了分解提纲、伪代码和流程图来完成自顶向下的程序设计和算法求精。
- 丰富的习题资源。扩展了与应用问题相关的“修改”题，章后“简述题”可帮助读者巩固知识，“编程题”则提供了动手操练的机会，且书后配有答案。

作者简介

德洛莉丝 M. 埃特尔（Delores M. Etter）以编写解决工程问题和科学问题的创新教材而得到广泛赞誉。目前任教于美国南卫理公会大学电子工程系和计算机科学系，并任该校工程教育Caruth研究所执行总监以及工程教育德州仪器杰出主席。她曾先后在科罗拉多大学博尔德分校、新墨西哥大学电气和计算机工程学院任教，也曾任斯坦福大学客座教授。埃特尔博士是IEEE会士，著有《工程问题C++语言求解》等多部教材。

Pearson
www.pearson.com

投稿热线：(010) 88379604
客服热线：(010) 88378991 88395066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上书店：www.china-pub.com
数字服务：www.hzmedia.com.cn

上架指导：计算机/程序设计
ISBN 978-7-111-55441-7
9 787111 554417 >
定价：79.00元