计 算 机 科 学 丛 书

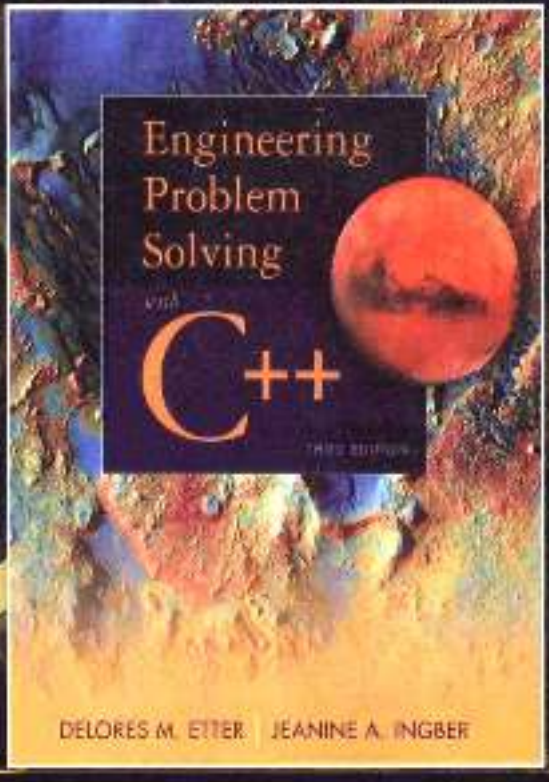
PEARSON

原书第3版

工程问题C++语言求解

（美）Delores M. Etter Jeanine A. Ingber 著
冯力 周凯 译

Engineering Problem Solving with C++
Third Edition



机械工业出版社
China Machine Press



工程问题C++语言求解 原书第3版

Engineering Problem Solving with C++ Third Edition

本书是以工程问题求解和C++编程语言知识结构相互融汇讲解的经典之作，书中利用已经过作者证明的求解工程问题的五步法，展现了大量来自工程、科学和计算机科学领域的不同示例，包括物体的速率、海水冰点、气象气球、臭氧测量、仪器可靠性、语音信号分析、飓风等级分析、海啸预警、地形导航以及电路分析等。

本书特点

- 真实世界的工程、科学示例和应用问题。
- 求解工程问题的五步法：
  1）清楚地描述问题。
  2）描述输入和输出信息，确定需要的数据类型。
  3）手动运行一个简单的例子。
  4）设计算法，并将它转换成计算机程序。
  5）使用大量数据测试解决方案。
- 类型丰富的练习题：节后的练习，与示例程序和“解决应用问题”节中的程序有关的“修改”问题，每章后的习题（包括判断题、语法题、多选题、编程题等）。

作者简介

Delores M. Etter 以解决工程和科学问题方面的创新教材得到广泛认可，目前是美国南卫理公会大学达拉斯分校工程教育学院德州仪器杰出主席。她曾先后在美国海军学院、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，也曾是斯坦福大学客座教授。Etter博士是美国国家工程院院士，IEEE、AAAS、ASEE会士，还曾是国家科学委员会成员。

Jeanine A. Ingber ASAP有限责任公司首席技术官，ASAP成立于2009年，主要研究工程和物理应用问题的数值解。她任教于美国艾奥瓦州立大学和新墨西哥大学，获得过多项教学奖。

PEARSON

PEARSON
www.pearson.com

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn



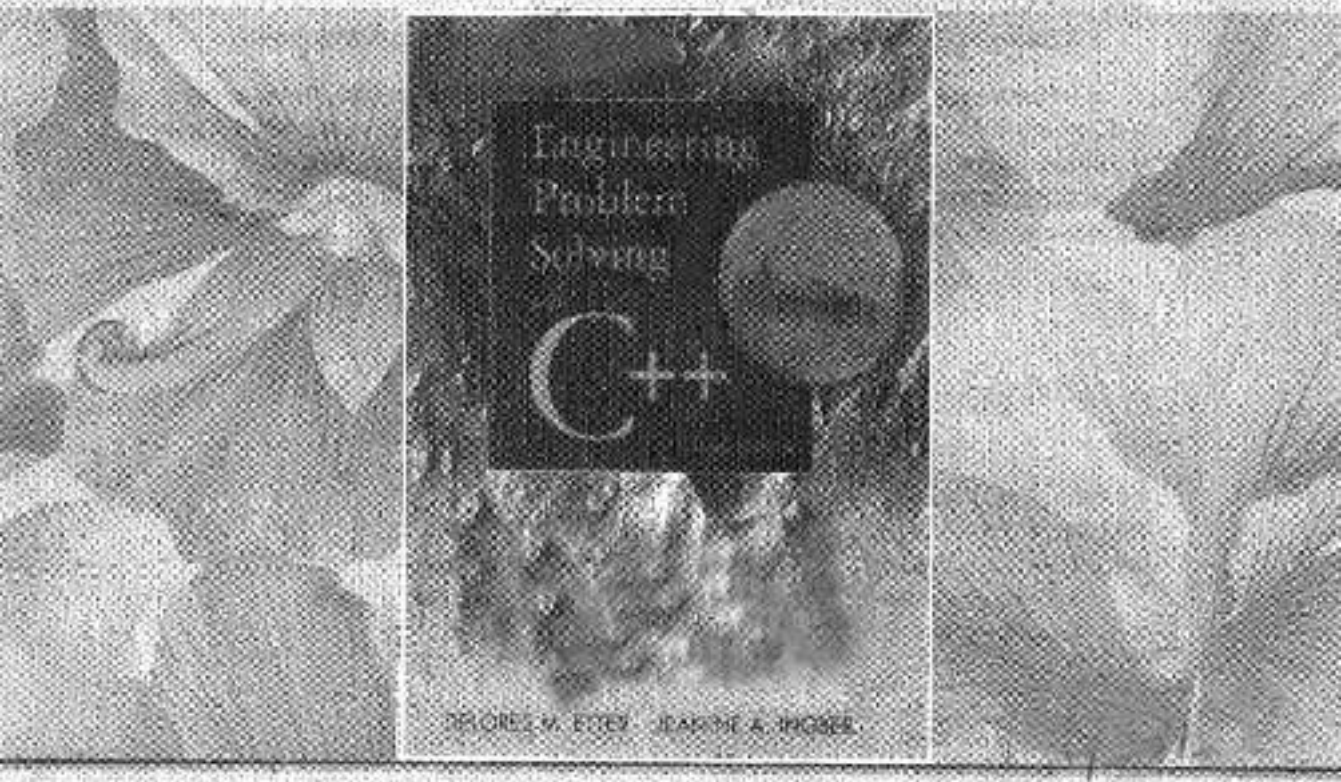
计 算 机 科 学 丛 书

原书第3版

工程问题C++语言求解

（美）Delores M. Etter　Jeanine A. Ingber 著
冯力　周凯 译

Engineering Problem Solving with C++
Third 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 3 版英美）埃特（Etter, D. M.），（美）恩格贝尔（Ingber, J. A.）著；冯力，周凯强．—北京：机械工业出版社，2014.4
（计算机科学丛书）
书名原文：Engineering Problem Solving with C++，Third Edition

ISBN 978-7-111-45907-1

Ⅰ.工… Ⅱ.①埃… ②因… ③冯… ④周… Ⅲ.C 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2014）第 030972 号

本书版权登记号：图字：01-2012-4655

Authorized translation from the English language edition, entitled *Engineering Problem Solving with C++, Third Edition*, 9780132492652, published by Pearson Education, Inc., Copyright © 2012.

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 © 2014.

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（不包括中国台湾地区和中国香港、澳门特别行政区）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。

本书介绍如何利用 ANSI C++ 编程语言以基于对象的编程方式来解决工程问题。书中从通用的工程问题解决方法论入手，以众多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、标准输入和输出、基本函数、控制结构、数据文件、模块化编程、数组以及指针等重要概念。

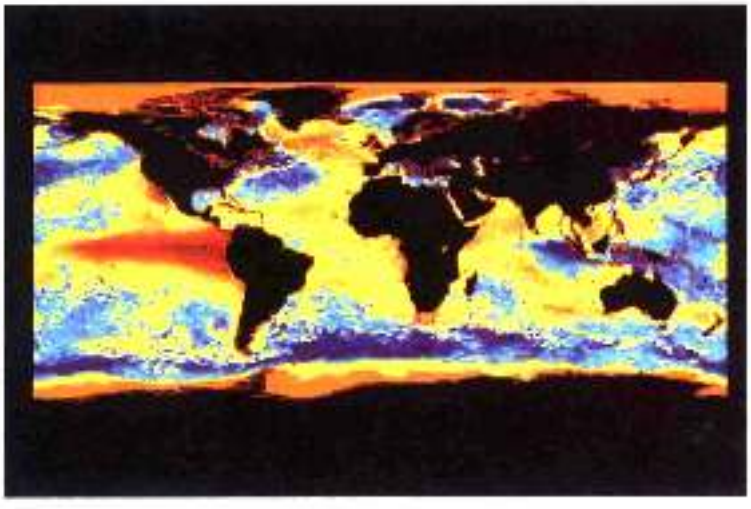
本书实例内容翔实、景脉所讲实职点，实战性强，可作为高等院校工程和科学计算相关专业的教材或数学参考书，也可作为初学者建立 C++ 编程实职与实际工程应用之间连接的桥梁。

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：赵 荷 迟振春　　责任校对：殷 虹
印 刷：北京市荣盛彩色印刷有限公司　　版 次：2014 年 8 月第 1 版第 1 次印刷
开 本：185mm×260mm 1/16　　印 张：29（含 0.25 印张彩插）
书 号：ISBN 978-7-111-45907-1　　定 价：79.00 元

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

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 韩大 / 张晓东

天气、气候和全球变化的预测



由NOAA（美国国家海洋和大气局）和环境卫星服务提供

◀为了预测天气、气候和全球变化，我们必须了解大气和海洋之间复杂的相互作用。这些相互作用受到很多因素的影响，包括温度、风、洋流、沉淀、土壤湿度、积雪、冰川、极地海冰和地球大气中其氧和发外线辐射的吸收。由超高分辨率辐射计（AVHRR）拍摄的地球卫星图像展现了东太平洋的一次厄尔尼诺现象。厄尔尼诺现象是海洋-大气系统的一次变动，它会影响全球的气候。图像中采用不同的颜色区分正常温度区域和厄尔尼诺现象区域。颜色从紫色（远低于正常温度）到蓝色，再到黄色，最后到红色（远高于正常温度）。陆地区域用图气标记，区域能够用使用红色标记。红色区域的厄尔尼诺正穿过太平洋，沿着赤道向东移动。



由Jeff MacDonald/Aunty

▲图中站在阿克萨哥洲奥斯汀的一家电视台内的气象部，气象学家正在查看显示器上显示的天气预报。

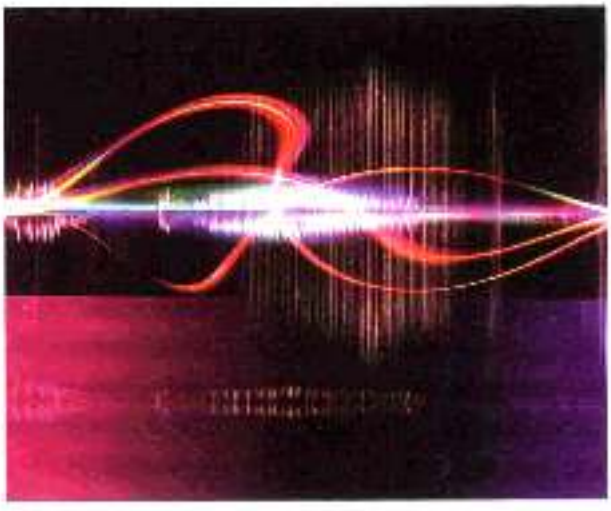


由The Stock Connection的Mark和Audrey Gibson提供

◀这是一幅北半球气象图的计算机图像展示，它使用了可视化技术来展示来自卫星的气象数据。

计算机语音识别和声音识别

计算机语音识别是将一段语音信号转换成一个单词序列的过程。在移动电话的语音拨号和自动应答系统中，语音识别已经得到了成功的应用。计算机语音识别则是辨别谁在说话，而不是分辨说了什么。声音识别关注讲话的声学特征，这些声学特征反映了讲话者的嘴和传统的物理尺寸和形状，以及口语的模式，如音色和音高。声学信号可以转换成电信号，这些电信号可以在计算机上进行可视化和分析。最后生成若干声音图形，声音图形在这个图中是锯齿状向包波形。▶

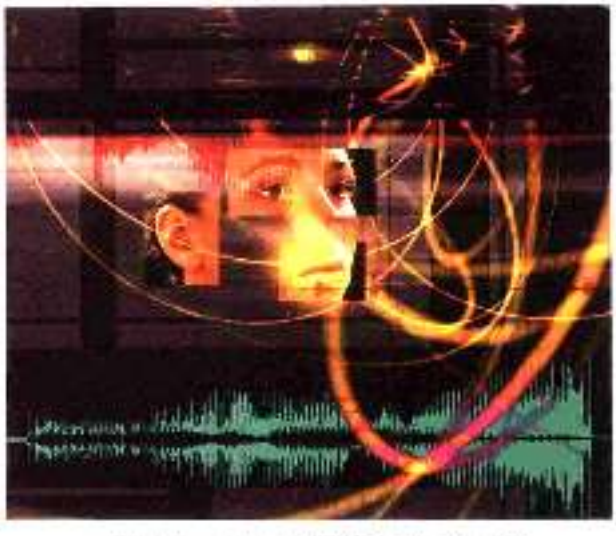


由 Photo Researchers 公司的 Mehau Kulyk 提供



由 Photo Researchers 公司的 Mehau Kulyk 提供

◀声音识别属于行为生物识别的范畴。生物识别研究通过各个人的有物理特征和行为特性来辨别个体。图中以人的体形解为背景的电路图，说明了将声音图形转换成某个个体声音的唯一数字化表示对应的复杂性。



由 Photo Researchers 公司的 Mehau Kulyk 提供

这幅计算机产生的图像中有含了声音、嘴和多个声音图形，这代表了声音识别和语音合成所需要的技术。▶

太空探索

1969年7月21日人类登陆月球，这也许是美国历史上最复杂、最有挑战的工程项目了。阿波罗11号在1969年7月16日发射，它是首个载人登月项目。7月21日，阿波罗11号宇航员Neil A. Armstrong在月球上跨出了他的第一步。他留在月球表面的脚印被拍了下来。▶



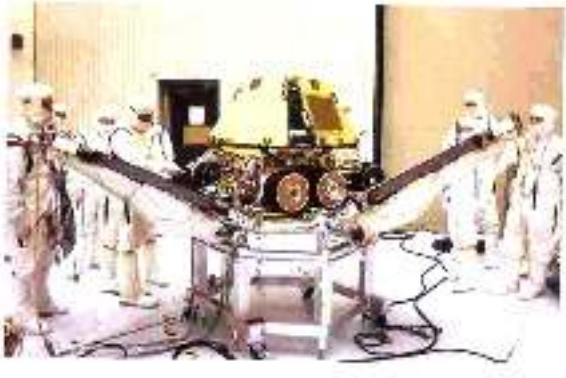
◇NASA和Photo Researchers公司提供



◇NASA和喷气推进实验室提供

◀太空探索促成了近年来科学数据收集方面的工程成就。漫步者火星探测器设计用于在行星表面行走，并对土壤和岩石进行了详细分析。漫步者火星探测器的任务是NASA火星探测项目的一部分，该项目是一个长期的火星机器人探测项目。在该任务诸多的科学目标中，基本的一项就是搜寻火星周围的土壤和岩石并记录它们的特征。在土壤和岩石中包含了火星上水活动的线索。宇宙飞船的降落目的地在火星的背面，因为在背面似乎有液态水出现的痕迹。

在美国出罗里达州的哈·肯尼迪航天中心工业区主人的西南侧的Payload Hazardous Servicing Facility中，技术人员重新打开漫步者火星探测器2号的着陆器外壳，以便操作飞船中的一块电路板。▶



由NASA和喷气推进实验室提供

计算机仿真

当物理实验不可行时，计算机仿真作为第三种科学范式，推动了知识的发展。如同应用于高级复合材料设计中一样，计算机仿真的应用是一件令人兴奋的事情，许多科学和工程领域受益于此，包括制造业、结构力学、材料科学和医学。磁泡塑料在三种应用中有特定的应用领域，如图中所示的自愈塑料。红色的圆圈是从自愈塑料中释放出来的胶囊。破坏塑料会让胶囊破裂，导致液体释放从而修复破损。这种塑料由伊利诺伊大学的团队设计，并用于宇宙飞船的材料设计和外科移植中。▶



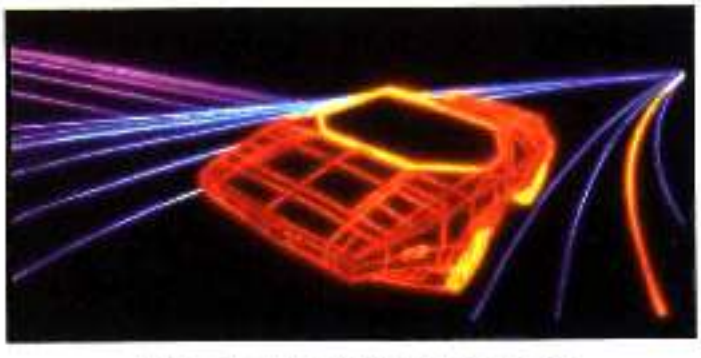
由 Scott White、UHLC（伊利诺伊大学香槟分校）和 Photo Researchers 公司提供



由 Frank Witzeman 提供

◀计算机仿真对于各种器械设计和性能测试起了很大作用。仿真可以预测在一定条件下车辆的响应情况，而不需要冒着破坏车辆的风险。图中是模拟一架 SR-71 侦察机在遭遇开度角攻击时的冲击波仿真图形。

图中所示为车辆设计中使用计算机仿真得到的关于气动跑车的线框模型图。▶



由 Photo Researchers 公司的 Ramon Santos 提供

出版者的话

文艺复兴以降，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坚实的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的过程中，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，不仅奠定了研究的基石，还树立了学术的源泉，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，也是与国际接轨、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，Brain 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襄助，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

华章教育

北京科技图书出版中心

译 者 序

Engineering Problem Solving with C++, 3e

人类对于语言的最初使用，是猿类进化到人类的重要标志，也是信息技术的诞生之源。而文字以及印刷技术的出现和使用，第一次突破了时空的约束而传递着更为复杂、容量更大的信息。"鸿雁传书"是中国文化中记录较多的信息传递方式，如晚唐诗人杜牧在《七律·寄远》中最早写道："碧云空断雁行处，红叶已凋人未来。塞外音书无信息，道傍车马起尘埃。"而直到20世纪60年代，计算机的普及与应用才彻底激发了人类充分利用信息的巨大潜力。无论身处哪个时代，语言始终都是描述、传递和处理信息的载体和有效工具，在当前的网络信息时代，计算机编程语言更是超越延续了几万年的语言工具，在这个技术时代留下深深的烙印。

然而面对浩如烟海的计算机软件编程书籍，每个初入门的读者都迫不及待地想弄清楚："究竟哪种编程语言最优秀？""哪种编程语言用得更多？""哪种编程语言能最快入门？"这些问题从学术上、理论上、工程应用上都有很多解释，而且由于每个程序员从事的专业领域、技术水平或者掌握程度的不一样，也会有不同的回答，因此更多的是"仁者见仁，智者见智"，或者"如人饮水，冷暖自知"。

至于学习编程，专一与执着尤为重要。曾国藩在其家书中说"掘井多而皆不及泉"，意思说挖了很多井，但没有一口井里挖出泉水。这对于那些在多种编程语言中徘徊不定的年轻人来说，应该有足够的启发了：为何不扎扎实实掌握一门编程语言，真正尝到泉水的甘甜清冽呢？难道大家从这之中还不能体会到专注的重要吗？

本书是以工程问题求解和C++编程语言知识结构相互融汇讲解的经典之作，由美国专家Delores M. Etter和Jeanine A. Ingber共同编著，从通用的工程问题解决方法论入手，以物体的速率、海水冰点、气象气球、臭氧测量、仪器可靠性、语音信号分析、飓风等级分析、海啸预警、地形导航以及电路分析等众多工程问题为应用对象，将C++语言中的基本操作符、标准输入和输出、基本函数、控制结构、数据文件、模块化编程、数组以及指针等重要概念娓娓道来，使得学习C++的各类知识点变得更加生动、有趣，更重要的是整个过程充满了解决工程问题需要的丰富而充满自信的经验，能够让初学者更快地建立C++编程知识与实际工程应用的连接，这样在读者的脑海中对编程知识点的印象与理解就更加深入透彻和胸有成竹了。

学习编程语言终究离不开多练勤思。其实，大体上来看，学习C++编程语言不外乎掌握语言与操作工具，工具的操作虽然有平台和操作方式的区别，但在熟练掌握之后不必过于迷恋，在暂时不得要领而难以登堂入室之时也不必徘徊彷徨。要想具备熟练的语言技能，除了看书阅读外，还需要有目的性地进行小型工程项目开发，需要思考与设计。日积月累，终有一天你会由编程路漫漫中的"渐悟"走向登高一览的"顿悟"，发现编程学习中很多东西原来"不外乎如此""本质上就是……"，从而豁然开朗，步入到"触一类而通万象"的知识启发的"二摩地"。

衷心感谢我们多年的朋友程雄先生对本书文字及内容的认真审核，您在C++方面的精湛技能以及深厚的经验令我们印象深刻！最后，祝大家学习愉快！

译 者

2014年5月于武汉光谷

C++语言源自于C语言，它通过使用类和程序员自定义类型来支持面向对象编程的特性。C语言中那些适用于系统级操作和嵌入式编程的特性在C++中也得到了支持。这使C++语言成为最强大和最通用的编程语言之一。同时对于科学家和工程师而言，它也是计算导论课程的不错选择。本书主要介绍如何使用C++来求解工程问题。同时也介绍了C++语言面向对象的特性。我们的目标如下：

- 设计一种用于求解工程问题的通用方法论。
- 在着眼于编程和解决问题的基本层面的同时，阐述C++的面向对象特性。
- 通过大量的工程示例和应用，说明使用C++解决问题的过程。
- 为了使内容适合易懂，融合了对于数据类型、函数、在C++标准模板库中定义的容器类的介绍。

为了达到这些目标，第1章中介绍了本书其他章节求解工程问题时使用的五步处理过程。第2章介绍了C++支持的内建数据类型，同时介绍了类、自定义对象和支持标准输入/输出的成员函数。第3～6章介绍了C++解决工程问题的基本能力，包括控制结构、数据文件、函数和自定义数据类型。第7和第8章介绍了数组、向量和字符串类。第9章介绍了指针、动态内存分配和链式数据结构的用法。第10章对于一些高级主题进行了更深入的介绍，包括函数模板、类模板、递归成员函数、继承和虚函数。贯穿所有这些章节，我们使用了大量来自不同的工程、自然科学和计算机科学的示例。这些示例的解决方案都是使用五步处理过程和标准C++开发的。

第3版的特征

- 介绍了两种集成开发环境（IDE）：
  - NetBeans
  - MS Visual Studio
- 包含了使用全球定位系统（GPS）数据和海啸预警系统数据的新工程应用程序。
- 包括了按位操作符的介绍。
- 扩展了控制结构的覆盖面
- 为了灵活性考虑，在本书可选章节提早介绍了类和自定义数据类型的开发。
- 整合了贯穿全书的类的覆盖范围，提供了标准解决方案和面向对象解决方案的比较。
- 包括了附加的语句块、程序跟踪和内存快照以及流程图。

学生资源和教师资源中心可以在线访问www.pearsonhighered.com/etter。

先决条件

本书不假定读者之前具备计算机使用经验。对于数学的要求是大学代数和三角知识。当然，如果学生曾用过其他计算机语言或者软件工具，则可以更快地阅读前面的内容。

课程结构

本书中的内容可以作为工程和科学计算专业一学期课程的基础部分。这些章节包含数学计算、字符数据、控制结构、函数、数组、类和指针的基本主题。具有其他计算机语言背景的学生应该能够在半学期内掌握这些内容。仅介绍 C++ 的短学时课程只需使用本书的必修章节（可选章节的内容使用星号标识）。下面是使用本书的三种方式以及相关推荐章节：

- **介绍 C++** 许多基础类课程除了介绍程序语言外还包括若干计算机工具。对于这类课程，我们推荐使用第 1 ~ 8 章中的非可选章节。这些内容介绍了 C++ 的基本能力，通过学习学生将能够使用数学计算、字符数据、控制结构、自定义数据类型、函数和数组写出内容完实的程序。
- **使用 C++ 求解问题** 在半学期的课程中专门教授学生掌握 C++ 语言，我们推荐涵盖第 1 ~ 10 章中所有的非可选章节。这些内容覆盖了 C++ 语言中所有的基础概念，包括数学计算、字符数据、控制结构、函数、数组、类、模板和指针。
- **使用 C++ 和数值方法求解问题** 高年级学生或者已经熟悉了其他高级语言的学生可以较快地学习书中的内容。此外，他们可以将数值方法的相关内容应用到其他课程中。因此，我们推荐这些学生学习第 1 ~ 10 章中的所有章节，包括可选内容。

本书章节的设计在主题的顺序上为教师提供了较大的灵活性。自定义类型和类的相关内容自第 2 章开始贯穿本书。但是，有关类的内容都作为一个可选小节放在每章的结尾部分。下面的依赖关系图对此进行了说明。



解决问题的方法论

需要强调的是，工程和科学问题求解方法在本书中是一个完整的过程。第 1 章中介绍了使用计算机解决一些问题的五步处理过程：

1）清楚地描述问题。

2）描述输入和输出信息，确定需要的数据类型。

3）手动运行一个简单的例子。

4）设计算法，并将它转换成计算机程序。

5）使用大量数据测试解决方案。

为了不断强化求解问题的能力，这五步中的每一步在每次解决完一个完整的工程问题时都要清楚地标识出来。此外，使用分解提纲、伪代码和流程图完成自顶向下的设计并逐步细化。

工程和科学应用

本书的重点放在将真实世界的工程、科学示例和问题相结合上。这个重点以各类工程挑战为中心，这些挑战包括：

- 天气、气候和全球变化的预测
- 计算机语音识别
- 图像处理
- 人工智能
- 提高油气采集率
- 仿真

每一章都以有关某个工程挑战的讨论开始，其中给出了工程师可能感兴趣的地方。每章的后面，我们不仅解决了开头所引出的问题，还将解决方案应用于其他的问题中。

标准 C++

书中所有的语句和程序都是使用符合国际标准组织和美国国家标准学会（ISO/ANSI）C++ 标准委员会发布的 C++ 标准编写的。ISO 和 ANSI 共同发布了 C++ 编程语言的第一个国际标准。通过使用标准 C++，学生可以学习编写可移植的代码，这些代码可以从一种计算机平台移植到另一种计算机平台上。本书中讨论了许多 C++ 编程语言的标准功能，同时在附录 A 中还讨论了 C++ 标准库中的附加组件。

软件工程的观点

工程师和科学家都希望设计并实现对用户友好且可重用的计算机解决方案，因此了解软件工程技术是很关键的。在程序的设计中需要强调可读性和文档。有关软件工程的主题在本书各处都有讨论，其中包括软件生命周期、可移植性、维护、模块化、递归、抽象、可重用性、结构化编程、确认和验证。

类型丰富的练习题

学习任何新的技能都需要进行大量不同难度层次的练习。本书中设计了多种类型的练习题，用于训练学生解决问题的能力。第一种类型是练习，这是答案较短的问题，与该节所讨论的内容相关。大部分节后面都带有一组练习，这样学生可以确定他们是否做好了继续学习下一节的准备。本书末尾给出了完整的练习答案。

本书设计了“修改”类型的问题几乎进行动手练习，一般与示例程序和“解决应用问题”节中的程序有关。在这些节中，我们使用五步处理过程开发一个完整的C++程序。“修改”类型的问题要求学生使用不同的数据集来运行程序，以测试他们对程序运行和工程变量之间关系的理解。这些练习题要求学生对程序进行简单的修改，然后运行程序对他们的修改进行测试。

每章都以习题结束，其中包括判断题、讲法题、多选题等，还包括一组编程题。大部分习题是与本章所介绍内容相关的、答案较短的问题，这些问题帮助学生确定他们是否很好地理解了本章所介绍的C++特性。编程题是与各种工程应用相关的新问题，难度不同，可能非常直接地看出解决办法（易），也可能需要较长的工程作业（难）。每个编程题都要求学生开发一个完整的C++程序或函数。

可选的数值方法

数值方法在解决工程问题时得到了广泛的应用，本书在可选章节中对数值方法进行了讨论，包括插值、线性建模（回归）、求根、数值积分和解联立方程。书中还介绍了矩阵的概念，并使用大量的例子进行了说明。所有这些主题都假定读者具有代数和三角知识背景。

附录

为了进一步方便读者参考，附录中包含了许多重要的主题。附录A包含了对于C++标准库的讨论。附录B给出了ASCII字符编码。附录C中介绍了MATLAB。附录D给出了练习答案。附录E包含了本书中用到的参考文献。

其他资源

所有教师和学生资源都可以访问网站www.pearsonhighered.com/etter得到。在这里，学生可以得到本书的所有源代码，教师还可以在教师资源中心注册。教师资源中心包含本书使用的全部示例程序、所有编程问题的完整解决方案、测试题库，以及应用问题中用到的数据文件和完整的课程讲座幻灯片。

致谢

感谢杰出的评审团队——科罗拉多矿业大学的Roman Tankelevich，克利州立大学的John Sustersic，波特兰大学的Tanya L. Crenshaw，纽约城市学院的Daniel McCracken，弗吉尼亚理工学院的Deborah L. Pollio，科罗拉多矿业大学的Keith Hellman，波特兰大学的Tammy VanDeGrift，塞州州立大学Behrend校区的Melanie Ford，加州理工大学的Amar Raheja，他们为本书提出了详尽和有建设性的建议，感谢他们颇具价值的观点。还要向优秀的编辑人员表示感谢，他们是Tracy Dunkelberger、Stephanie Sellinger和Emma Snider，感谢他们把每件事都处理得井井有条。最后，还要感谢杰出的产品团队，包括Eric Arima，Kayla Smith-Tarbox和Lily Ferguson，感谢他们对每个细节的洞察与关注。

推荐阅读













系统分析与设计导论（英文版）

作者：Jeffrey A. Whitten等 ISBN：978-7-111-35275-3 定价：79.00元

数据结构与算法分析：Java语言描述（英文版·第3版）

作者：Mark Allen Weiss ISBN：978-7-111-41236-6 定价：79.00元

Python语言程序设计（英文版）

作者：Y. Daniel Liang ISBN：978-7-111-41234-2 定价：79.00元

C++程序设计（英文版·第3版）

作者：Y. Daniel Liang ISBN：978-7-111-42305-2 定价：79.00元

C语言程序设计教程（英文版）

作者：H.H. Tan T.B. D'Orazio S.H. Or Maran M.Y. Choy

ISBN：978-7-111-40432-5 定价：49.00元

Java语言程序设计（英文版·第8版）

作者：Y. Daniel Liang

基础篇 ISBN：978-7-111-36122-0 定价：89.00元

进阶篇 ISBN：978-7-111-36125-1 定价：89.00元

推荐阅读













Java编程思想（英文版·第4版）
作者：（美）Bruce Eckel
ISBN：978-7-111-21250-8
定价：79.00元

Java编程思想 第4版
作者：（美）Bruce Eckel
ISBN：978-7-111-21382-6
定价：108.00元

Java核心技术 卷I 基础知识（原书第9版）
作者：（美）Cay S. Horstmann 等
ISBN：978-7-111-44[illegible]X-2
定价：119.00元

Java核心技术 卷II 高级特性（原书第9版）
作者：（美）Cay S. Horstmann 等
ISBN：978-7-111-44250-9
定价：139.00元

Java虚拟机规范（Java SE 7版）
作者：（美）Tim Lindholm 等
ISBN：978-7-111-44515-9
定价：69.00元

Java应用架构设计：模块化模式与OSGi
作者：（美）Kirk Knoernschild
ISBN：978-7-111-43768-0
定价：69.00元

推荐阅读



深入理解计算机系统（原书第2版）

作者：Randal E. Bryant 等 译者：龚奕利 等 ISBN：978-7-111-32133-3 定价：99.00元



计算机系统：系统架构与操作系统的高度集成

作者：Umakishore Ramachandran 等 译者：陈文光 等



计算机系统概论（原书第2版）

作者：Yale N. Patt 等 译者：宋国嘉 等 ISBN：978-7-111-21556-1 定价：49.00元



计算机系统基础

作者：唐志凤 ISBN：978-7-111-46477-8 定价：49.00元

出版者的话
译者序
前言

第1章　计算与工程问题求解导论 …… 1
1.1　历史回顾 …… 1
1.2　现代工程成就 …… 3
1.3　计算机系统 …… 6
1.3.1　计算机硬件 …… 6
1.3.2　计算机软件 …… 6
1.4　数据表示与存储 …… 9
1.4.1　数制 …… 10
1.4.2　数据类型与存储 …… 14
1.5　解决工程问题的方法论 …… 16
本章小结 …… 18
习题 …… 19
第2章　简单的C++程序 …… 22
工程挑战：汽车性能 …… 22
2.1　程序结构 …… 22
2.2　常量和变量 …… 25
2.2.1　科学记数法 …… 27
2.2.2　数值数据类型 …… 27
2.2.3　布尔数据类型 …… 28
2.2.4　字符数据类型 …… 29
2.2.5　字符串数据 …… 30
2.2.6　符号常量 …… 31
2.3　C++类 …… 31
2.3.1　类声明 …… 32
2.3.2　类实现 …… 32
2.4　C++操作符 …… 34
2.4.1　赋值操作符 …… 34
2.4.2　算术操作符 …… 36
2.4.3　操作符的优先级 …… 38
2.4.4　上溢和下溢 …… 40
2.4.5　自增和自减操作符 …… 40
2.4.6　缩写赋值操作符 …… 41
2.5　标准输入和输出 …… 42
2.5.1　cout对象 …… 42
2.5.2　流对象 …… 43
2.5.3　操纵符 …… 44
2.5.4　cin对象 …… 46
2.6　使用IDE构建C++解决方案：NetBeans …… 48
2.7　包含在C++标准库中的基本函数 …… 55
2.7.1　基本的数学函数 …… 55
2.7.2　三角函数 …… 56
*2.7.3　双曲函数 …… 57
2.7.4　字符函数 …… 58
2.8　解决应用问题：速率计算 …… 59
2.9　系统限制 …… 61
本章小结 …… 62
习题 …… 65
第3章　控制结构：选择 …… 68
工程挑战：个体变化 …… 68
3.1　算法设计 …… 68
3.2　结构化编程 …… 69
3.2.1　伪代码 …… 70
3.2.2　可选方案的评估 …… 71
3.3　条件表达式 …… 71
3.3.1　关系操作符 …… 71
3.3.2　逻辑操作符 …… 72
3.3.3　优先级和结合性 …… 74
3.4　选择语句：if语句 …… 74
3.4.1　简单的if语句 …… 75
3.4.2　if/else语句 …… 76
3.5　数值方法：线性插值 …… 79

3.6 解决应用问题：海水的冰点 …… 81
3.7 选择语句：switch语句 …… 85
3.8 使用IDE构建C++解决方案：NetBeans …… 87
3.9 为自定义数据类型定义操作符 …… 93
本章小结 …… 97
习题 …… 98
第4章 控制结构：循环 …… 101
工程挑战：数据收集 …… 101
4.1 算法设计 …… 101
4.2 循环结构 …… 102
4.2.1 while循环 …… 102
4.2.2 do/while循环 …… 105
4.2.3 for循环 …… 107
4.3 解决应用问题：GPS …… 110
4.4 break和continue语句 …… 114
4.5 结构化输入循环 …… 114
4.5.1 计数器控制循环 …… 114
4.5.2 标志控制循环 …… 116
4.5.3 数据终止循环 …… 117
4.6 解决应用问题：气象气球 …… 118
4.7 使用IDE构建C++解决方案：Microsoft Visual C++ …… 122
本章小结 …… 128
习题 …… 129
第5章 使用数据文件 …… 132
工程挑战：天气预报 …… 132
5.1 定义文件流 …… 132
5.1.1 流的类层次 …… 132
5.1.2 ifstream类 …… 134
5.1.3 ofstream类 …… 135
5.2 读取数据文件 …… 137
5.2.1 指定记录的数目 …… 137
5.2.2 标志值 …… 139
5.2.3 文件结束 …… 141
5.3 生成数据文件 …… 143
5.4 解决应用问题：数据过滤器——修改HTML文件 …… 145
5.5 错误检查 …… 148
*5.6 数值方法：线性递减 …… 154
*5.7 解决应用问题：臭氧测量 …… 156
本章小结 …… 160
习题 …… 162
第6章 使用函数进行模块化编程 …… 166
工程挑战：仿真 …… 166
6.1 模块化 …… 166
6.2 自定义函数 …… 168
6.2.1 函数定义 …… 171
6.2.2 函数原型 …… 175
6.3 参数传递 …… 177
6.3.1 值传递 …… 177
6.3.2 引用传递 …… 179
6.3.3 存储类型和作用域 …… 183
6.4 解决应用问题：计算重心 …… 185
6.5 随机数 …… 188
6.5.1 整数序列 …… 188
6.5.2 浮点序列 …… 192
6.6 解决应用问题：仪器可靠性 …… 192
6.7 定义类方法 …… 198
6.7.1 公共接口 …… 198
6.7.2 访问方法 …… 199
6.7.3 修改方法 …… 200
6.8 解决应用问题：复合材料设计 …… 204
*6.9 数值方法：多项式的根 …… 208
6.9.1 多项式的根 …… 209
6.9.2 增量搜索方法 …… 211
*6.10 解决应用问题：系统稳定性 …… 211
*6.11 数值方法：积分 …… 219
本章小结 …… 222
习题 …… 224
第7章 一维数组 …… 229
工程挑战：海啸预警系统 …… 229
7.1 数组 …… 229
7.1.1 定义和初始化 …… 230
7.1.2 伪代码 …… 231
7.1.3 计算与输出 …… 235
7.1.4 函数参数 …… 238
7.2 解决应用问题：飓风等级 …… 241

7.3 统计特征数……245
7.3.1 简单分析……246
7.3.2 方差和标准差……247
7.3.3 自定义头文件……249
7.4 解决应用问题：语音信号分析……250
7.5 排序和搜索算法……254
7.5.1 选择排序……254
7.5.2 搜索算法……256
7.5.3 无序列表……256
7.5.4 有序列表……257
7.6 解决应用问题：海啸预警系统……258
7.7 字符串……263
7.7.1 C风格字符串定义和I/O……263
7.7.2 字符串函数……265
7.8 string类……266
7.9 vector类……267
7.10 解决应用问题：摆率计算……270
本章小结……280
习题……281
第8章 二维数组……285
工程挑战：地形导航……285
8.1 二维数组……285
8.1.1 声明和初始化……286
8.1.2 计算与输出……290
8.1.3 函数参数……292
8.2 解决应用问题：地形导航……297
8.3 二维数组和vector类……300
8.4 矩阵……303
8.4.1 行列式……304
8.4.2 转置……304
8.4.3 矩阵加法和减法……305
8.4.4 矩阵乘法……305
8.5 数值方法：解联立方程……307
8.5.1 图形分析……307
8.5.2 高斯消元法……309
8.6 解决应用问题：电路分析……311
8.7 高维数组……316
本章小结……317
习题……318
第9章 指针……322
工程挑战：大气模式……322
9.1 地址与指针……322
9.1.1 地址操作符……323
9.1.2 指针的分派……324
9.1.3 指针的算术……326
9.2 指向数组元素的指针……329
9.2.1 一维数组……329
9.2.2 字符串……331
9.2.3 指针作为函数参数……332
9.3 解决应用问题：厄尔尼诺南方涛动数据……336
9.4 动态内存分配……338
9.4.1 new操作符……338
9.4.2 动态分配数组……339
9.4.3 delete操作符……339
9.5 解决应用问题：地震监测……340
9.6 使用new和delete的常见错误……345
9.7 链式数据结构……346
9.7.1 链表……346
9.7.2 栈……348
9.7.3 队列……348
9.8 C++标准模板库……349
9.8.1 list类……349
9.8.2 stack类……350
9.8.3 queue类……352
9.9 解决应用问题：文本文件的索引……353
本章小结……357
习题……358
第10章 高级主题……361
工程挑战：人工智能……361
10.1 泛型编程……361
10.2 数据抽象……365
10.2.1 操作符重载……365
10.2.2 像素类……365
10.2.3 算术操作符……367
10.2.4 友元函数……371
10.2.5 验证对象……374

10.2.6 接位操作符 …… 378
10.3 解决应用问题：彩色图像处理 …… 380
10.4 递归 …… 385
10.4.1 阶乘函数 …… 385
10.4.2 斐波纳契序列 …… 387
10.4.3 BinaryTree类 …… 388
10.5 类模板 …… 396
10.6 继承 …… 401
10.6.1 Rectangle类 …… 401
10.6.2 Square类 …… 404
10.6.3 Cube类 …… 406
10.7 鉴方法 …… 409
10.8 解决应用问题：可重复的囚徒困境 …… 411
本章小结 …… 418
习题 …… 419
附录A C++标准库 …… 422
附录B ASCII字符编码 …… 430
附录C 使用MATLAB从ASCII文件中绘制数据点 …… 434
附录D 练习答案 …… 437
附录E 参考文献 …… 445

第 1 章

Engineering Problem Solving with C++, 3e

计算与工程问题求解导论

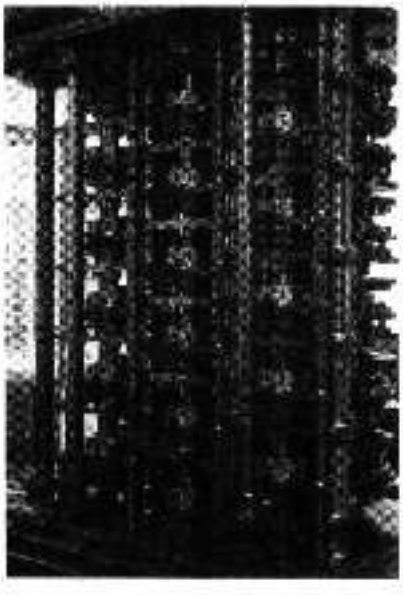
教学目标

本章我们将简要介绍计算与工程问题解决方法，主要内容包括：

- 计算的历史概览
- 现代工程成就
- 有关硬件和软件的讨论
- 关于数字系统的讨论
- 解决问题的“五步法”

1.1 历史回顾

在 19 世纪初，英国数学家 Charles Babbage 提出了计算机的概念，当时他称其为“分析机”（Analytical Engine）。这种分析机设计成可处理十进制数字运算，包括四部分：输入设备、“仓库”或者叫做“存储单元”、处理单元以及输出设备。存储单元由很多组的盘片组成，每个盘片都沿着边沿依次刻上 0 ～ 9 这 10 个数字。处于每组最底层的盘片代表数字的个位数，紧邻其上的代表十位数，依此类推。通过对其组盘片进行排列，就可以表示一个十进制数了。



分析机，Charles Babbage 设计

1842 年，法国工程师和数学家 Luigi F. Menabrea 发表了题为《Charles Babbage 分析机概述——Ada Lovelace 译注》（Sketch of the Analytical Engine Invented by Charles Babbage, Esq. with notes by translator Ada Lovelace）的论文。该文描述了 Babbage 提出的分析机设计概念，这种机器具有输入 / 输出设施，能够执行写在打孔纸上的程序，Babbage 试图通过它解决各种问题。在本章的后续内容中我们将看到，分析机类似于我们现在使用的数字计算机，它们都有输入 / 输出设备、存储单元和处理单元，不过数字计算机使用二进制表示数据，而非十进制。虽然我们现在已经不再使用打孔纸编写程序，但事实上直到 20 世纪 80 年代，打孔纸才慢慢消失。

由于曲高和寡，Babbage 没能获得足够资助来完成他所设计的机器。

分析机的设计概念不是只言片语能说清的，我在此仅说明分析机几点关键的功能，作为抛砖引玉，启发那些愿意接受和尝试其设计及概念的人。



Charles Babbage，1792—1871



Augusta Ada Byron，1815—1852

Augusta Ada Byron 就是那个做好了充分思想准备的人，Ada Byron 是 19 世纪的诗人 Lord Byron 和 Anna Isabella Milbanke 的女儿。在 Ada 出生后不久，由于 Milbanke 认为 Byron 与她的妹妹有婚外恋并且还有些精神失常，Milbanke 离开了 Byron。由于害怕 Ada 遗传她父亲的精神病，本身就是一个业余数学家的 Milbanke 为 Ada 设计的培养方案是进行大量的数学和科学教育。1842 年，Ada 将 Menabrea 讲述分析机架构的论文从法文翻译为英文，在她的译作中包含了关于数值计算的大量评注和详细论述。

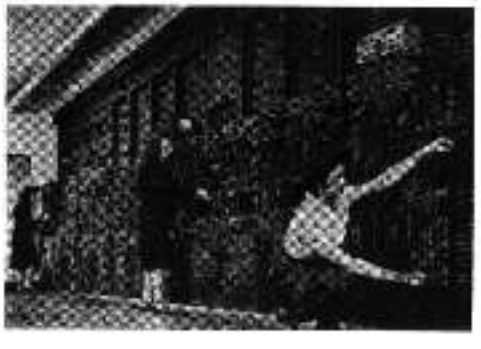
除此之外，Ada Byron 还在她译作的评论里设想了分析机在其他学科方面的诸多潜在用途：

"例如，假设声学和乐曲的音调间的基本关系容易受到表达方式和改编的影响，那么分析机能够求解、科学地创作出具有任意复杂度或任意长度的音乐片段。"

而 Babbage 则说，

"Ada 所做的译注将研究报告原稿的长度扩大了 2 倍。她全力以赴，对于和分析机有关的那些困难而又抽象的问题都作了解释。这两篇备忘录对能够理解那些数理论证的人是很有益的。现在整个开发和分析操作都可以使用机械来执行了。"

许多人认为 Ada Byron 所写的实例是第一个"计算机程序"。后来用于嵌入式系统开发的高级程序语言 Ada，就是用 Ada Byron Lovelace 的名字命名的。



Electronic Numerical Integrator and Calculator（ENIAC）



Intel Pentium 4 处理器

第一台基于二进制的计算机由艾奥瓦州立大学的 John Atanasoff 教授和他的毕业生 Clifford Berry 在 1939～1942 年间建成。这台计算机 ABC（Atanasoff Berry Computer）可

约 700 磅，每 15 秒执行一条指令。第二次世界大战的开始阻止了 Atanasoff 和 Berry 为他们的作品申请专利，但是许多人已经开始研究他们的工作，并且由于计算机的潜力加速了战争的进程。1943 年，美国政府资助了一个由 John Mauchly 和 J. Presper Eckert 所领导的研究团队，该团队负责开发用于计算武器射击表的计算机。他们的工作促进了 ENIAC（Electronic Numerical Integrator and Calculator）的开发。ENIAC 重三十吨，每秒可执行数百条指令，直到 1955 年才从研究领域退役。今天，处理器重量以盎司计，而其运算速度则可达到每秒执行数万亿条指令。

1.2 现代工程成就

在过去的五十年间，在数字计算机的帮助下产生了许多重大的工程成就。这些成就阐明了各个学科的工程属性，并展示了工程学如何改善我们的生活并创造未来的无限可能。下面简要地介绍其中的一些主要成就。

几大主要成就都是与太空探索相关的。1969 年 7 月 21 日，人类历史上第一次登月（moon landing）成功。登月工程可能是有史以来最复杂和最雄心勃勃的工程项目。一些主要的突破都产生在阿波罗宇宙飞船、登月飞行器和土星五号二级火箭的设计过程中。而在宇航服的设计中，则产生了包括一个二件壳和双向包在内的重约 190 磅的系统。计算机不仅在各种系统的设计中扮演着关键的角色，还在单次月球飞行的通信中扮演着关键的角色。一次飞行需要发射控制中心超过 450 人以及其他分布在 9 艘船、54 个飞行器和地面观测站等各处超过 7000 人的共同协作。

空间探索促进了在科学数据收集方面的工程发展并取得很大成就。火星勘探者号（Mars Global Surveyor）是 NASA 在 1996 年发射的用于在环绕火星的轨道上搜集科学数据的宇宙飞船。火星勘探者号上携带的科学仪器包括用于采集图像的火星轨道相机（Mars Orbiter Camera）和用于高速数据收发的高增益天线（High Gain Antenna）。飞船在 2006 年 11 月 2 日完成了它与地球之间的最后一次通信。在 http://mars.jpl.nasa.gov/mgs/gallery/images.html 上可以看到火星轨道相机所采集的一系列图像。

火星勘探者号搜集到的一些照片表明火星表面过去曾经有水的存在，这促使 NASA 又发射了火星侦测轨道器（Mars Reconnaissance Orbiter），后者用于寻找火星表面曾长期有水存在的证据，而水是生命赖以依存的基本物质。火星侦测轨道器使用了洛克希德马丁空间系统所提供的新型飞船设计方案。这是第一个在设计中利用空气制动减速的航天器，轨道器利用火星大气的摩擦力减速并飞行至预定轨道。在 2006 年 3 月 10 日，轨道器开始了它的绕火星飞行。从 2006 年 3 ～ 11 月，航天器利用火星大气减速，通过 400 多次的调整不断降低自己的轨道，最终到达了预定轨道——距离火星表面 155 ～ 196 英里的近圆轨道。

轨道器上的小型侦察影像频谱仪（Compact Reconnaissance Imaging Spectrometer for Mars，CRISM）占了上面搭载的所有仪器的六分之一。CRISM 由约翰霍普金斯大学的应用物理实验室设计建造，它可以通过火星表面反射的阳光中分辨出 544 种颜色，并借此来检测其表面的材料组成，它的最高分辨率比之前任何对火星的观测结果要高出约 20 倍。高分辨率的图像表明了最可能含有水的地点以及最适合火星探测漫游者（Mars Exploration Rovers）着陆的地点。火星探测漫游者用于在火星表面行走，并对火星表面的土壤和岩石进行更详细的检测。

空间项目也为应用卫星的发展提供了动力。应用卫星用于提供大气信息、通信中继、地

理信息，以及有关大气组成的环境变化。全球定位系统（Global Positioning System，GPS）是一个由24颗卫星组成的卫星群，用以向全球广播位置、速度以及时间信息。GPS接收器通过计算其接收到的从GPS卫星发射到接收器的信号来测量时间。只要接收到4颗GPS卫星的信息，接收器的微处理器就可以计算出接收器的准确位置；定位的精度从几米~~几厘米，具体的误差取决于所使用的计算方法。

航空工业是率先使用高级复合材料（advanced composite material）的工业领域。复合材料由可以互相结合的材料组成，其中某种材料可以加强其他材料的纤维。高级复合材料用于制造航空器和航天器，使其具有更轻便、强度高、更耐热的特性。现在复合材料已经用于制造业和体育商品市场。例如，速降滑雪板就使用了多层凯拉弗尔纤维用以增加强度，同时减轻重量。而石墨或环氧树脂制作的高尔夫球杆比传统的钢制球杆强度更高且更轻便。复合材料还用于制造假肢和人造器官。

高级复合材料的研发很大程度上受益于计算机仿真（computer simulation）。用计算机来仿真高级复合材料使许多由于尺寸、速度、安全，或者经济成本而不可能进行的实验成为可能。在仿真中，计算机软件通过计算来模拟物理现象的材料模型。通过不同的数据来重复仿真，就可以了解到这种现象的特征。我们将使用程序用于仿真缩聚塑料的设计过程，溶融盐就是电子组装中隔热防震的关键组件。

计算机轴断层摄影（Computerized Axial Tomography，CAT）扫描仪（CAT scanner）的出现，将药物治疗、生物工程和计算机科学结合到了一起。这种设备通过在不同角度使用X光对物体进行照射，可以生成目标的二维或三维图像。不同角度的X光可以测量这一角度的物体密度，再使用复杂的计算机算法将所有X光的测量信息重构，从而产生物体内部的清晰图像。CAT扫描通常用于查找肿瘤、血块以及脑部异常。美军一直在研究可以用于战场医疗站的加固、轻便的CAT扫描仪。

激光（laser）是一种具有良好的定向性和聚集性的频率相同、光束很窄的光波。CO_2激光器用于在材料上打孔，包括从陶瓷材料到复合材料的各种材料。在医疗过程中，激光还用作黏合分离的视网膜，止血，消除脑肿瘤，以及完成精确的内耳外科手术。称作全息摄影的三维图像也是使用激光生成的。

关于天气、气候以及全球变化的预测需要我们了解大气系统和海洋生态系统。这需要我们了解由于化学物或能量排放引入大气和海洋的CO_2变化、臭氧消耗以及其他气候变化。这种复杂的相互作用还包括来自太阳的作用。来自邻近日冕洞（太阳风的释放点）的太阳风暴的爆发会从太阳表面向地球表面喷射出时速高达100万英里的大量高温气体。这些高温气体的喷发会给地球带来X射线，并且会干扰通信，引起输电线的功率波动。关于天气、气候以及全球变化的预测涉及大量研究数据的收集和用来表示各种变量关系之间相关性的新数学模型。在后面的章节中我们将对丹佛国际航空港一年的天气模式进行分析，我们还将使用卫星数据建立模型，用来对中层大气中的臭氧比例进行预测，我们还将分析充满氦气的气象气球的高度与速度信息。

计算机语音识别（computerized speech understanding）可能会彻底改变现有的通信系统，但目前仍有许多问题需要解决。当前所使用的语音识别软件所能识别的词汇集还相当小，要开发一个与说话人无关的、词汇集很大且支持不同语言的系统是十分困难的。即使是同一个人声音上的细微变化，如患上感冒或紧张时，都会对语音识别的效果造成影响。而假设计算机可以正确识别出词汇，确定词汇的意思也不是那么简单。许多词汇的意思都是与上下文相

关的，因此不能脱离语境来分析它的含义。语调的变化也会影响语句的含义，如提高声调可能会让陈述句变为疑问句。尽管现在还有诸多难题有待解决，但是许多令人兴奋的应用已经遍布我们周围了。想象一下，一台可以分辨说话者的语言并将其翻译为对方可以听懂的语言的电话系统会给人们生活带来怎样的变化。我们将对真实的语音数据进行分析，用以展示语音识别中的相关技术。

变化的工程环境

21世纪的工程师将工作在一个需要许多非技术性的技巧和能力的环境中。虽然对于大多数工程师来说，计算机仍是基本的计算工具，但计算机对于提升其他非技术性的能力也是大有裨益的。

工程师需要很强的沟通技巧（communication skill），包括口语表达和书面表达。计算机提供了用于帮助编写函要、准备演讲和技术报告材料的软件。本章最后的问题包括写作和口头表达的作业，用以锻炼这些重要的能力。

“设计－处理－建造”（design/process/manufacture）是一条从设想到产品的必由之路。每个工程师都必须首先懂得这个过程。这个流程中用到计算机的部分包括设计分析、机器绘制、机器装配、质量保证和市场分析。本书中的几个问题都与这些主题相关。例如，第5章中的程序就是用来仿真使用了多个组件的系统的可靠性。

关于近50年的工程成就的讨论已经清楚地表明了工程团队具备跨学科的特点，未来的工程师将需要跨学科团队去迎接交叉学科的挑战。对于工程师来说，学会团队内的交流和建立高效的团队沟通机制是一种十分重要的能力。培养这种能力的好办法之一就是建立一个学习团队。团队中的每个成员都被分配给特定的主题，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。

21世纪的工程师需要理解全球市场（world marketplace），这包括理解不同的文化、政治制度和商业环境。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。

工程师是问题解决者，但是问题并不是一成不变的。一个工程师必须能够从关于问题的讨论中提取出问题的描述，然后确定与问题相关的重要事项，不仅包括建立程序，还要学习在混乱中找出各种关联。这意味着不仅要分析（analyzing）数据，还要从众多零碎的信息中形成（synthesizing）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。对于环境的考虑应该体现为对于不同的环境有可选的解决方案。工程师在提供测试结果、质量验证和设计约束时应当考虑伦理问题（ethical issue）。伦理问题从来都是不易解决的，某些新的令人兴奋的技术成就也伴随着更多的伦理问题。例如，基因组技术可能会导致潜在的伦理、法律和社会影响。应当允许医生使用基因疗法来治疗糖尿病或者来提高运动员的能力吗？应当将一个未出生婴儿的物理和精神特征的详细信息提供给准父母吗？对于个人的基因编码应该使用什么样的隐私策略呢？福兮祸所伏，新的技术成就是一把双刃剑，往往带来许多复杂的问题，有利有弊。

工程师们需要构建知识、自信和充分理解21世纪，这里只给出一点建议而已。下面，我们将怀热情地开始介绍有益于工程师的计算机系统，以及本书中使用C++解决工程问题

时所用到的方法论。

1.3 计算机系统

在介绍 C++ 之前，先简要了解一下计算机系统无疑是很有帮助的。计算系统是一个完整的工作系统。系统不仅包括计算机，还包括各种软件以及外部设备。计算机就是用于执行指定操作的机器，这些操作通常是一组指令的集合，这些指令也称为软件（software）。计算机硬件（hardware）是系统的物理组成部分，是可以真真切切触摸到的。硬件包括计算机及其周边设备，如键盘、鼠标、显示器、硬盘、打印机，甚至打印墨水等。

计算机软件是以电子形式在硬件中驻留和运行的程序，如编译器、操作系统以及应用程序等。软件提供了人机接口（Human Computer Interface, HCI），并定义了计算机应执行的操作。

1.3.1 计算机硬件

回想一下 Menabrea 关于 Babbage 分析引擎的描述：它是一种使用输入设备、输出设备和写在打孔纸上的程序来解决问题的机器。1946 年，冯·诺伊曼（John von Neumann）提出了一种计算机模型（如图 1.1 所示）。今天该模型仍然为绝大多数的数字计算机所采用。在冯·诺伊曼的模型里，我们可以看到输入设备、输出设备、存储单元，以及由控制单元（control unit）和带有累加器（accumulator）的算术逻辑单元（Arithmetic Logic Unit, ALU）构成的模块。存储单元存储数据，控制单元则控制数据的传输和数据的处理。控制单元取回指令并对其进行译码后存入存储单元中，同时从输入设备接收数据，如来自键盘、鼠标的数据，并将数据发送到特定的输出设备，如打印机或显示器，并将数据存入存储单元。

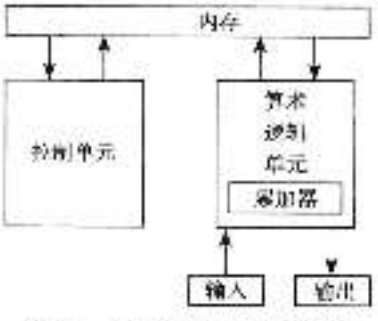


图 1.1 von Neumann 计算机型

假定我们写这样一个程序，该程序对计算机发出指令——将两个数相加并将结果显示出来。整个程序的执行过程类似于这样：控制单元对可执行指令进行译码，并将数据送入 ALU。ALU 执行加法操作，控制单元则将计算结果送入输出设备。这里的控制单元和 ALU 一起称为中央处理单元（Central Processing Unit, CPU）。ALU 中的累加器是一组高速寄存器的集合，用作算术和逻辑操作的操作数和结果的临时存储。与 ALU 执行算术或逻辑操作的时间相比，访问存储单元的时间要长得多。因此，使用 ALU 内部的寄存器提高了执行操作的整体速度。ALU 中的寄存器大小对应着计算机的字长（word size）。一般的字长有 16 位、32 位、64 位。这取决于处理器的设计方案。注意，1 位代表一个二进制位，而且处理器的字长都是 2 的幂。

1.3.2 计算机软件

计算机软件包含了我们希望计算机执行的指令或命令。在计算机软件中的分类中有几类比较重要，包括操作系统、软件工具、各种程序语言的编译器。图 1.2 说明了这几类软件之间的关系以及与计算机硬件的关系。下面我们将详细讨论这几类软件。

操作系统（operating system）。有一些软件，如操作系统，在购买计算机硬件时就已经安装在计算机中了。通过为用户（比如你）提供便捷和高效的操作环境，操作系统在用户和

硬件之间建立了良好的接口，用户可以在自己的系统之上选择各种软件执行。

操作系统包含了一组实用程序，你可以使用它们完成诸如文件打印、文件复制、列出文件系统中存储的文件等各种功能。虽然不同的操作系统命令执行环境差异很大，但是绝大多数操作系统都包含这些实用程序。例如，在 UNIX（一种常用作工作站的强大操作系统）或 Linux（UNIX 的 PC 版本）上列出文件清单的命令是 ls。一些操作系统通过使用图标和菜单简化了与用户的接口，比如 Macintosh 和 Windows 就提供了便于用户使用的图形用户接口（Graphical User Interface, GUI）。

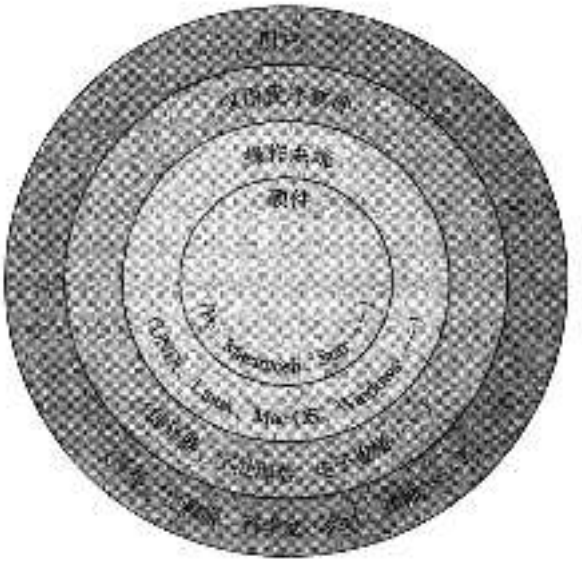


图 1.2 计算机的软件接口

因为 C++ 程序可以运行在多种平台和计算机系统上，而且每台计算机上都可以使用不同的操作系统，所以本书不再讨论可能用到的操作系统的广泛差异性。假定你的课程教授已经提供了课程学习时所要使用的操作系统的信息，这些信息也包含在操作系统手册中，通过帮助菜单可以找到。

应用软件。多数应用程序都能够完成一些常用的功能操作，例如，类似 Word 和 Pages 等的字处理程序（word processor），能够帮助你创建简历、通讯和报告等格式化文档。字处理程序支持输入数学公式，还能帮你检查拼写和语法。它还提供了帮你创建带标题和内容的图表的工具。文本编辑器（text editor），如 vi、记事本和写字板，是用于创建诸如 C++ 应用程序一样的文本文件和其他数据文件。更复杂一点的文本编辑器，如 emacs，还包含了相关编译器，并提供了界面友好的环境用于应用程序开发。电子表格软件（spreadsheet software）则是以行列表格的形式来显示数据的软件，它能够帮助你更方便地处理数据。电子表格最初是用于会计和会计应用方面的，但是使用电子表格也能轻松地解决许多科学和工程问题。大多数电子表格包都具有数据制图功能，因此在分析和展现信息上特别有用。Lotus 1-2-3、OpenOffice 和 Excel 都是流行的电子表格软件。

另一类流行的软件工具是诸如 MySQL 和 Oracle 一样的数据库管理软件（database management software）。这些软件可以使你更高效地存储和读取大量的数据。许多 Web 应用和搜索引擎都使用数据库，此外还有银行、医院、旅馆、航空公司等机构都使用数据库。用于科学研究的数据库一般用于分析大规模的数据，气象数据就是需要用大数据库来存储和分析的科学数据。

计算机辅助设计软件（computer-aided design software），如 AutoCAD、Land Development Desktop、Civil 3D 和 Architectural Desktop 等，能帮助你定义各种对象并通过图形化的方式对它们进行操作。例如，你可以定义一个对象，然后从不同的角度来观察或者查看对象从一个位置旋转到另一个位置的过程。

还有一些功能强大的数学计算软件（mathematical computation software），如 MATLAB、Mathematica 和 Maple。这些工具不仅有功能强大的数学命令，还支持生成图像等扩展功能。

这种强大的计算能力和可视化功能的结合使得它们成为了工程师们的好帮手。附录C中包含了有关使用MATLAB从一个由C++程序生成的数据文件中读取数据并进行数据制图的过程。

如果某个工程问题可以通过使用软件工具解决，通常来说这比写一个程序来解决问题更高效。然而，很多问题都不能通过现有软件工具来解决，或者软件工具在需要解决问题的计算机系统上不可用，因此，我们也需要知道如何使用计算机语言来写程序。随着一些功能强大的软件的出现，这些软件除了支持专门的操作还包含了编程语言，如MATLAB和Mathematica，这使得软件工具和计算机语言之间的区别变得越来越模糊。

计算机语言。计算机语言有不同的层次。机器语言（machine language）是最基础的语言，是与计算机硬件设计密切相关的。因为计算机的设计是建立在二态技术（类似的情况有电路的开合、开关的打开与关闭、电池的正负极）基础上的，所以机器语言也使用两种符号来编写，通常使用数字0和1来表示。因此，机器语言也是一种二进制语言，它的指令都是使用由0和1组成的序列写成的。这些序列称为二进制串。因为机器语言是与计算机硬件紧密相关的，所以SUN计算机上与HP计算机上的机器语言是不同的。

对于某种特定的计算机设计而言，汇编语言（assembly language）也是唯一的，但是汇编语言的指令是用可读性更好的符号语句而非二进制串编写的。汇编语言的语句类型通常并不是很多，因此编写汇编程序时可能会觉得乏味。此外，在使用汇编语言时，你必须了解与之相应的硬件信息。包含微处理器的设备通常要求程序能够极快地执行，这样的程序称为实时程序（real-time program）。实时程序通常使用汇编语言编写，这样可以发挥特定计算机硬件的优势，以提高执行速度。高级语言（high-level language）是具有类似自然语言的命令和指令的计算机语言，C++、C、Fortran、Ada、Java、Basic等都是高级语言。使用高级语言编写程序比使用机器语言或汇编语言编写程序要容易得多。高级语言包含大量命令和使用这些命令的语法（syntax）的扩展集合。为了说明不同高级语言中的语法和符号，我们以给定直径来计算圆的面积为例，将几种不同语言的表示方式列在表1.1中。请注意在这个简单的计算中不同语言的相似之处和不同之处。

表1.1 不同语言的语句比较

软　件	示例语句
C++	area = 3.141593 * (diameter/2) * (diameter/2);
C	area = 3.141593 * (diameter/2) * (diameter/2);
MATLAB	area = pi * (diameter/2) ^2;
Fortran	area = 3.141593 * (diameter/2.0) **2
Ada	area := 3.141593 * (diameter/2) **2
Java	area = Math.PI*Math.pow (diameter/2,2);
Basic	a = 3.141593 * (d/2) * (d/2)
Scheme	area 3.141593 * (d/2) * (d/2)

执行计算机程序。要执行使用诸如C++这样的高级语言编写的程序，必须先将高级语言指令翻译成机器语言。执行这种翻译任务的程序称为编译器（compiler）。因此，要在计算机上编写和执行C++程序，计算机软件中必须包含一个C++编译器。对于不同的计算机硬件，从超级计算机到便携计算机，都有相应可用的C++编译器。

如果编译器在编译时检测到错误，它会将错误信息打印出来。我们必须更正自己的程序语句，然后再次执行编译。在编译阶段出现的错误称作解析错误（parse error）或语法错误（syntax error）。例如，如果我们想用变量sum除以3，则在C++中正确的表达式是sum/3。如果我们在表达式中使用了不正确的反斜线，即sum\3，那么编译器在编译程序时将给我们一个语法错误并打印出错误消息。在程序编译完全正确之前，必须经过若干轮“编译→修改（修改出现语法错误的语句）→重编译”这样的过程。当没有语法错误后，编译器就可以

成功地翻译我们的程序，并生成机器语言形式的程序。生成的程序将完成我们最开始的C++程序所要执行的功能。这里，C++程序称为源文件（source file），而生成的机器语言版本则称为目标文件（object file）。因此，源程序和目标程序所描述的功能是一致的，但是源程序是用高级语言编写的，而目标程序则是以机器语言的形式。

一旦程序成功通过编译，就应当完成剩下的步骤以使目标程序可以执行（execution）。这些步骤包括将目标程序与机器语言语句进行链接（linking）和把程序载入（loading）内存。在链接及载入之后，计算机就可以执行程序了。在执行阶段出现的错误称为执行错误（execution error）、运行时错误（run-time error）或逻辑错误（logic error）。这些错误也叫做程序bug。执行错误通常会导致程序终止。例如，程序语句可能试图引用某个非法的内存地址，这将会引发执行错误。某些执行错误并不会导致程序终止，但是会导致计算结果出错。程序员在确定解决方案的正确步骤时会导致这种类型的错误，也会在程序处理数据出错时发生。如果执行错误是由于程序中所编写的语句产生的，我们必须在源程序中修正错误并重新进行编译。这个过程叫做调试（debugging），并且可能会花费很长时间。即使程序看起来执行正常时，我们也必须仔细检查结果以确保结果是正确的。计算机会准确地执行我们所指明的每一个操作。如果我们指定了错误的操作，它也会执行错误的（但是语法上正确的）操作，然后返回给我们错误的答案。图1.3给出了编译、链接/载入、执行的流程。

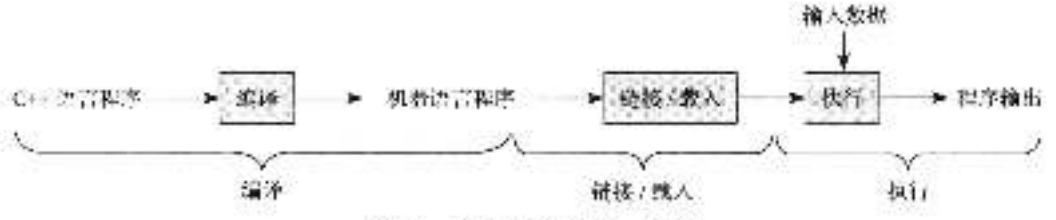


图1.3 程序编译、链接、执行

C++编译器通常都会提供友好的界面供用户进行C++程序的编写和测试。例如Microsoft Visual C++包含了文字处理程序，这样程序的编写、编译和执行都在同一个软件中实现了；而使用独立的文字处理程序，则要使用操作系统命令在文字处理程序和编译器之间来回奔波。许多C++编程环境都包括了调试程序，这对检查程序中的错误是很有用的。调试程序允许我们查看程序执行过程中不同时刻存储的变量值，并可以对程序单步地逐行执行。

当我们使用C++中的新语句时，将会指出与之相关的常见错误以及定位与这些错误相关的有用技巧。在每章的结尾我们将总结相关的调试技巧。

1.4 数据表示与存储

回想一下分析机的存储单元设计：由无数个由盘片形成的卷组成，每个盘片上刻有0～9这10个数字。通过排列某个卷上的盘片，就存储了一个十进制数。

现代数字计算机的设计原理与之相似，但是信息是用二进制表示的，而非十进制。在基于二进制的系统中只有0和1两个数字。因此在数字计算机中一个二进制位可以使用一个位（bit）表示。位上的值在任何时候都只能是0或1。从硬件的角度来说，当该位处于关闭或低电位时，它的值为0，而当其处于打开或高电位时，其值为1。

二进制数在内存中以位序列的形式存储，位序列称为字（word）。字的最右边的数位是1的倍数，相邻的下一数位表示2的倍数，再下一数位表示4的倍数，最左边的数位表示2^{n-1}的倍数，这里的n是二进制数所具有的位数。字的长度每增加1位，字所表示的数的大小就以2的幂增加，而其所能表示的范围就翻一倍。内存中可用的字的数目称为内存空间，或者地址空间（address space）。

图1.4给出了地址空间为8（2^3）、字长为16（2^4）的内存组成。存储在地址000的二进制数的值为$0000101011011101_2 = 2781_{10}$。需要重点注意的是，字长决定了存储在某个地址中的数值范围，而地址空间则决定了可以存储多少个字。对于分析机，它的字长是由一个卷上的盘片的数目确定的，而它的存储空间则是由卷数决定的。

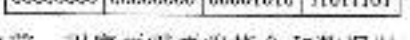
地址	16位字
000	0000101011011101
001	0101100010010000
010	0001011010010100
011	0111011010011011
100	0000000000000000
101	0001000010010000
110	0111111111111111
111	0001011010010110

图1.4 地址空间为8，字长为16的内存组成

C++包含整型数、浮点数、字符和布尔值等内建的数据类型。每种数据类型都有一个以字节（byte）为单位来确定的预定义大小，1字节就是一个8位的序列。为了定义标识符（identifier）并为它分配空间，则需要使用类型声明语句（type declaration statement）。当定义一个标识符时，便确定了其数据类型，并在内存中分配相应字节数的空间。例如类型声明语句

```
int iValue = 2781;
```

定义了一个名为iValue的标识符，它表示存储在内存中的某个整数的第一个字节，对应内存中存储的初始值是2781_{10}的二进制表示，对应的内存组成如下所示。



在编译器和高级编程语言出现之前，程序员需要将指令和数据以二进制形式载入内存。现在我们已经有了可以完成这些任务的软件（编译器、链接器和加载器）。在第2章中我们将开始讨论C++语句和数据类型，但这里我们将首先讨论数制和数据的底层表示，这将有利于我们更好地理解程序执行时，其中的操作是如何进行的。

1.4.1 数制

十进制中有10个数字（0～9），每个数位上的数字都要乘上一个10的幂。大多数人都能很容易地按照十进制来数数，也很容易理解基本十进制的数。当我们读出数字245_{10}时，读作二百四十五。我们将2看作是百位数（10^2），4看作十位数（10^1），而5则是个位数（10^0）。如果我们将数字写成下面的形式：

$$2 * 10^2 + 4 * 10^1 + 5 * 10^0$$

则可以计算加法$200 + 40 + 5$得到245_{10}。在下面的章节中，我们将讨论三种数制：二进制、八进制和十六进制，这几种数制在学习数字计算机系统时都是很有帮助的。我们还会讨论不同数制之间的转换算法。

二进制数。数字计算机以二进制形式来表示数据。二进制中包括两个二进制数：0和1。在二进制数中，每个数位上的数字都需要乘上2的幂次。现在我们看看图1.4中存在地址000中的二进制数0000101011011101_2，若我们想知道它的十进制表示，则可以按照下面的形式将它展开，最右边的二进制位乘以2^0，最左边的二进制位乘以2^{15}，依此进行。

$$0000101011011101_2$$

$$-0*2^{15}+0*2^{14}+0*2^{13}+0*2^{12}+1*2^{11}+0*2^{10}+1*2^9+0*2^8+1*2^7$$
$$+1*2^6+0*2^5-1*2^4+1*2^3+1*2^2+0*2^1+1*2^0$$
$$-0+0+0+0-2048+0+512+0+128-64+0+16+8+4+0+1$$
$$-2781_{10}$$

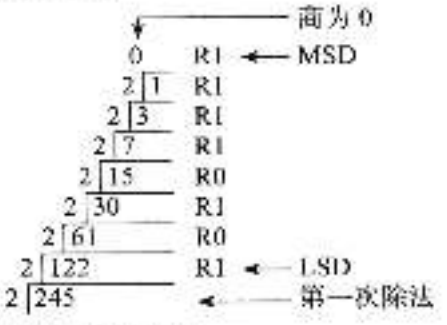
我们在后面的章节中将看到，上面的算法适用于将任意进制的数转换成十进制表示。

假如我们现在需要将一个十进制的数转换成二进制表示，转换后的二进制数将是一个二进制数字序列。为了生成这个数字序列，我们将采用以十进制数不断除以 2 的转换算法，通过记录每次除法所得到的余数并将其形成一个连续的数字序列，即可得到我们想要的二进制数。

通过下面的例子，我们来看如何使用这个算法。注意，整个除法过程到得到的商为 0 时终止，第一次除法所得到的余数作为二进制数的最低有效位（Least Significant Digital, LSD），最后一次除法所得到的余数作为二进制数的最高有效位（Most Significant Digital, MSD）。

[示例]

$245_{10}=?_2$



$$245_{10}=11110101_2$$

上面的转换算法也适用于将十进制数转换为任意其他数制表示，但是除数则要对应相应的数制基数（二进制为 2，依此类推），在下面的章节中我们将看到这些计算方式。

八进制数。八进制中有八个数字（0～8）。八进制数对压缩 8 位字符编码比较有用，也便于理解 Linux/UNIX 平台上的文件和目录的权限设置。例如，我们希望让某个文件对每个用户都具有读、写、执行权限，那么我们可以使用 chmod 命令来对这个文件设置权限：

```
chmod 777 aFile
```

这里对于每个用户而言，aFile 的权限都将设置为 7₈（或者写成 111_2）。如果我们接下来执行命令

```
ls -l
```

将看到类似下面的输出结果：

```
-rwxrwxrwx 1 jeanlnel jeanlnel 256 Jun 26 12:27 aFile.
```

上面的输出结果表明，读、写、执行的权限已经被授予所有用户（程序所有者、组、其他用户）。如果我们只想给予组和其他用户读权限，那么我们可以使用下面的命令来变更权限：

```
chmod 744 aFile
```

再次使用 ls 命令可以得到如下输出：

```
-rwxr--r-- 1 jeanlnel jeanlnel 256 Jun 26 12:27 aFile
```

这表明文件所有者具有读、写、执行权限，而其他用户（组、其他用户）则只有读权限，而没有写权限和执行权限。这里权限值 $4_8 = 100_2$，因此为 1 的为授予权限，而为 0 则为拒绝权限。

八进制数中的每个数位都要乘以 8 的幂。为了将一个八进制数转换成十进制表示，我们仍然使用将十进制数转换为二进制数的算法。只是这里我们需要乘的是 8 的幂次。

【示例】

$$217_8 = ?_{10}$$
$$2 * 8^2 + 1 * 8^1 + 7 * 8^0$$
$$= 2 * 64 + 1 * 8 + 7 * 1$$
$$= 128 + 8 + 7 = 143_{10}$$

为了说明将十进制数转换成任意其他进制数的算法，我们将 143_{10} 变换成八进制表示。这里我们的算法中使用 8 作为除数。

$$143_{10} = ?_8$$

```
  0   R2
8|2   R1
8|17  R7
8|143
```

$$143_{10} = 217_8$$

到这里为止，我们已经说明了将十进制数转换成任意其他进制数，以及将任意其他进制数转换成十进制数的方法。但是现在假定我们要将一个八进制数转换成二进制数，那又该如何呢？一种方法是先将八进制数转换成十进制数，再将十进制数转换成二进制数。这种方法没有问题，但是由于 8 是 2 的幂，$8 = 2^3$，每个八进制位可以用三个二进制位来表示，利用这种关系我们可以快速地在八进制和二进制之间进行相互转换，下面的两个例子将说明这种做法。为了参考方便，表 1.2 列出了每个八进制数字的二进制表示。

【示例】

$$143_8 = ?_2$$

```
1    4    3
↓    ↓    ↓
001  100  011
```

$$143_8 = 001100011_2$$

【示例】

$$0001010110111101_2 = ?_8$$

```
000  101  011  011  101
 ↓    ↓    ↓    ↓    ↓
 0    5    3    3    5
```

$$0001010110111101_2 = 5335_8$$

表 1.2 八进制数字的二进制表示

八进制数字	二进制表示
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

练习

将下面的十进制数转换成二进制数。

1. $921_{10} = ?_2$　　2. $8_{10} = ?_2$　　3. $100_{10} = ?_2$

将下面的八进制数转换成十进制数。

4. $100_8 = ?_{10}$　　5. $247_8 = ?_{10}$　　6. $16_8 = ?_{10}$

将下面的各数转换成要求的进制数。

7. $100_2 = ?_8$　　8. $3716_8 = ?_2$　　9. $110100111_2 = ?_{10}$

10. $221_6 = ?_8$

十六进制数。十六进制中包括 16 个十六进制数字（0 ~ 9，A ~ F），每个数位上的数字都要乘上 16 的幂。其中用字母表示的十六进制数字对应的十进制值分别是（10、11、12、13、14、15）。

$2BF_{16}$ 是一个十六进制数的例子，使用将其他进制数转换成十进制数的算法，我们可以得到对应的十进制数，如下所示。

【示例】

$$2FB_{16} = ?_{12}$$

$$2 * 16^2 + F * 16^1 - B * 16^0 = 2 * 256 - 15 * 16 + 11 * 1$$

$$= 512 - 240 + 11 = 763_{10}$$

使用将十进制数转换成其他进制数的算法，我们可以将 763_{10} 转换回十六进制，这里每次除法得到的余数都是 0 ~ F 之间的值。

【示例】

$$763_{10} = ?_{16}$$

```
   0     R2
 16|2    R15(F)
16|47    R11(B)
16|763        ↑——十六进制
```

$$763_{10} = 2FB_{16}$$

16 也是 2 的幂，$16 = 2^4$，并且每个十六进制数字都可以用 4 个二进制位表示。利用这种关系我们可以快速地在十六进制和二进制之间进行相互转换，下面的两个例子将说明这种做法。为了方便参考，表 1.3 列出了每个十六进制数字的二进制表示。

【示例】

$$7BF_{16} = ?_2$$

```
 7      B      F
 ↓      ↓      ↓
0111   1011   1111
```

$$7BF_{16} = 011110111111_2$$

【示例】

$$1010001101_2 = ?_{16}$$

```
1010   000   1101
 ↓     ↓     ↓
 A     0     D
```

$$101000001101_2 = A0D_{16}$$

表 1.3　十六进制数字的二进制表示

十六进制数字	二进制表示	十六进制数字	二进制表示
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

练习

将下面的十进制数转换成十六进制数。

1. $921_{10} = ?_{16}$　　2. $8_{10} = ?_{16}$　　3. $100_{10} = ?_{16}$

将下面的十六进制数转换成十进制数。

4. $1C0_{16} = ?_{10}$　　5. $29B_{16} = ?_{10}$　　6. $16_{16} = ?_{10}$

将下面的各数转换成要求的进制数。

7. $10010011_2 = ?_{10}$　　8. $3A1B_{16} = ?_2$　　9. $110100111_2 = ?_8$

10. $261_8 = ?_5$

1.4.2 数据类型与存储

当数据存储在内存中时，它是以位序列存在的。这些位序列可能是指令、数字、字符、图像或数字信号的一部分，抑或是其他类型的数据。例如我们看一个位序列 01000110_2，它的十进制值是 70，同时它也可以是美国国家标准机构（American National Standard Institute，ANSI）颁布的字符编码中的字符 F。

在工程学、数学以及计算机科学中，数据表示已经变成一个越来越重要和越来越令人感兴趣的领域。随着计算机的能力变得越来越强以及计算机在生物计算、通信以及信号处理等数据密集型应用中的广泛使用，其所产生和需要处理的数据量也在不断增长，在数据的定义和表示上也面临着新的挑战。本节我们将讨论整型数和浮点数两种基本数据类型的表示。

整型数据类型。在前一节中，我们使用了将十进制转换成二进制的算法。这个算法适用于任意的十进制整数，所以理论上我们可以用二进制表示所有的十进制数。但在实际中，我们可能会受到计算机系统的字长限制。在内存中，整型数据通常使用 4 字节（32 位）存储。最左边的位用作符号位，其余的 31 位表示数字的数值部分。

为了简单起见，在下面的例子中我们采用的字长为 8。在这 8 位中可表示的最大有符号整数为 2^7-1 或 127_{10}，如下所示。

0	1	1	1	1	1	1	1

数字计算机中的数据表示方式影响着算术和逻辑运算的效率。许多计算机系统按照上面的方式存储正整数，而以 2 的补码形式（2's complement form）来存储负整数。对于算术运算而言，以 2 的补码形式来存储负整数可以避免对符号位进行检查，这样提高了执行效率。首先，我们学习如何得到一个负整数的补码形式，然后说明以 2 的补码形式来存储负整数是如何简化算术操作的。

一个二进制数的 2 的补码是通过将其所有位取反后再加 1 得到的。将一位取反意味着将该位的值由 0 变成 1，或者 1 变成 0。将二进制数的所有位取反得到了它关于 1 的补码形式（1's complement form）。在 1 的补码上加 1 即可得到 2 的补码。下面的例子中展示了一个字长为 8 的数 -127_{10} 是如何计算得到它关于 2 的补码的。

【示例】计算值 -127_{10} 的 2 的补码表示。

为了得到负整数的 2 的补码表示，我们首先将其作为无符号整数转换成二进制表示。

$$127_{10} = 01111111_2$$

接着，对所有的位取反得到关于 1 的补码。

$$10000000_2$$

最后，我们对 1 的补码加 $1_{10} = 00000001_2$，得到关于 2 的补码。

$$10000001$$

因此 -127_{10} 的 2 的补码形式是 10000001。

注意，当我们把 127_{10} 与 -127_{10} 的关于 2 的补码形式相加时发生了什么？

	01111111_2	127_{10}
+	10000001_2	-127_{10} 的关于 2 的补码
=	00000000_2	加法的结果是 0

以 2 的补码形式存储有符号整数有这样的特性：对于任意整数 n，将其与 n 关于 2 的补码相加，其最后结果为 0。2 的补码形式的另一重要特性是其关于 0 的表示是唯一的。

当以 2 的补码形式进行有符号整数加法时，如果最后结果为负，那么结果将是以其关于 2 的补码形式存储的，如下所示。

【示例】二进制加法，结果为正。

	11110110	-10_{10} 以 2 的补码形式表示
+	00001101	13_{10}
=	00000011	3_{10}

【示例】二进制加法，结果为负。

	00001010	10_{10}
+	11110011	-13_{10} 以 2 的补码形式表示
=	11111101	-3 以 2 的补码形式表示

练习

按出下面整数关于 2 的补码。

1. 1100111_2　　2. -192_{10}　　3. -45_8

浮点数据类型。浮点数也叫做实数，比如 12.25，其中包含了一个小数点。小数点的左边是整数部分，右边是小数部分。小数部分可以通过以小数部分不断乘 2 转换为二进制表示，整个过程中小数部分不断乘 2，直到小数部分变为 0 为止，并将每次乘法结果中的进位（carry bit）记录下来。下面的例子给出了算法的过程。

【示例】将 12.25_{10} 转换成二进制。

首先，将整数部分 12_{10} 转换成二进制。

```
 0   R1
2|1   R1
2|3   R0
2|6   R0
2|12
```

因此，$12_{10} = 1100_2$。

接下来，将小数部分 $.25_{10}$ 转换成二进制。通过将小数部分不断乘 2，并记录每次乘法结果的进位。

$.25*2 = 0.5$ C0

$.5*2 = 1.0$ C1

$.25_{10} = .01_2$

将整数部分和小数部分的转换结果组合起来，就得到了整个数的二进制表示。因此，$12.25_{10} = 1100.01_2$。

要将二进制的浮点数 1100.01_2 转换回十进制，方法与将二进制整数转换成十进制相同，但是小数部分的各位数所乘的要变为 2 的负指数，如下所示。

$$1100.01_2 = 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2}$$

$$= 1*8 + 1*4 + 0*2 + 0*1 + 0*\left(\frac{1}{2}\right) + 1*\left(\frac{1}{4}\right)$$

$$= 8 + 4 + 0 + 0 + 0 + 0.25 = 12.25$$

$1100.01_2 = 12.25_{10}$

在上面的例子中，我们得到了 12.25_{10} 二进制准确表示形式。遗憾的是，在下面的例子

中我们将看到，许多浮点数都只有近似的二进制表示。

【示例】将 12.6_{10} 转换成二进制。

将整数部分 12_{10} 的二进制表示为 1100_2。将小数部分转换成二进制，我们不断乘 2，并记录进位。

.6 * 2 = 1.2 C1
.2 * 2 = 0.4 C0
.4 * 2 = 0.8 C0
.8 * 2 = 1.6 C1
.6 * 2 = 1.2 C1
.2 * 2 = 0.4 C0
.4 * 2 = 0.8 C0
.8 * 2 = 1.6 C1

可以看到，最终得到的位形式将会以 1001 一直重复下去，小数部分永远不会变为 0 而终止。因此，0.6_{10} 最接近的表示是二进制数 $0.100110011001\cdots_2$。如果取 8 位精度，则可以得到：

$$1*2^{1}+0*2^{0}+0*2^{-1}+1*2^{-4}+1*2^{-5}+0*2^{-6}+0*2^{-7}+1*2^{-8}$$

$$-1*\left(\frac{1}{2}\right)-0*\left(\frac{1}{4}\right)-0*\left(\frac{1}{8}\right)+1*\left(\frac{1}{16}\right)+1*\left(\frac{1}{32}\right)+0*\left(\frac{1}{64}\right)+0*\left(\frac{1}{128}\right)+1*\left(\frac{1}{256}\right)$$

$$-0.5+0+0+0.0625+0.031\ 25+0+0+0.003\ 906\ 25=0.597\ 656\ 2$$

$$12.6_{10}\approx 1100.10011001_2$$

认识浮点数的二进制表示是近似而非相等这一点很重要。这将会影响到我们在程序中使用和测试浮点数，还会影响到数值计算的准确性。

1.5 解决工程问题的方法论

解决问题不仅仅是工程课程的关键部分，也是计算机科学、数学、物理学和化学课程的关键部分。因此，找到一种统一的方法来解决问题是很重要的。如果这种方法足够抽象，能适用于各种不同的领域，就不用为了解决数学问题去学习一种技术，然后为了解决物理问题去学习另一种技术。我们解决工程问题的过程也可以用于解决其他领域的问题，但是这都建立在我们使用计算机来帮助解决问题的前提下。

我们所使用的解决问题的过程或者方法论将贯穿本书全文，包括以下五个步骤：

1）清楚地描述问题。

2）描述问题的输入和输出信息。

3）使用一组简单的数据来解决问题。

4）设计解决方案并将其用计算机程序实现。

5）使用大量数据测试解决方案。

现在我们用一个计算平面上两点之间距离的例子来说明上述步骤。

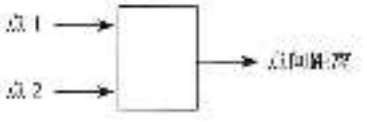
1. 问题描述

第一步是将问题描述清楚。为了避免错误的理解，给出清楚、简洁的问题描述是很重要的。对于这个问题的描述如下：

计算一个平面上两点之间的直线距离。

2. 输入/输出描述

第二步是描述解决问题所需的信息或者数据，并指明最终解决问题需要计算的值。这些值都将用 C++ 程序中的对象表示。这些对象标识了问题的输入和输出，统一称为 I/O。对许多问题来说，使用图示来表示输入与输出是很有用的。在这里，程序还只是一个抽象，因为我们还没有确定如何得到输出，而只是知道计算输出所需要的信息。本例的 I/O 图示如下。



3. 用例

第三步是自己动手或者使用计算器，用一组数据来解决问题。这个步骤非常重要，即使对于很简单的问题也不应该跳过，这是你了解问题解决细节的步骤。如果你不能使用一组简单的数据来计算并得到输出（不论是手工还是用计算器计算），那么你将无法进行下一步；你应当重新思考问题，也许还需要查阅一些相关的参考文献。对于本例，我们给出的用例如下：

令点 p1 和 p2 的坐标如下：

$$p1=(1,5);\ p2=(4,7)$$

我们希望得到两点间的距离，即图 1.5 中直角三角形的斜边。使用 Pythagorean（毕达哥拉斯）定理，我们可以使用下面的公式计算距离：

$$\begin{aligned}\text{距离} &= \sqrt{(side_1)^2+(side_2)^2}\\ &= \sqrt{(4-1)^2+(7-5)^2}\\ &= \sqrt{13}\\ &= 3.61\end{aligned}$$

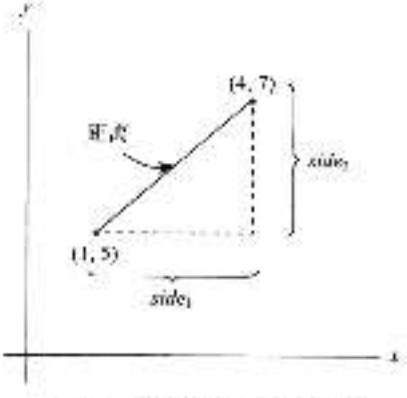


图 1.5 两点之间的直线距离

4. 算法设计

一旦你能使用一组简单的数据来解决问题，就可以开始设计解决问题的算法或者分步的提纲。对于类似之例的简单问题，详尽一步操作列出即是算法。这种分步的提纲将问题分解为简单的步骤，下面给出了计算和打印两点之间距离的分解提纲。

分解提纲

1）给两个点赋值。

2）计算由这两个点所形成的直角三角形的斜边长度。

3）计算两点间的距离，即直角三角形斜边长度。

4）打印出两点间的距离。

然后将分解提纲用 C++ 实现，这样我们就可以使用计算机来执行计算了。从下面的 C++ 程序实现中，可以看到程序中的命令与我们在用例中使用的步骤很类似。这些命令的具体含义将在第 2 章中解释。

```
/*-------------------------------------------------------------*/
/*  Program chapter 1                                          */
/*                                                             */
/*  This program computes the                                  */
/*  distance between two points.                               */
#include<iostream>//Required for cout
#include<cmath>//Required for sqrt()

using namespace std;

int main()
{
   //  Declare and initialize objects.
   double x1=1, y1=5, x2=4, y2=7,
          side1, side2, distance;

   //  Compute sides of a right triangle.
   side1 = x2 - x1;
   side2 = y2 - y1;
   distance = sqrt(side1*side1 + side2*side2);

   //  Print distance.
   cout << "The distance between the two points is"
        << distance << endl;

   //  Exit program.
   return 0;
}
/*-------------------------------------------------------------*/
```

5. 测试

我们解决问题的最后一个步骤就是对解决方案进行测试。首先我们应当使用用例中的数据来测试解决方案，因为我们已经使用它进行过计算。在本例中的C++解决方案运行后，计算机输出如下：

The distance between the points is 3.60555

上面的输出与手工计算的结果一致。如果C++解决方案与用例的手工解决方案不一致，那么应当对两个解决方案都进行检查以找出问题所在。当用例手工方案与C++方案匹配后，我们应当使用其他的数据集合进行测试，以保证解决方案在其他情况下也适用。

后续其他各章中解决应用问题都将会用到这里讲的步骤。

本章小结

本章简要介绍了数字计算机的发展历史，为了展现不同工程应用之间的差异和计算机对工程的影响，还介绍了近代的一些杰出工程成就。我们还讨论了成为一个成功的工程师所必需的非技术性技巧。因为大部分工程问题的解决都将使用计算机完成，我们从计算机系统软、硬件以及学习新技术的重要性上对计算机系统进行了介绍。我们还讨论了数值和数据表示，同时给出了解决问题的五个步骤，这些步骤如下：

1）清晰地描述问题。

2）描述问题的输入和输出信息。

3）使用一组简单的数据来解决问题。

4）设计解决方案并将其由计算机程序实现。

5）使用广泛的数据来测试解决方案。

当我们设计问题的解决方案时，将一直使用上述步骤。

关键术语

algorithm（算法）
Arithmetic Logic Unit（ALU，算术逻辑单元）
assembler（汇编器）
assembly language（汇编语言）
binary（二进制）
binary string（二进制串）
bug（错误）
carry bit（进位）
Central Processing Unit（CPU，中央处理单元）
compiler（编译器）
complement（补码）
computer（计算机）
database management（数据库管理）
debug（调试）
debugger（调试器）
decomposition outline（分解提纲）
electronic copy（电子拷贝）
execution（执行）
graphics tool（图形工具）
hardware（硬件）
high-level language（高级语言）
I/O diagram（I/O 图）
least significant digit（最低有效位）
linking/loading（链接/载入）
logic error（逻辑错误）
machine language（机器语言）
memory（内存）
microprocessor（微处理器）
most significant digit（最高有效位）
number systems network（数字系统网络）
object program（目标程序）
objects（对象）
operating system（操作系统）
parse error（解析错误）
Personal Computer（PC，个人计算机）
problem-solving process（问题求解过程）
processor（处理器）
program（程序）
real-time program（实时程序）
software（软件）
source program（源程序）
spreadsheet（电子表格）
syntax（语法）
word processor（字处理程序）
workstation（工作站）

习题

判断题

1. CPU 是由 ALU、内存和处理器组成的。
2. 链接和载入是为了让目标程序执行的准备步骤。
3. 算法是逐步描述问题解决方案的，但是计算机程序则只需要一步就可以解决。
4. 计算机程序是算法的实现。

多选题

从给定的选项中选出合适或正确的答案来完成每个语句。

5. 指令和数据被存放在（　　）中。

(a) 算术逻辑单元（ALU）　　(b) 控制单元（处理器）
(c) 中央处理单元（CPU）　　(d) 内存
(e) 键盘

6. 操作系统是（　　）。
   (a) 由用户设计的软件　　(b) 用户与硬件之间的良好接口
   (c) 允许我们完成常用操作的一组工具　　(d) 一组软件工具
7. 源代码是（　　）。
   (a) 编译器操作的结果　　(b) 从处理器获得信息的过程
   (c) 用计算机语言来解决特定问题的指令集合　　(d) 存储在计算机内存里的数据
   (e) 通过键盘输入的值
8. 算法是（　　）。
   (a) 解决特定问题的分步解决方案　　(b) 一组计算机可以理解的指令集合
   (c) 可以由文本材料写出来的代码　　(d) 逐步的优化
   (e) 一组表示问题解决方案的数学公式
9. 目标代码是（　　）。
   (a) 编译器对元代码的操作结果　　(b) 从处理器获取信息的过程
   (c) 一个计算机程序　　(d) 一个列出解决特定问题所需命令的过程
   (e) 链接和载入过程的结果
10. 将下面的二进制数转换成八进制表示。
    (a) 110110101　　(b) 1111010011011111
    (c) 1010111001　　(d) 1000000001
    (e) 1111111
11. 将下面的二进制数转换成十六进制表示。
    (a) 100010101011　　(b) 1110011111110101
    (c) 1010111001　　(d) 1000000000000001
    (e) 111111111111
12. 将下面的十进制数转换成八进制表示。
    (a) 1292　　(b) 607
    (c) 9350　　(d) 1000010
    (e) 1111111
13. 将下面的数转换成十进制表示。
    (a) 11010101_2　　(b) 4762_8
    (c) $30AF2_{16}$　　(d) 4103_5
    (e) 1111111_4

附加题

通过下面的作业，你可以学到有关本章某个主题的更多知识，同时也是一个提升自己书面交流能力的机会。（说不定你的教授还会选择其中一个题目作为课堂口头报告。）每篇报告都至少要有两篇以上的参考文献，因此你还要学习如何使用图书馆的计算机来查找引用信息。使用字处理软件来准备你的报告，如果你还不知道如何使用字处理软件，请向别人求助或者访问校园网站上的相关资源链接。

14. 选择下面的杰出工程成就之一，写一篇简短的报告。
    - 分析机
    - 全球定位系统
    - 复合材料
    - 应用卫星
    - 火星轨道相机
    - CAD/CAM

- 光学纤维
- CAT 扫描

在互联网上和本地图书馆寻找参考资料将是一个好的开始。

15. 选择下面的工程挑战之一，写一篇简短的报告。
    - 天气预报和全球气候变化
    - 计算机语音识别
    - 人类基因组映射
    - 车辆性能改进

    在 NASA 的网站（www.nasa.gov）上寻找参考资料将是一个好的开始。
16. 写一篇关于未在本章列出的杰出工程成就的简短报告。参考《科学美国人》杂志的历史期刊和 Web，可以找到关于最近成就的一些好想法。

第 2 章

Engineering Problem Solving with C++, 3e

简单的 C++ 程序

工程挑战：汽车性能

喷气式引擎由于采用了高效推进器，动力性很强，而涡轮旋桨引擎则兼备高动力性和可靠性特点。涡轮引擎采用气体涡轮来驱动推进器。然而在初期，涡轮引擎的应用范围只限于小型飞机领域，因为涡轮引擎的最大飞行速度不超过 500 英里 / 小时。随着新材料、叶片形状和推进器的发展，增加了涡轮引擎推进速度和效率，对于这种引擎的需求也增加了。

教学目标

在本章中，我们讨论的问题解决方案中包括：

- 简单的算术计算
- 用户通过键盘提供的信息
- 打印在屏幕上的信息
- 程序员自定义数据类型

2.1 程序结构

在本节中，我们先分析一个具体 C++ 程序的结构，然后给出 C++ 程序的通用结构。下面的程序是我们在第 1 章中介绍过的程序，它计算两点之间的距离并把距离显示出来。

```
/*-------------------------------------------------------------*/
/*  Program chapter2_1                                         */
/*                                                             */
/*  This program computes the                                  */
/*  distance between two points.                               */

#include <iostream>  //  Required for cout, endl
#include <cmath>     //  Required for sqrt()

using namespace std;

int main()
{
   //  Declare and initialize objects.
   double x1(1), y1(5), x2(4), y2(7),
          side1, side2, distance;

   //  Compute sides of a right triangle.
   side1 = x2 - x1;
   side2 = y2 - y1;
   distance = sqrt(side1*side1 + side2*side2);

   //  Print distance.
   cout << "The distance between the two points is "
        << distance << endl;
```

```
    //  Exit program.
    return 1;
}
/*-------------------------------------------------------------*/
```

我们先简要地介绍这个程序中的语句，在本章的后续小节中将会对每个语句进行详细介绍。

程序中的前五行是程序的注释（comment），其中包含了程序名（chapter1_1）和程序完成的功能。

```
/*-------------------------------------------------------------*/
/*      Program chapter1_1                                     */
/*                                                             */
/*      This program computes the                              */
/*      distance between two points.                           */
```

注释以“/*”开头，以“*/”结束，对于单行的注释，也可以使用“//”开头，从“//”开始到本行结束的部分都将成为注释部分。注释可以独占一行，也可以与程序语句在同一行中；使用“/*”形式的注释则可以形成包含多行的注释。

示例中的每行注释都是单独的注释行，因为每行都是以“/*”开始、以“*/”结束的。虽然注释不是必须的，但好的编码风格要求我们在程序中使用注释以增强程序的可读性，帮助说明计算过程。在程序代码中，我们通常在开头的注释中写出程序命名，并描述程序的主要功能；而在程序的其他部分也会包含对于程序语句的解释性注释。C++允许程序语句和注释在一行的任意地方开始。

预处理指令（preprocessor directive）给出了在程序编译之前由预处理器处理的指令。最常见的预处理指令就是在程序中包含附加的语句，这种指令以#include开头，其后则为包含附加语句的文件名称。这个程序中包括了下面两条预处理指令：

```
#include <iostream>
#include <cmath>
```

这两条指令表明在编译程序之前，上述语句指定了在编译程序之前，相关的声明应该被文件iostream和cmath中的声明语句替换掉。“<”和“>”符号表示所包含的文件在标准C++库中，标准C++库包含在ANSI C++编译器附带的文件中。文件iostream包含了程序中使用的输出语句的相关信息，文件cmath包含了程序中用于计算平方根的相关内容。预处理指令通常放在描述程序功能的注释之后。下面一条语句：

```
using namespace std;
```

称作using指令（using directive）。这条using指令告诉编译器使用在命名空间std中声明的库文件。命名空间是新的ANSI标准C++中的一部分。一些旧的编译器不支持命名空间。如果你的编译器不识别using指令，那么则需要忽略掉这条语句，并将预处理指令中的文件名替换成下面的形式：

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

旧的编译器将.h作为大部分头文件的后缀名。你会看到在C程序以及C++程序中包含的math.h文件，因为math.h是C库头文件中C风格的名字。新的编译器区分C和C++的头文件名。头文件math.h的C++名字是cmath。一般来说，C库中头文件的C++名字是通过C名字变化而来的：即去掉.h的后缀，而在文件名的开头加上字母c。本书我们将使用新

的 ANSI 标准 C++ 名字。

每个 C++ 程序都包含一个名为 main 的函数，它由一个代码块（block of code）定义。在 C++ 中，代码块由一对大括号“{}”定义。关键字 int 表示函数返回一个整型值给操作系统。因为 main 是一个函数，所以在 main 后面必须跟一对圆括号“()”。为了简单说明定义函数体的代码块，我们将大括号单独放一行。因此，在预处理指令之后的下面两行就表示了 main 函数的开始：

```
int main()
{
```

主函数里包括了两种类型的命令：声明（declaration）和语句（statement）。声明定义标识符并分配内存，因此声明必须在任何引用了相关标识符的语句之前。在声明中可以给存储在内存中的标识符赋初值，也可以不赋初值。下面的程序中在声明语句之前给出了相关的注释：

```
// Declare and initialize objects.
double x1(1), y1(5), x2(4), y2(7),
       side1, side2, distance;
```

这段声明表示程序将使用 7 个名字分别为 x1、y1、x2、side1、side2 和 distance 的对象。术语 double 表示对象的类型是 double 类型，这是 C++ 的内建数据类型的一种。每个对象都将存储一个双精度的浮点数，这些对象可以存储诸如 12.5 和 −0.0005 一样包含若干有效位的非整型值。此外，语句还指明 x1 应当被初始化（initialized）(赋予它一个初始值）为 1，y1 应当被初始为 5，x2 被初始化为 4，y2 被初始化为 7。side1、side2 和 distance 没有指定初始值，且它们的初始值不应当被假定为 0。因为声明放在一行太长，所以我们将它分为两行；第二行行首的缩进表示它是前一行的延续。缩进只是增加可读性的一种方式，不是必需的。分号是声明语句的结束。

示例程序中执行操作的语句如下所示：

```
// Compute sides of a right triangle
side1 = x2 - x1;
side2 = y2 - y1;
distance = sqrt(side1*side1 + side2*side2);

//  Print distance
cout << "The distance between the two points is "
     << distance << endl;
```

这些语句计算了由两点所形成的直角三角形两边的长度（参见图 1.5），并且计算了直角三角形的斜边长度。这些赋值语句的语法细节将在后续章节中讨论。将距离计算出来后，使用 cout 语句把结果打印出来。由于打印语句放在一行显得太长，我们将它分为两行；这里第二行的缩进同样表示它是第一行的延续。分号是输出语句的结束。附加的注释用于解释计算过程和输出语句。注意，所有的 C++ 语句都要用分号结尾。

为了结束程序执行，并将控制权返回给操作系统，我们使用了“return(0);”语句。

```
//  Exit program
return (0);
```

这条语句向操作系统返回了 0，返回值为 0 表示执行过程成功结束。

main 函数体以右括号“}”结束，另一行注释用以描述 main 函数的结束。

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

注意，我们在程序中包含了空行，用以分隔不同的功能部分。这些空行使得程序更易读，也更容易修改。在主函数中的语句使用了缩进，以展现程序的结构。这些空白提供了一致的风格，使我们的程序更易读。

既然已经仔细分析了第1章中的C++程序，我们可以从它的结构得到C++程序的通常形式：

```
#include <iostream>
using namespace std;
int main()
{
   declarations;
   statements;
}
```

本章和后续章节涉及的程序中都体现了这种结构。

修改

1. 使用C++编译器的编辑器或者一个字处理程序[⊖]创建一个包含本节所讨论的示例程序的文件，然后编译并执行程序。你应当会得到下面的输出：

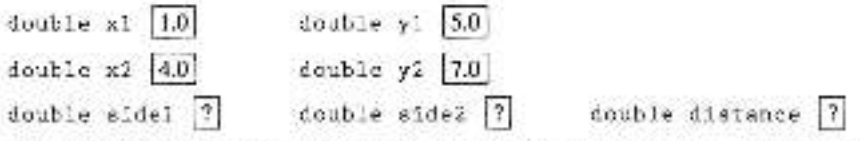
```
The distance between the two points is 5.61
```

2. 改变给定的两点坐标，将其设为（-1, 6）和（2, 4）。用新的值重新运行程序，距离改变了吗？请说明结果。
3. 将两点坐标改为（1, 0）和（5, 7）。用计算器验证程序的结果。
4. 将两点坐标都改为（2, 4）。程序运行结果正确吗？

2.2 常量和变量

常量和变量代表我们在程序中使用的对象。字面常量（constant）是我们用于构成C++语句的，被赋予如2、3.1416、-1.5、'a'或'hello'等值的对象，这些对象的值不能被改变。变量（variable）是与标识符或名称相关联的存储单元。标识符（identifier）用来引用存储在内存单元中的值。可以做一个类比，将内存单元和其相应的标识符看作是一个与个人姓名相关联的邮箱；内存单元（或邮箱）可能包含了一个对象。下面的实例展示了程序chapter1_1里声明语句完成后各变量、标识符、初始值的情况。

```
double x1(1), y1(5), x2(4), y2(7),
       side1, side2, distance;
```



没有被初始化的变量值是不确定的，因此使用“?”来表示；有时候这些不确定的值被称为垃圾（garbage）值，因为它们的值是不可预测的。像这样表示对象以及其标识符、值和数据类型的图称为内存快照（memory snapshot）。内存快照展示了在程序执行的某个时刻内存的内容。前述内存快照展示的是在执行了声明语句之后的对象和对象的内容。被声明为double的对象在内存中将使用标准的IEEE格式存储。为了简单起见，我们只给出了变量的值和它的数据类型，但是我们应当记住它是浮点型的值，即使初始化时赋予了一个整型常量，但它在内存中仍使用IEEE浮点格式存储。

⊖ 如果你使用字处理程序来编辑源文件，请确保将其以纯文本文件保存。

我们会频繁使用内存快照来展示对象在语句执行前后的内容变化，以展示语句执行的影响。选择一个合法的标识符的规则如下：

- 标识符必须以字母或下划线开头。
- 标识符中的字母可以是大写或小写。
- 标识符中可以包含数字，但不能以数字开头。
- 标识符长度可以任意，但对每个标识符而言前 31 个字符必须是唯一的。

C++ 是区分大小写（case-sensitive）的，这意味着大写字母不同于小写字母，因此 Side1、SIDE1 和 side1 代表着三个不同的对象。C++ 还包含了对于 C++ 编译器有着特定含义的关键字，你不能将这些关键字用作标识符，表 2.1 列出了这些关键字。

表 2.1 关键字

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

合法标识符的例子如 distance、x1、xSum、averageMeasurement、initalTime 等。不合法的标识符例子如 1x（以数字开头）、switch（这是一个关键字）、$sum（包含了非法字符 $）、rate%（包含了非法字符 %）。

应当小心选择标识符的名字，以便让它能够反映对象的内容。如果可能的话，标识符的名字还应当体现出对象的计量单位。例如，如果一个对象用来表示以华氏为单位的温度，可以使用 tempF 或 degreesF。如果对象代表一个角度，则可以使用 thetaRad 来表明这个角度是以弧度表示的，或使用 thetaDeg 表明是用角度表示的。

在任何代码块的声明语句中不仅要包含我们计划在程序中使用的对象标识符，还应该指明这些对象的数据类型。C++ 是一种强类型（strongly typed）的程序语言。这意味着每个标识符在使用前都必须先声明。在讨论科学记数法之后我们将讨论数据类型。

练习

下面给出的名字中哪些是合法的？如果某个名字是不合法的，请说明理由，并给出你的修改建议。

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. xSy

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 科学记数法

浮点型值可以表示为整数与非整数值的组合，如2.5、-0.004和15.0。以科学记数法表示的浮点值会被重写为一个定点数与10的幂乘积，这个定点数的绝对值大于等于1.0且小于10.0。例如，在科学记数法中，25.6写作2.56×10^{1}，0.004写作-4.0×10^{-3}，而1.5则写作1.5×10^{0}。在指数表示法（exponential notation）中，字母e用于分隔定点数和10的幂。因此，在指数表示法中，25.6写作2.56e1，0.004写作4.0e-3，1.5写作1.5e0。

计算机支持的定点数的有效位数决定了数的精度（precision），支持的指数位数决定了数的范围（range）。因此，具有两位有效位数和指数范围在-8～7之间的数值包括了如2.3×10^{7}（230 000）和5.9×10^{-8}（0.000 000 059）这样的数。这样的精度和指数范围对于我们在工程问题中所使用的许多类型而言都是不够的。例如，从火星到太阳的距离（以英里计算），使用7位有效位表示为141 517 510或$1.415\ 175\ 1\times10^{8}$，为了表示这个值，我们至少需要7位有效位和包括8在内的指数范围。

练习

将练习1～6中的数用科学记数法表示出来。指出表示每个值需要的有效位数。

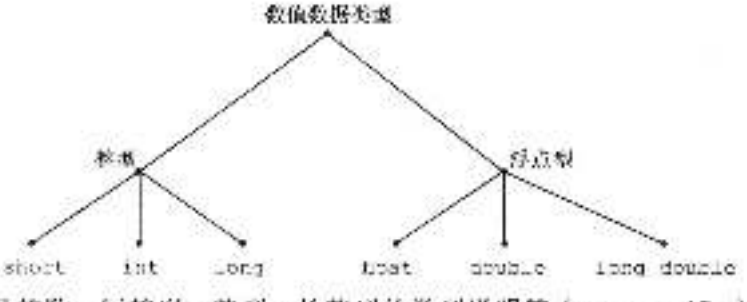
1. 35.004	2. 0.000 42	3. -50 000
4. 3.157 23	5. -0.099 97	6. 10 000 028

将练习7～12中的数值表示为浮点形式。

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

数值数据类型用于指定对象所包含的内容为数值。在C++中，内建的数值类型包括整型和浮点型。下图给出了内建的数值数据类型，后面将分别对它们进行讨论。



对于有符号整数，短整型、整型、长整型的类型说明符（type specifier）分别是short、int、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。在本章的前一节中，我们给出了一个用于确定你所使用的系统的数值数据类型范围的程序。在大部分系统上，短整型的范围从-32 768到32 767，整型和长整型的范围通常在-2 147 483 648到2 147 483 647。（像32 767和2 147 483 647这样的范围限制值是对应的二进制限制值的十进制表示。）C++还允

许在整型说明符前加上 unsigned（无符号的）限定符。一个 unsigned 整数只能表示正数。有符号正数和无符号整数所能表示的数的个数相同，但是范围不同。例如，无符号短整型的表示范围为 0 ～ 65 535，而有符号整型的范围则在 -32 768 ～ 32 767 之间，这两种数据类型都能表示 65 536 个数值。

声明： 一个类型声明语句定义了一个新的标识符并为之分配内存。在定义标识符的同时可以为其指定初始值。在声明类型时也可以添加限定符。

语法

```
[限定符] 类型说明符 标识符 [=初始值];
[限定符] 类型说明符 标识符 (初始值);
```

示例

```
double x1, y1(0);
int counter=0;
const int MIN_SIZE=0;
bool error(false);
char ch=',';
```

浮点值的类型说明符分别是 float（单精度）、double（双精度）和 long double（扩展精度）。下面的语句来自 chapter1_1，定义了 7 个双精度对象。

```
double x1(1), y1(3), x2(4), y2(7),
       side1, side2, distance;
```

几种浮点类型之间的不同主要在于精度和所表示的数值范围，它们的精度和数值范围是与系统相关的。表 2.2 给出了 Microsoft Visual C++ 编译器中整型和浮点型的精度和数值范围。在 2.9 节中给出的程序将帮助你得到所使用的计算机系统上相关数据类型的信息。在大多数系统上，double 类型所能表示的十进制有效位是 float 类型的两倍。此外，double 类型的值所能表示的范围比 float 类型的值更大。而 long double 在有效位数和范围上都要更大，但这也是与系统相关的。形如 2.3 这样的浮点常量都被当做是 double 类型的常量。为了指明 float 常量或者 long double 常量，应当在常量后面加上字母 F 或 L。因此，2.3F 和 2.3L 分别表示 float 常量和 long double 常量。

表 2.2 数据类型范围示例[1]

数据类型		范围
整型	short	最大值 = 32767
	int	最大值 = 2147483647
	long	最大值 = 2147483647
浮点型	float	6 位精度
		最大指数 = 38
		最大值 = 3.402823e + 38
	double	15 位精度
		最大指数 = 308
		最大值 = 1.797693e + 308
	long double	15 位精度
		最大指数 = 308
		最大值 = 1.797693e + 308

① Microsoft Visual C++ 6.0 编译器。

2.2.3 布尔数据类型

布尔数据类型（boolean data type）是使用数学家 George Boole 的名字命名的，这种数据类型只能表示两个值：真和假。在 C++ 中，值 0 视作假，将其他非 0 值都视作真。C++ 中有两个预定义的布尔型常量：true 和 false。布尔型对象可以使用类型说明符 bool 来定义。下面的例子说明了布尔对象和常量的用法：

```
bool error(false), status(true);
cout << error << endl << status;
```

这段程序的输出是

```
0
1
```

布尔对象在控制程序流程和条件标识时非常有用。

2.2.4 字符数据类型

使用C++处理字符数据是很简单的，但是要有效地使用字符（character）数据，我们还需要对它们在计算机内存中的表示有更多的了解。回想一下，在计算机中所有信息的内部表示都是二进制位的序列。每个字符都对应着一个二进制码（binary code）的值。在下面的讨论中，我们假定是用ANSI码来表示字符。

表2.3中包含了一些字符的ANSI形式，以及对应的二进制数的数值。字符'a'的二进制值是1100001，这等于整数97。有128个字符可以使用ANSI码来表示。完整的ANSI码表在附录B中给出。

表2.3 ANSI码示例

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

字符常量要使用单引号表示，如'A'、'b'、'3'。字符对象的类型说明符是char。当字符以一个二进制值存在内存中后，这个二进制值就可以当做一个字符或者一个整数来解释，就像表2.3中所示。但是，需要注意的是一个字符形式的数字（如'3'）的ANSI表示并不等于对应数字（整数3）的二进制表示。

在表2.3中，我们可以看到字符形式的数字'3'的ANSI二进制表示是0110011，它等于整数值51的二进制表示。因此，与字符形式的数字进行计算的结果并不等于与纯粹二进制数字形式的数进行计算的结果。下面的程序说明了字符数据的特点。

```
/*-------------------------------------------------------------------*/
/* Program chapter2_1                                                */
/*                                                                   */
/* This program prints a char and an int                             */

#include<iostream> // Required for cout
using namespace std;

int main()
{
//  Declare and initialize objects.
char ch ('3');
int i(3);

//  Print both values.
cout << "value of ch: " << ch << " value of i: " << i << endl;

//  Assign character to integer
i = ch;

//  Print both values.
cout << "value of ch: " << ch << " value of i: " << i << endl;
```

```
// Exit program
return( 0);
}
//-----------------------------------
```

该程序的输出如下所示：

```
value of ch: 3; value of i: 3
value of ch: 3; value of i: 51
```

2.2.5 字符串数据

字符串（string）常量是使用双引号包含起来的字符序列，如"sensor"、"F18"、"Jane Doe"等。在C++中字符串变量可以是使用字符（char）数组的C风格字符串（C-style string），也可以使用string类来定义字符串对象（string object）。我们将在第7章对这两种表示方式进行详细讨论。本节中我们对string类进行简单介绍。

在C++编译器中没有名为string的内建数据类型，但是在标准C++库文件string中提供了一个string类的定义。为了使用这个string类，在程序中必须包含下面的预处理指令：

```
#include<string>
```

下面的程序给出了C++中预定义的string类的用法。

```
/*-------------------------------------------------------------*/
/* Program chapter2_2                                          */
/*                                                             */
/* This program prints a greeting                              */
/* using the string class.                                     */
/*-------------------------------------------------------------*/

#include <iostream>  // Required for cout
#include <string>    // Required for string
using namespace std;

int main()
{
// Declare and initialize two string objects.
string salutation("Hello"), name("Jane Doe");

// Output greeting.
cout << salutation << " " << name << "!" << endl;

// Exit program.
return(0);
}
```

该程序的输出如下所示：

```
Hello Jane Doe!
```

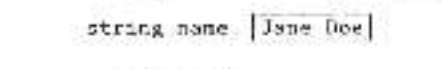
程序Chapter2_2的内存快照如下所示：

```
string salutation("Hello"),name("Jane Doe");
string salutation  Hello
string name  Jane Doe
```

程序中string对象salutation的长度为5，string对象name的长度为8。在程序执行过程中string对象的长度可以改变。例如，若将赋值语句

```
salutation = "Goodbye";
```

添加到 chapter2_2 中，那么内存快照将会被修改，salutation 的长度将变为 7。



2.2.6 符号常量

符号常量（symbolic constant）使用 const 限定符来定义一个程序的常量，如 π 或者重力加速度等，那适合用符号常量来表示。例如，使用下面的语句将一个值赋给符号常量 PI：

```
const double PI = acos(-1.0);
```

当 acos 函数的参数赋值为 -1 时，它将返回一个具有 15 位有效数字的 π 的近似值。需要使用 π 值的语句就可以像下面的语句一样来使用符号常量 PI：

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

这条语句将计算圆的面积。

符号常量通常使用全大写来表示（使用 PI 而不是 pi），以表明它们是符号常量。当然，所定义的符号常量也应该容易记忆。一旦使用 const 限定符定义并初始化符号常量后，它的值在程序中就再也不能改变了。

练习

给出下面这些需要定义成符号常量的常数值的定义语句。

1. 光速，$c = 2.997\,92 \times 10^8$ m/s
2. 一个电子的电量，$e = 1.602\,177 \times 10^{-19}$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，$r = 1.74 \times 10^3$m
8. 长度单位，UnitLength = 'm'
9. 时间单位，UnitTime = 's'

2.3 C++ 类

C++ 支持定义新的数据类型，这些新定义的数据类型通常称作程序员自定义数据类型（programmer-defined data type）。在前一节中，我们看到了使用包括 int、double、char 在内的内建数据类型的例子，还讨论了在 chapter2_2 中使用的预定义数据类型 string。回想一下在标准 C++ 库中所包含的预定义数据类型以及使用这些预定义数据类型时所需要包含的编译器指令。内建数据类型不需要包括任何 include 指令。一旦定义好一个自定义数据类型，它使用起来就与使用预定义的数据类型一样。本节我们将来看一看如何构建一个自定义数据类型。

在介绍构建自定义数据类型时，我们将定义一个新的数据类型用来表示平面上的一个点。这个平面上的点被定义为一个形如 (x, y) 的对象。这一对值通常是浮点型，因此我们将使用内建数据类型 double 来作为我们自定义数据类型的内部表示。自定义数据类型通常由两部分组成：类声明（class declaration）和类实现（class implementation）。通常类声明放在以类的名字命名、以 .h 为后缀的文件中，而类实现则放在以类的名字命名、以 .cpp 为后缀的文件中。应用程序要使用自定义数据类型，必须在源代码中包含类的声明文件。

2.3.1 类声明

类声明以关键字 class 开始，class 后跟用来表示类名的标识符。类声明部分是一个代码块，以分号结束。声明部分包括数据成员（data member）的声明语句和类方法（class method）的声明语句。关键字 public、private、protected 用于控制数据成员和类方法的访问权限。对数据成员和成员方法的访问控制是类的重要特征，我们将在后面的章节中进行详细讨论。为了说明类声明的语法，我们将用一个名为 Point 的类声明作为示例。

```
/*-------------------------------------------------------------*/
/* Point class chapter2_3                                      */
/* Filename: Point.h                                           */
class Point
{
// Type declaration statements
// Data members.
private:
double xCoord, yCoord; //Class attributes
public:
//Declaration statements for class methods
//Constructors for Point class
Point(); //default constructor
Point(double x, double y); //parameterized constructor
};
/*-------------------------------------------------------------*/
```

注意，声明部分需要以分号结束。

类方法定义了能对类对象进行的操作。构造函数（constructor）是一个用来完成类型声明操作的特殊方法。回想一下我们声明和初始化一个类型为 double 的对象时的类型声明语句，如下所示：

```
double xCoord, yCoord(0.0);
memory snapshot:
double xCoord ?
double yCoord 0.0
```

在 Point 类声明中声明的构造函数提供了 Point 类对象的类型声明语句，如下所示：

```
Point p1, p2(1.3, 1.0);
memory snapshot:
Point p1  -> ? double xCoord
             ? double yCoord
Point p2  -> 1.0 double xCoord
             1.0 double yCoord
```

2.3.2 类实现

为了完成我们的 Point 类的定义，还需要写出类实现文件。类实现文件中必须包含在类声明中所定义的每个方法的实现代码块。每个实现代码块都必须以类名开头，类名后跟域限定操作符（::）和方法名。我们的 Point 类的实现文件内容如下所示。注意，在实现文件中包含了编译器指令 #include "Point.h"。因为 Point.h 不是 C++ 标准库的一部分，因此使用双引

号来包含声明文件名。

```
/*-------------------------------------------------------*/
/* Class implementation for Point                        */
/* filename: Point.cpp                                   */
#include "Point.h"        //Required for Point
#include <iostream>       //Required for cout
using namespace std;

//Parameterized constructor
Point::Point(double x, double y)

{
  //Input parameters x,y
  cout << " Constructing Point object, parameterized: \n" ;
  cout << " input parameters: "  << x << " ,"  << y << endl;
  xCoord = x;
  yCoord = y;

}
//Default constructor
Point::Point()
{
   cout << " Constructing Point object, default: \n" ;
   cout << " initializing to zero"  << endl;
   xCoord = 0.0;
   yCoord = 0.0;
}
```

带参数的构造函数以 Point::Point (double x, double y) 开头。带参数的构造函数提供了用以初始化数据成员的参数。在带参数的 Point 构造函数中提供了两个参数 x 和 y，它们的类型均为 double。在实现代码块中，参数的值被赋给类的数据成员。为了调试和说明，在实现代码块中我们以两个 cout 语句开始。这些语句在 Point 类被完全设计好并实现之后去除。在 cout 语句之后，x 的值被赋给 xCoord，y 的值被赋给 yCoord。

默认的构造函数以 Point::Point() 开头。因为在开头没有提供参数，所以在实现代码块中给 xCoord 和 yCoord 都赋值为常量 0.0。注意，我们可以选择任意常量作为初始值。下面的程序声明了两个 Point 类型的对象。

```
/*-------------------------------------------------------*/
/* Program chapter2_3                                    */
/*                                                       */
/* This program illustrates the use of the               */
/* programmer-defined data type Point                    */

#include <iostream>   //Required for cout
#include "Point.h"    //Required for Point
using namespace std;

int main()
{
  //Declare and initialize objects.
  cout << " In main, declare p1..."  << endl;
  Point p1;
  cout << " \nIn main, declare p2..."  << endl;
  Point p2(1.5, -4.7);
  cout << " \nIn main, declare ORIGIN..."  << endl;
  const Point ORIGIN(0.0, 0.0);
  return 0;
```

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

这个程序的输出如下：

```
In main, declare p1...
Constructing Point object, default:
initializing to zero

In main, declare p2...
Constructing Point object, parameterized:
input parameters: 1.5,-4.7

In main, declare ORIGIN...
Constructing Point object, parameterized:
input parameters: 0,0
```

在学习 C++ 新的特性后，我们将继续为 Point 类添加功能。

类定义：程序员自定义类由两部分组成：类声明和类实现

语法：类声明

```
// 文件名：className.h
class className
{
    访问限定符：
        属性声明
    访问限定符：
        方法声明
};
```

语法：类实现

```
// 文件名：className.cpp
#include"className.h"
成员函数定义
```

示例：类定义

```
class Point
{
  private:
    double xCoord;
    double yCoord;

  public:
  Point(double x, double y);
};
```

示例：类实现

```
#include "Point.h"

Point::Point (double x,double y)
{
  xCoord = x;
  YCoord = Y;
}
```

用法

```
#include "Point.h"
...
int main()
{
  Point p1(1.5, 2.7);
...
```

2.4 C++ 操作符

为了在程序中完成诸如加法、乘法、对象比较等操作，我们需要使用 C++ 中为内建数据类型提供的特定操作符（operator）。本节我们将看到这些操作符中的几种。

2.4.1 赋值操作符

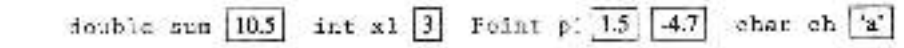
赋值语句（assignment statement）使用赋值操作符（assignment operator）“=”将结果存入标识符所对应的内存中。赋值语句的通常形式如下：

标识符 = 表达式

这里的表达式可以是一个常量、一个对象或者某个操作的结果。下面的两组语句声明了对象 sum，x1，p1 和 ch，并分别对它们进行了赋值。

```
//Set One
double sum(10.5);
int x1(3);
Point p1(1.5, -4.7);
char ch('a');

//Set Two
double sum;
int x1;
Point p1, p2(1.5, -4.7);
char ch;

sum = 10.5;
x1 = 3;
p1 = p2;
ch = 'a';
```

在其中任一组语句执行后，sum 的值为 10.5，x1 的值为 3，p1 的值为（1.5，-4.7），ch 的值为 'a'，如下面的内存快照所示：



第一组语句在类型声明语句中声明对象的同时对各个对象进行了初始化；第二组语句可以放在程序中任何地方，也可以用于改变（而不是初始化）已经声明了的对象的值。注意，编译器为 Point 类提供了赋值操作符。赋值操作符将操作符右边对象的数据成员按位拷贝的方式复制到操作符左边的对象对应的数据成员中。

C++ 中允许连续赋值，在下面的语句中，将 0 赋给了对象 x、y 和 z。

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

在本节结尾将进一步讨论连续赋值。

赋值语句：赋值语句使用赋值操作符将表达式的值存储到一个由标识符名字所确定的内存空间中。表达式可以是常量、变量或者一个最终计算结果类型与标识符类型兼容的表达式。

语法

```
标识符 = 表达式;
```

示例

```
x1 = y1;
counter = 0;
counter = counter + 1;
```

不要将赋值操作符与等号混为一谈。赋值语句应该读作“将……值赋给……”，因此语句 rate = stateTax 读作：将 stateTax 的值赋给 rate。如果 stateTax 的值为 0.06，那么在语句执行后 rate 的值将为 0.06；而 stateTax 的值不改变。语句执行前后的内存快照如下所示：

执行前： double rate ? double stateTax 0.06

执行后： double rate 0.06 double stateTax 0.06

如果我们为某个对象所赋的值与对象本身的类型不同，那么在语句执行过程中将会发生类型转换。有时候这种转换会导致信息丢失。例如，考虑下面的声明和赋值语句：

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

因为a被声明为一个整数，它不能存储有非零小数的值。因此在这种情况下，执行语句后，内存快照如下所示：

```
int a 12
```

注意，12.8被截断了，而不是取整。

为了确定数值转换是否符合预期，我们使用下面的顺序（由高到低）进行判断。

```
high:   long double
        double
        float
        long integer
        integer
low:    short integer
```

如果一个值被赋给比它的次序更高的数据类型，那么不会发生信息丢失。但是如果赋给比它的次序低的数据类型，那么就会发生信息丢失。因此，把int类型常数赋给double类型浮点数不会有问题，而将double类型浮点数赋给整型数将会发生信息丢失或者得到不正确的结果。通常来说，只应该使用不会引起潜在转换问题的赋值。无符号整数没有在上表中列出，因为有关它的赋值在两个方向（赋值或被赋值）都可能发生错误。

2.4.2 算术操作符

赋值语句可用于将算术操作（arithmetic operation）的结果赋给某个对象，下面的语句计算了矩形的面积：

```
areaSquare = side*side;
```

这里的*操作符表示乘法。操作符+和-分别表示加法和减法，而/则用于除法。因此，下面的每个语句都是合法的三角形面积计算方式：

```
areaTriangle = 0.5*base*height;

areaTriangle = (base*height)/2.0;
```

第二条语句中的圆括号的使用不是必须的，只是为了增强可读性。

看看下面的赋值语句：

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

在代数中，这是不合法的，因为一个值不能等于它自身的值与非零值1的和。但是，在赋值语句中不应被读作“等于”；相反，这里应该读作将x的值加1后赋给x。实际上就是将x的值增加1。因此，如果在这条语句之前x的值为5，那么执行语句后x的值变为6。

C++中还包括了取余操作符（modules operator）%，它用于计算两个整数相除后得到的余数。例如，5%2等于1，6%3等于0，2%7等于2（2/7的商为0，余数为2）。如果a和b都是整数，那么a/b计算得到的是a除以b的商，a%b得到的是余数。因此，如果a等于9，b等于4，则a/b为2，a%b为1。如果a/b和a%b中b的值为0，或者a%b中a和b有一个为负，那么最终的结果与系统相关。

在确定一个数是否是另一个数的整数倍时，取余操作符是很有用的。例如，如果a%2等于0，那么a是偶数，否则是奇数。如果a%5等于0，那么a是5的倍数。在工程问题解决方案的开发中我们将经常用到取余操作符。

前面讨论的5种操作符（+、-、*、/、%）都是二元操作符（binary operator），即它们的操作都是对两个操作数进行的。C++中还包括一元操作符（unary operator），即对一个操作数进行操作的操作符。例如，+和-用于像-x这样的表达式中时就是一元操作符。

对两个类型相同的值进行二元操作最后得到的结果类型也与操作数类型相同。例如，如果a和b都是double类型的值，那么a/b的计算结果也是double类型的。类似地，如果a和b都是整数，那么a/b的结果也是整数。但是在整数除法中有时可能会得到非预期的结果，因为整数除法中得到的结果的小数部分都会被丢弃，得到的结果是被截断的结果（truncated result），而不是一个取整的结果。因此，5/3等于1，而3/6等于0。编译器没有为自定义类提供算术操作符，但是程序员可以将这些操作作为类方法来实现。

如果操作的两个值的类型不同，则称为混合操作（mixed operation）。在这样的操作进行前，低次序的数据类型先被提升为高次序的数据类型（如同赋值语句中所讨论的转换一样），这样操作就是对相同类型的数据进行了。例如，如果一个操作是在一个整数和一个浮点数间进行，那么在操作进行之前整数将先被提升为浮点型，最后的计算结果将是浮点型。

假设我们希望计算一组整数的平均数。如果这组整数的和与整数的个数分别被存储在整型对象sum和count中，那么正确计算平均数的代码应该类似于如下形式：

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

但是，两个整数相除的结果最后将被提升为double类型。因此，如果sum为18，count为5，那么赋给average的值是3.0，而不是3.6。为了正确计算出平均值，我们使用了类型转换操作符（cast operator），这是一种允许我们在进行下一步操作前将数值类型进行改变的一元操作符。在这个例子中，我们对sum应用了到double的转换：

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

在除法进行之前，sum的值先被提升为double类型，而在double类型值和整数之间的除法就是一个混合操作。因此count在计算前被提升为double类型，最后计算的结果作为double类型赋给average。如果sum的值为18，count的值为5，现在average的值则被正确地计算为3.6。类型转换操作符只影响计算过程中的值，并不改变存储在对象sum中的值。

练习

给出下面每条语句的计算结果，并画出内存模型情况。使用在2.3节中定义的Point类来回答练习5和6。

```
1. int a(27), b(5), c;
   ...
   c = bba;

3. int a;
   double b(6), c(18.6);
   ...
   a = c/b;

5. Point p1, p2( 5.2, 0.0);
```

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

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

6. double x(2.0), y(4.3);
   Point p1(x,y);
```

2.4.3 操作符的优先级

在含有多个算术操作符的表达式中，我们需要了解操作进行的顺序。表 2.4 中包含了算术操作符的优先级（precedence of arithmetic operator），这与代数中计算优先级相吻合。如果在表达式中使用了圆括号，那么在括号内的操作将最先进行。如果圆括号被查出现，那么最内层的圆括号中的操作最先进行。一元操作符优先于二元操作符 *、/、% 执行，二元操作符 + 和 - 的优先级最低。如果在表达式中含有若干个优先级相同的操作符时，计算顺序按照表 2.4 中所述规则进行。例如，在下面的表达式中：

表 2.4 算术操作符的优先级

优先级	操作符	结合性
1	圆括号：()	与最接近的结合
2	一元操作符： - -（类型）	从右向左
3	二元操作符： */%	从左向右
4	二元操作符： + -	从左向右

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

因为乘法和除法的优先级相同，而相关的操作对象关联顺序（对操作进行分组的顺序）是从左向右，因此这个表达式的计算顺序可以表示如下：

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

优先级顺序中并未指明 a*b 是否应该在（b/c）*d 之前进行，实际的计算顺序与系统实现相关，但是这不会影响最终结果。

算术表达式中的空格只是一种书写风格。有的人则在每个操作符左右都加入空格。我们只在二元加、减法操作符两边加入空格，因为它们是最后被计算的。可以选择加入空格的方式，但是应当保持风格一致。

假如我们想计算梯形的面积，并且已经声明了 4 个 double 类型的对象：base、height1、height2 和 area。我们还进一步假定这些对象都已经有值。那么一个正确的计算梯形面积的语句如下：

```
area = 0.5*base*(height1 + height2);
```

假如我们忽略表达式中的括号，那么语句如下：

```
area = 0.5*base*height1 + height2;
```

该语句的实际执行效果与下面的语句相同：

```
area = ((0.5*base)*height1) + height2;
```

这将得到一个错误的计算结果，并且我们不会得到任何错误提示。因此，再将算术表达式转

换成 C++ 表示时要十分小心。通常来说，在复杂的表达式中使用圆括号来指明操作顺序可以避免混淆，并确保表达式按照预想的方式执行。

你可能注意到没有用于完成如 x^4 这样的指数计算的操作符。在有关基本数学函数讨论的章节中我们将会看到用于执行指数计算的数学函数。当然，将数次方的计算可以通过连乘来实现，如 a^2 可以用 a*a 计算得到。

长表达式的计算应该被分解为多个语句来进行。如下面的等式：

$$f = \frac{x^3 - 2x^2 + x - 6.3}{x^2 - 0.050\,05x - 3.14}$$

如果我们想在一条语句中完成表达式的计算，那么这条语句将会变得很长而不易读：

```
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分为两行来写：

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

另一种解决方法则是分步来计算分子和分母：

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

为了计算得到正确的 f 值，对象 x、numerator、denominator 和 f 都必须是浮点类型的对象。

练习

在练习 1～3 中，给出用于计算指定值的 C++ 语句。假定表达式的标识符都已经被定义为 double 类型并赋予了合适的值。使用的重力加速度常量值为 $g = 9.80665\ m/s^2$。

1. 移动距离

$$\text{Distance} = s_0 + v_0 t + at^2$$

2. 绳子的拉力

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

3. 管道末端的液体压力

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

在练习 4～6 中，给出由 C++ 语句计算的数学等式。假定下面的符号常量都已经被定义，其中 G 的单位为 $m^3/(kg*s^2)$。

```
const double PI = acos(-1.0);
const double G = 6.6725e-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));
```

2.4.4 上溢和下溢

存储在内存中的数值都有一个允许的数值范围。当某次计算结果超出了允许的数值范围就会出错。例如，假定浮点值的指数范围在 -38 ~ 38 之间。这样的范围对于大部分计算都是足够的。但是有可能某个表达式的计算结果会超出这个范围。例如，假设我们执行了下面的语句：

```
x = 2.5e30;
y = 1.3e30;
z = x*y;
```

x 和 y 的值都在允许的范围之内，但是 z 的值是 2.5e60，这已经超出了范围。这种错误称为指数上溢（overflow），因为算术操作结果的指数太大而不能存储在分配给对象的内存中。发生指数上溢时的行为是与系统相关的。

指数下溢（underflow）的错误也是类似的，它是由于算术操作结果的指数太小而不能存储在分配给对象的内存中所导致的。在前述指数范围的假定前提下，下面的语句执行将会出现指数下溢：

```
x = 2.5e-30;
y = 1.3e30;
z = x/y;
```

同样，x 和 y 都在允许的范围内，但是计算出的 z 值为 2.5e-60。因为结果的指数小于所允许的最小指数范围，所以出现了指数下溢。同样，发生指数下溢时的行为也是与系统相关的。在某些系统上，发生指数下溢时，会将发生下溢的操作结果置为 0。

2.4.5 自增和自减操作符

C++ 语言中包含了用于对象增减的一元操作符，这些操作符不能对常量和表达式使用。自增操作符（increment operator）++ 和自减操作符（decrement operator）--，可以放在前缀（prefix）位置（标识符前），如 ++count，也可以放在后缀（postfix）位置（标识符后），如 count--。如果自增或自减操作符在语句中单独使用，则等价于对对象进行增加或减少的赋值操作。因此，下面的语句

```
y--;
```

等价于语句

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

如果在表达式中使用自增或自减操作符，必须注意计算顺序的问题。如果是前缀位置的自增或自减操作符，那么标识符将先被修改，修改后的新值将被用于表达式的其他部分。如果处于后缀位置，那么在表达式中都会使用标识符现有的值，在语句执行完之后标识符的值才会修改。因此，下面的执行语句

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

等价于执行下述语句：

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

类似地，语句

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

等价于下列语句：

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

当执行等式（2.1）或（2.2）时，若假定x等于5，y等于3，那么x的值将会增至6。但是，在执行等式（2.1）后w的值为3，而执行等式（2.2）后w的值为2。

自增和自减操作符与其他一元操作符的优先级相同。如果在一个表达式中有多个一元操作符，那么它们的结合顺序都是从右到左。当使用后缀表示形式时，对象的值直到语句结束后才增加。准确地说，值的增加何时发生是与系统相关的。由于这个原因，从使用后缀操作符直到语句结束，对象的值在什么时候增加是不确定的。

2.4.6 缩写赋值操作符

C++允许将赋值语句简化成缩写形式。例如，下面的每组语句都有其等价语句：

缩写操作符	等价语句
x += 5;	x = x + 5;
sum *= x;	sum = sum * x;
d /= 4.5;	d = d/4.5;
z %= 2;	z = z%2;

事实上，任意如下形式的语句

```
标识符 = 标识符 操作符 表达式;
```

都可以写成

```
标识符 操作符 = 表达式;
```

在本节的前面，我们用过多重赋值（multiple-assignment）语句：

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

这种语句的表示很清晰，但是下面的表示就不那么明确了：

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

为了正确地计算，我们使用表2.5所示规则，表中规则说明赋值操作应该最后计算，并且其结合规则是从右至左。因此，上面的语句等价于下面的语句：

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

表2.5 算术和赋值操作符的优先级

优先级	操作符	结合性
1	圆括号：()	与最深层的结合
2	一元操作符：+ - ++ --（类型）	从右到左
3	二元操作符：* / %	从左到右
4	二元操作符：+ -	从左到右
5	赋值操作符：= += -= *= /= %=	从右到左

如果我们将缩写赋值操作符用一般的赋值形式替换，可以得到

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

或

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

计算这条语句是熟悉优先级和结合性的一个很好的实践。但是，一般来说，程序中的语句应该具有良好的可读性。因此，不推荐在一个连续赋值语句中使用缩写赋值操作符。同时，请注意我们在缩写赋值操作符和连续赋值操作符周围插入了空格，因为这些操作是在算术操作之后进行的。

练习

给出下面语句在执行后的内存快照，假定执行前 x 等于 2，y 等于 4。同时假定所有的对象都是整数。

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

2.5 标准输入和输出

C++ 使用对象 cin（读作“see in”）完成标准输入，使用 cout（读作“see out”）完成标准输出。这些对象都在标准 C++ 库文件 iostream 中定义。为了在程序中使用这些对象，需要在程序中包括下面的预处理指令：

```
#include <iostream>
```

2.5.1 cout 对象

cout 对象用于将流（stream）输出到标准输出设备上。流是由程序生成的持续的字符流，字符流被发送到输出缓冲区（output buffer）中。当输出缓冲区被填充后，其中的内容将会在标准输出设备上显示。在我们的例子中，假定标准输出设备是屏幕。

标准输出： C++ 使用 ostream 对象 cout 来将表达式的值以流的方式输出到标准输出。cout 定义在头文件 iostream 中。为了使用 cout 对象，必须在程序中包含编译器指令 #include<iostream>。

语法

```
cout << 表达式 << 表达式;
```

示例

```
cout << "The radius is "<< radius << " centimeters\n";
```

cout 使用“<<”操作符来将值输出到屏幕上。下面的语句将 3 个值输出到屏幕上。

```
cout << "The radius of the circle is "
     << radius << " centimeters\n";
```

每个被输出的值都使用“<<”操作符处理。在前面的例子中，输出的第一个值是字符串“The radius of the circle is”，输出的第二个值是对象 radius 的值，第三个输出值是字符串“centimeters\n”。字符（\n）表示换行（newline）符，换行符使得输出时在屏幕上输出一个新行。下面的例子说明了 cout 的用法：

```
cout:

double radius(10), area;
const double PI = acos(-1.0);
cout << "The radius of the circle is: " << radius
     << " centimeters\n"
     << "The area is " << PI*radius*radius
     << " square centimeters\n";
```

这些语句的输出如下：

```
The radius of the circle is: 10 centimeters
The area is 3.14159 square centimeters
```

注意输出显示的 radius 值没有小数部分，虽然它被声明为 double 类型。这是 C++ 对于浮点类型的缺省打印形式，可以使用流函数和操纵符（manipulator）覆写这种格式。

2.5.2 流对象

标识符 cout 被声明为 ostream 类型的对象，它被编译器定义用于将流数据输出到标准输出。类 ostream 也包含了自己的类方法或称为成员函数（member function）。成员函数在类定义中定义，只能被同类型的类对象所调用。因为 cout 是一个 ostream 类型的对象，所以它可以调用 ostream 所有的成员函数来控制与标准输出相关的标识状态。这些格式的标识都是 ostream 类的数据成员。当一个类对象引用其成员函数时，需要使用称为点操作符（dot operator）的特殊操作符。下面的程序中使用了 ostream 类的两个成员函数来设置所需的浮点数的输出格式。函数 setf() 用于将输出的浮点数设置为定点格式，函数 precision() 用于设置所要输出的有效个数位数。

```
/*------------------------------------------------------------------*/
/* Program chapter2_4                                               */
/*                                                                  */
/*    This program computes area of a circle.                       */
/*    Results are displayed with two digits                         */
/*    to the right of the decimal point.                            */
#include<iostream>  //Required for cout, setf() and precision().
#include<cmath>     //Required for acos().
using namespace std;

const double PI = acos(-1.0);

int main()
{
      //Declare and initialize objects.
      double radius(10), area;
      area = PI*radius*radius;

      //Call the setf member function using dot operator.
      cout.setf(ios::fixed); //Fixed form(xx.xx).


      //Call the precision member function using dot operator.
      cout.precision(2);   //Display 2 digits to right of decimal.

      cout << "The radius of the circle is: " << radius
           << " centimeters\nThe area is "
           << area << " square centimeters\n";

      //exit program
```

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

程序的输出结果如下：

```
The radius of the circle is: 10.00 centimeters
The area is 314.16 square centimeters
```

当使用 setf() 函数设置格式标识后，格式标识将一直保持，直到下一次调用 setf() 来改变它，或者使用 unsetf() 函数来重置（重新设置为 0）格式标识。表 2.6 中列出了在 ios 类中定义的一些格式标识。例如，在 ios 类中定义的 ios::showpoint，当 cout 以 ios::showpoint 为参数调用 setf() 函数时，语句如下：

```
cout.setf(ios::showpoint);
```

那么与 cout 对象相关的 showpoint 标识将被设置为 true，所有浮点数在输出到标准输出时都会显示小数点。而语句

```
cout.unsetf(ios::showpoint);
```

将 showpoint 标识重置为 false，变为默认的浮点输出格式。

表 2.6 常用格式标识

标志	意义
ios::showpoint	显示小数点
ios::fixed	定点表示
ios::scientific	科学记数法表示
ios::right	右对齐打印
ios::left	左对齐打印

成员函数 precision() 指明了将要显示的浮点数的有效位数，它的行为还取决于其他格式标识的状态。例如，当 cout 在设置输出格式为定点输出之后调用 precision(2)，那么输出的结果将显示小数点右边的两位有效数字，如程序 chapter2_4 所示。如果 cout 在将输出格式设置为科学记数法之后调用 precision(2)，那么输出结果将总计只显示两位有效数字。

练习

1. 创建一个包含本节讨论的 chapter2_4 程序的文件，并编译运行。

```
The radius of the circle is: 10.00 centimeters
The area is 314.16 square centimeters
```

2. 将 chapter2_4 程序中的 serf () 函数语句替换成下面的语句后编译运行。输出变化了吗？试着解释原因。

```
cout.serf(ios::scientific);
```

3. 将 chapter2_4 程序中的 setf() 函数语句替换成下面的语句后编译运行。输出变化了吗？试着解释原因。

```
cout.setf(ios::showpoint);
```

2.5.3 操纵符

在 chapter2_4 程序中，我们使用了流函数来控制输出格式。输出格式还可以使用操纵符来控制。操纵符是用于控制输出流状态的预定义对象。在 chapter2_5 程序中说明了使用操纵符来控制输出格式的方法。为了使用操纵符，程序必须包含头文件 <iomanip>。

```
/*-------------------------------------------------------------------------*/
/* Program chapter2_5                                                      */
/*                                                                         */
/*   This program computes area of a circle.                               */
/*   Results are diplayed with two digits                                  */
```

```
/*   to the right of the decimal point.                                    */

#include <iostream> //Required for cout, endl.
#include <iomanip>  //Required for setw() setprecision(), fixed.
#include <cmath>    //Required for acos().
using namespace std;

const double PI = acos(-1.0);

int main()
{
   //Declare and initialize objects.
     double radius(10),  area;
     area = PI*radius*radius;

     cout << fixed << setprecision(2);
     cout   << "The radius of the circle is: "
          << setw(10)   << radius << "  centimeters"  << endl;
     cout << "The area of the circle is:   " << setw(10)  << area
          << " square centimeters"  << endl;

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

程序输出如下：

```
The radius of the circle is:   10.00 centimeters
The area of the circle is:  314.16 square centimeters
```

操纵符 endl 在头文件 <iostream> 中定义，它在 chapter2_5 程序中用来另起一个新行。操纵符 endl 给标准输出发送了一个换行符（'\n'），并且立即刷新输出缓冲区，而不是等到输出缓冲区被写满才写入。当为了调试程序而使用 cout 对象打印内存快照时，应该使用操纵符 endl（而不是 '\n'）来确保输出缓冲区被刷新，这样可以在下一条语句执行前看到你的输出。

操纵符 fixed 和 setprecision() 用于设置浮点数的显示格式和有效数字。注意，操纵符不是成员函数，它们不需要使用（.）操作符来引用。但是，它们的效果与在 chapter2_4 程序中使用的成员函数是相同的。操纵符 setw() 用于指定下一个发送到输出缓冲区的值的输出宽度。当需要以表格形式输出结果时，使用 setw() 很有用。表 2.7 中列出了在头文件 <iomanip> 中定义的几个常用操纵符。为了使用这些操纵符，必须在程序中包含头文件 <iomanip>。

表 2.7　常用操纵符

操 纵 符	含　义	操 纵 符	含　义
showpoint	显示小数点	setw(n)	将要打印的下一个整数的最小输出域宽度为 n
fixed	定点显示	right	打印时右对齐
scientific	科学记数法显示	left	打印时左对齐
setprecision(n)	将要打印的整数的有效数字个数设为 n		

练习

假定整型对象 sum 的值为 150，double 型对象 average 的值为 12.368，且已经包含了头文件 <iomanip> 和 <iostream>，给出下面语句的输出，假定每组语句都是相互独立的。

```
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;

7. cout << setprecision(2) << sum << endl << average;

8. cout << fixed << setprecision(3)
        << setw(10) << average << endl << setw(10)
        << sum << endl;
```

2.5.4 cin 对象

对象 cin 是 istream 类型的对象，它被编译器定义用于从标准输入设备中接受输入流。在我们的例子中，假定输入设备是键盘。

标准输出：C++ 使用 istream 类型的对象 cin 来将表达式的值以流的方式从标准输入中将数据存入标识符对应的存储位置。cin 定义在头文件 iostream 中。为了使用 cin 对象，必须在程序中包含编译器指令 #include<iostream>。

语法

```
cin >> 标识符 >> 标识符 >> 标识符;
```

示例

```
cin >> x >> y >> count;
```

cin 使用的输入操作符“>>”用于从键盘接收输入的值并将其赋给某个变量。“>>”操作符使用空白（空格、tab、新行）作为分隔符，它将跳过所有的空白。下面的语句接收了来自键盘的三个输入值。

```
cin >> var1 >> var2 >> var3;
```

在上面的语句中，第一个来自键盘的输入被赋给变量 var1，第二个值赋给 var2，第三个值赋给 var3。当执行一个 cin 语句时，程序将等待输入。在 cin 语句之前通常会使用 cout 输出信息提示（prompt）用户输入数据。提示通常是一段打印在屏幕上的内容，用于提示用户在输入时要注意的顺序和值的数据类型。

来自键盘的输入会被存储在输入缓冲区中，当按“回车”按钮确认后，这些数据才会被发送给程序。这允许用户在输入数据时对数据进行更正或删除。

从键盘输入的数值数据应该使用空白分隔，使用多少空白来分隔都没关系。cin 操作符将跳过所有的空白，直到它收到为语句中的标识符输入的值为止。从键盘输入的数据类型必须与 cin 语句中指定的标识符的数据类型匹配。我们将在第 5 章讨论潜在的输入错误。

下面的例子说明了 cin 的用法：

```
#include <iostream>      //Required for cin
using namespace std;
int main()
{
   double rate, hours;
   int id;
   char code;
   // Prompt user for input
   cout << "Enter the floating point rate of pay "
        << "and hours worked: ";
   cin >> rate >> hours;
   cout << "Enter the employee's integer id: ";
   cin >> id;
   cout << "Enter the tax code (h,r,l): "
   cin >> code;
   cout << rate << endl << hours << endl
        << id << endl << code << endl;
   return0;
}
```

如果键盘的输入如下：

```
10.5 40
556
r
```

那么输入语句中对象的赋值情况如下面的内存快照所示：

double rate	10.5	double hours	40
int id	556	char code	'r'

cout 语句将下列结果输出到屏幕上：

```
10.5
40
556
r
```

“>>”操作符跳过所有的空白，并且根据输入语句中的标识符类型来解释所接收的值。对于某些需要字符数据的应用程序，它并不希望将空白丢弃。cin 可以调用 istream 类的成员函数 get() 来从输入流中读取单个字符。下面的语句

```
char ch;
cin.get(ch);
```

将会从键盘读入下一个字符，并将字符赋值给变量 ch。get() 不会跳过空白，而是将空白作为一个合法的字符，下面的例子进行了说明。

```
char ch1, ch2, ch3;
cout << "Enter three characters: ";
cin.get(ch1);
cin.get(ch2);
cin.get(ch3);
cout << ch1 << ch2 << ch3;
```

如果来自键盘的输入包括下面两行：

```
a
)
```

那么对象 ch1、ch2 和 ch3 得到的值如下面的内存快照所示：

char ch1	'a'	char ch2	'\n'	char ch3	')'

输出如下：

^
:

因为 get() 函数将空白当做合法字符，所以来自键盘的换行符被存储在对象 cb2 中，并在 cout 语句中被打印出来。

2.6 使用 IDE 构建 C++ 解决方案：NetBeans

我们已经讨论了 C++ 程序的通用结构，生成变数据类型、算术操作符以及输入、输出操作符的用法时也写过了一些简单的程序。在本节中，我们将讨论如何使用集成开发环境（Integrated Development Environment，IDE）来开发 C++ 解决方案。IDE 是用于辅助软件解决方案开发的软件包。IDE 包含了编辑器、编译器、调试器，以及许多用于帮助设计开发大型软件解决方案的附加工具。和大部分软件包一样，使用 IDE 也有一条学习曲线，我们需要花费时间和经历来成为一个高效多产的 IDE 用户。在本节中，我们将使用 NetBeans 来为 chapter1_1 程序开发一个 C++ 解决方案。在第 3 章中我们将再次使用 NetBeans 开发 C++ 解决方案，而在第 4 章我们将使用 MS Visual C++ Express IDE 来进行开发。

NetBeans

NetBeans 是一个开源项目，它为包括 Java、C 和 C++ 等多种语言提供开发环境。当启动 NetBeans 应用程序时，将会出现如图 2.1 所示的画面。

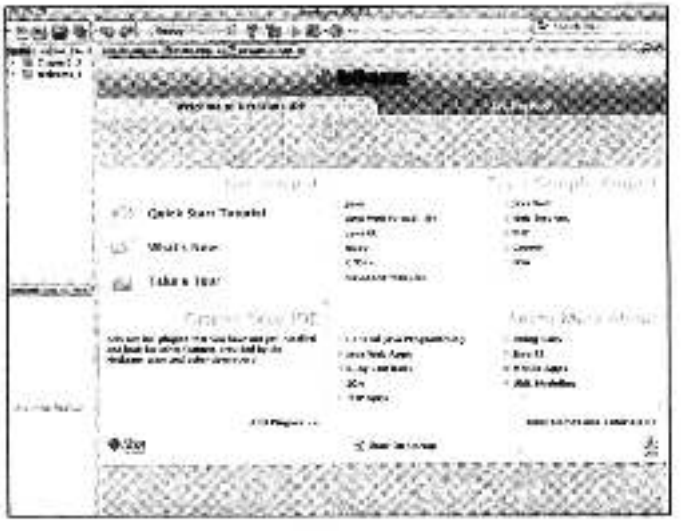


图 2.1

我们将通过一个简单的例子来说明 NetBeans 中的编辑器和编译器的用法。NetBeans 提供了快速入门指南。我们推荐你阅读这个指南，看看如何成为一个更加熟练的 IDE 用户。

为了创建一个新的 C++ 解决方案，我们必须首先创建一个新的 NetBeans 工程。从 NetBeans 窗口的 File 菜单中选择 New Project，将会出现一个新建工程的窗口，类似图 2.2 所示。

对于 C++ 工程，我们从 Categories 中选择 C/C++，然后从 Projects 中选择 C/C++ Application，并点击 Next 按钮，将会出现如图 2.3 所示的 Project Name and Location 窗口。

通过 Browse 按钮选择你所选定的目录，将工程命名为 ProgramChapter1_1，点击 Finish 按钮完成 NetBeans 工程的创建。

回想一下，每个 C++ 解决方案都需要一个名为 main 的函数。为了创建 main 函数，我们必须向工程中添加一个新文件。为了添加新文件，从 File 菜单中选择 New File，将会出现如图 2.4 所示的 Choose File Type 窗口。

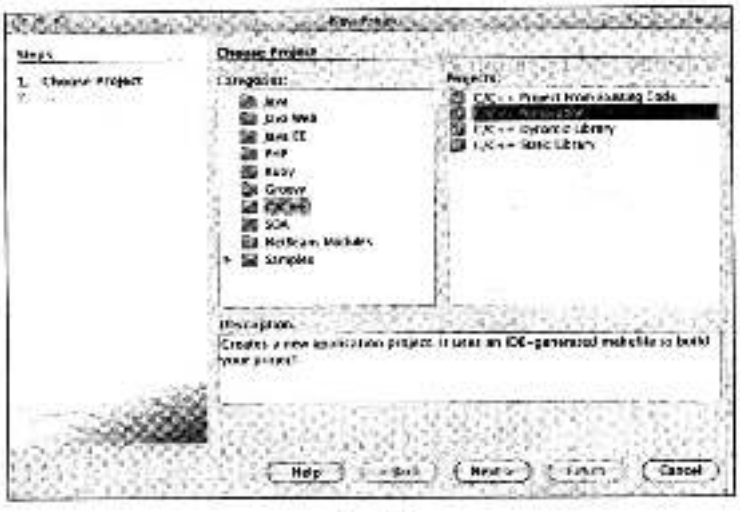

Steps
1. Choose Project
Choose Project
Categories:
Projects:
Description:
Help
Next >
Finish
Cancel

图 2.2

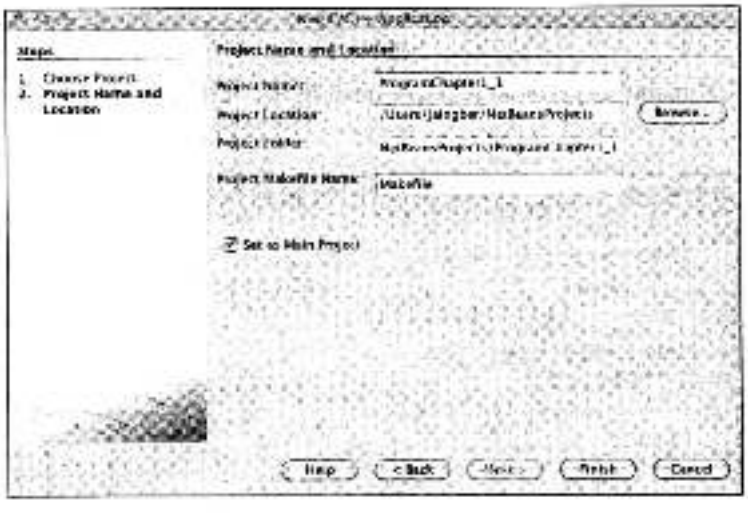

Steps
1. Choose Project
2. Project Name and Location
Project Name and Location
Project Name:
Project Location:
Browse...
Project Folder:
Project Makefile Name:
Makefile
Set as Main Project
Help
< Back
Next >
Finish
Cancel

图 2.3

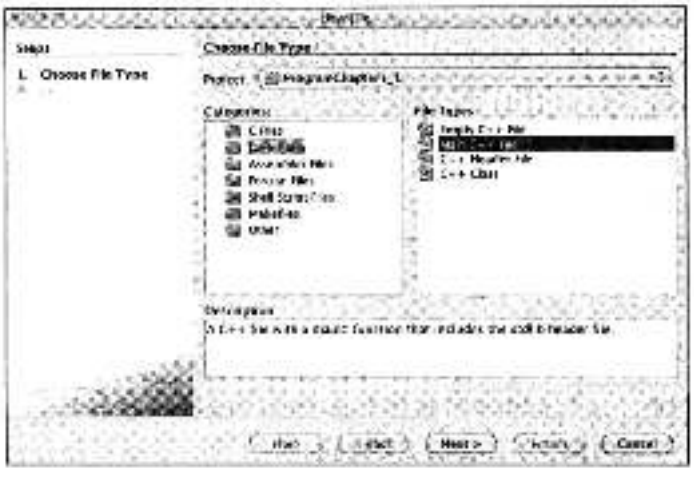

Steps
1. Choose File Type
Choose File Type
Project:
Categories:
File Types:
Description:
Next >
Finish
Cancel

图 2.4

从 Categories 中选择 C++ Files，再从 File Types 中选择 Main C++ File，点击 Next 按钮，将会出现一个 Name and Location 窗口，如图 2.5 所示。

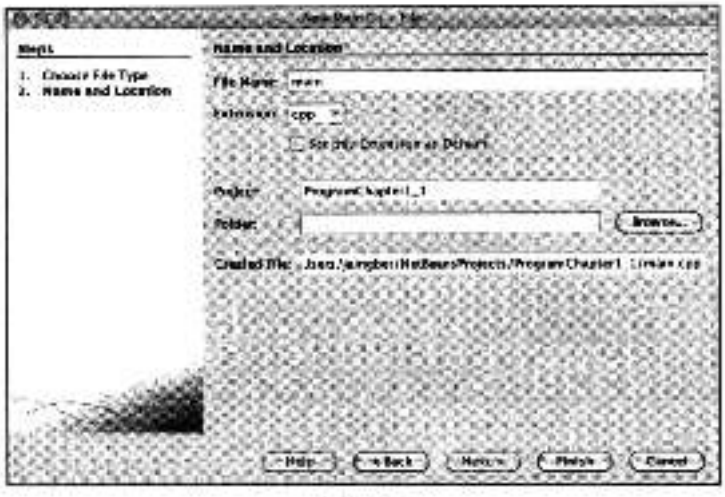


图 2.5

将文件命名为 main，点击屏幕下方的 Finish 按钮，在编辑器窗口中将出现如图 2.6 所示的新建文件。

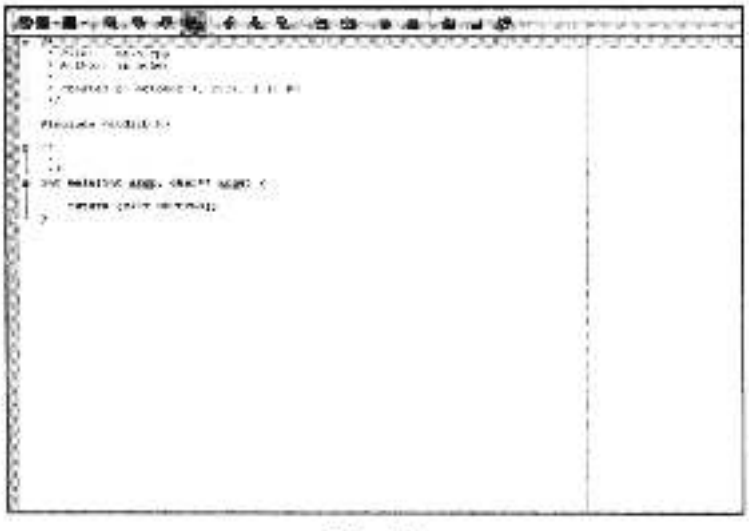


图 2.6

注意 main() 函数有参数 int argc 和 char** argv，我们将在后续内容中讨论这些可选参数。我们将把程序 chapter1_1 中的语句添加到 main() 中，修改后的程序如图 2.7 所示。

为了编译程序，从 NetBeans 窗口顶部的 Run 菜单中选择 Build Main Project，输出窗口如图 2.8 所示。

从窗口的输出内容中我们可以看到构建成功，因此我们可以尝试执行程序，并观察执行结果。为了执行程序，从 Run 菜单中选择 Run Main Project，这样我们的程序将会被执行，输出将显示在一个单独的窗口中，如图 2.9 所示。

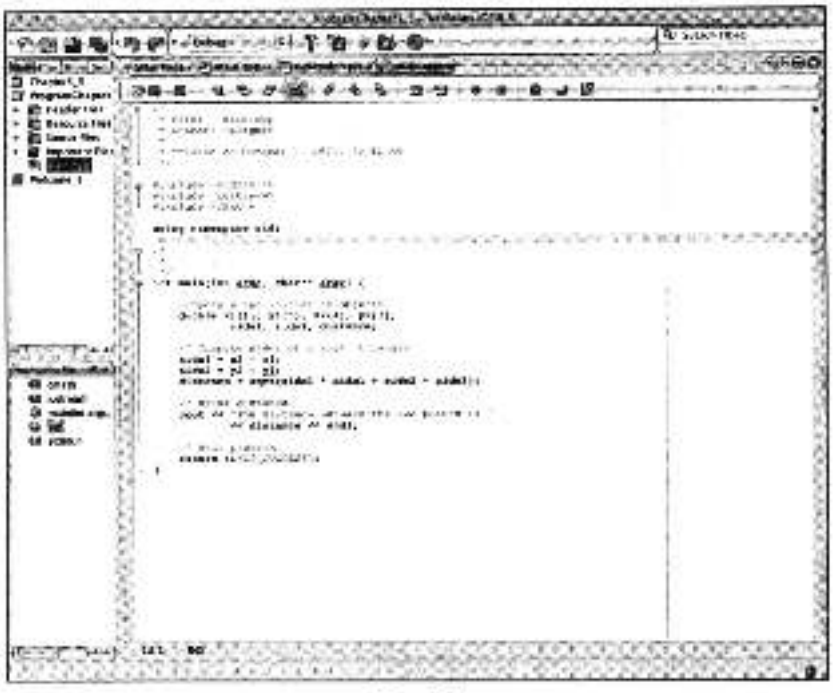


图 2.7



图 2.8

下面我们将程序员自定义类 Point 添加到工程中。从 File 菜单中选择 New File。从 Categories 中选择 C++ Files，从 File Types 中选择 C++ Class，然后点击 Next 按钮，如图 2.10 所示。



图 2.9

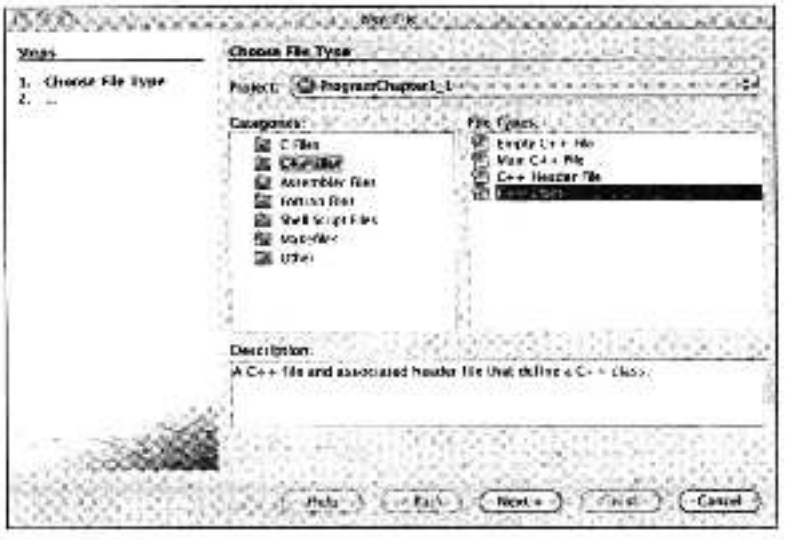


图 2.10

注意，IDE 同时创建了 point.cpp 和 point.h 文件。两个文件中都包含了基本的构造函数和析构函数，用以帮助开发新的类。在 point.h 中还包括了 #ifndef、#define、#endif 等编译器指令。这些编译器指令可以防止 point.h 被多次包含。我们将在第 10 章再次讨论这些内容。我们将 Point 类的声明和实现部分添加到文件中，如图 2.11 所示。

当我们编译运行这个工程时，应该得到图 2.12 所示的输出，因为我们还没有修改 main.cpp。为了测试我们自定义的 Point 类，将 2.3.2 节声明两个 Point 类型对象的语句添加到 main.cpp 中，如图 2.13 所示。



图 2.11



图 2.12

接下来，我们再次编译运行我们的工程，得到如图 2.14 所示的输出。

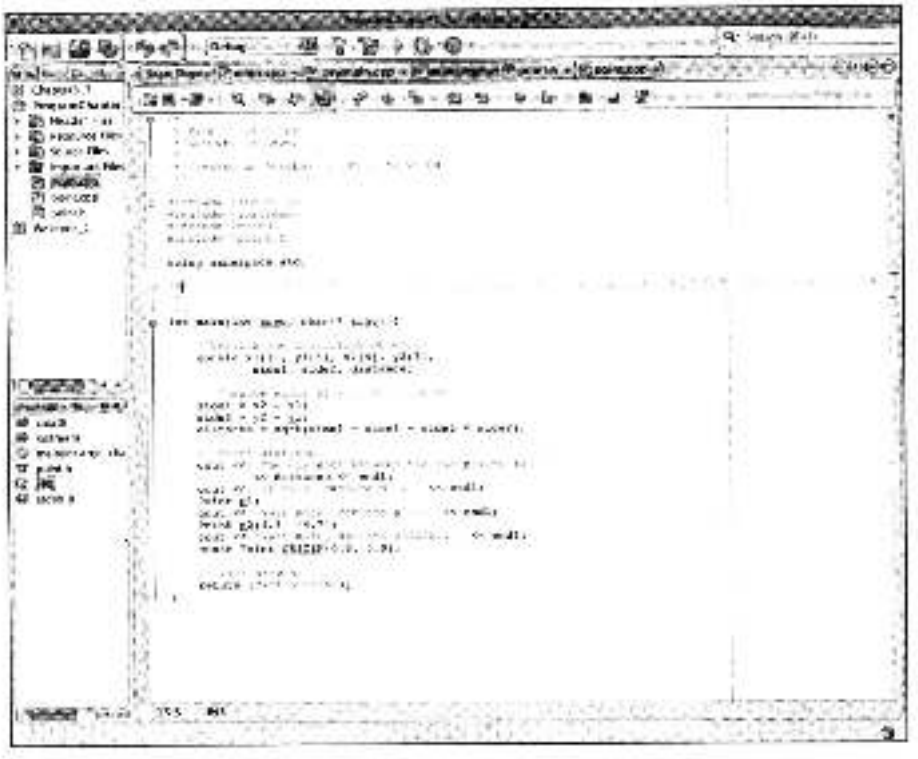


图 2.13

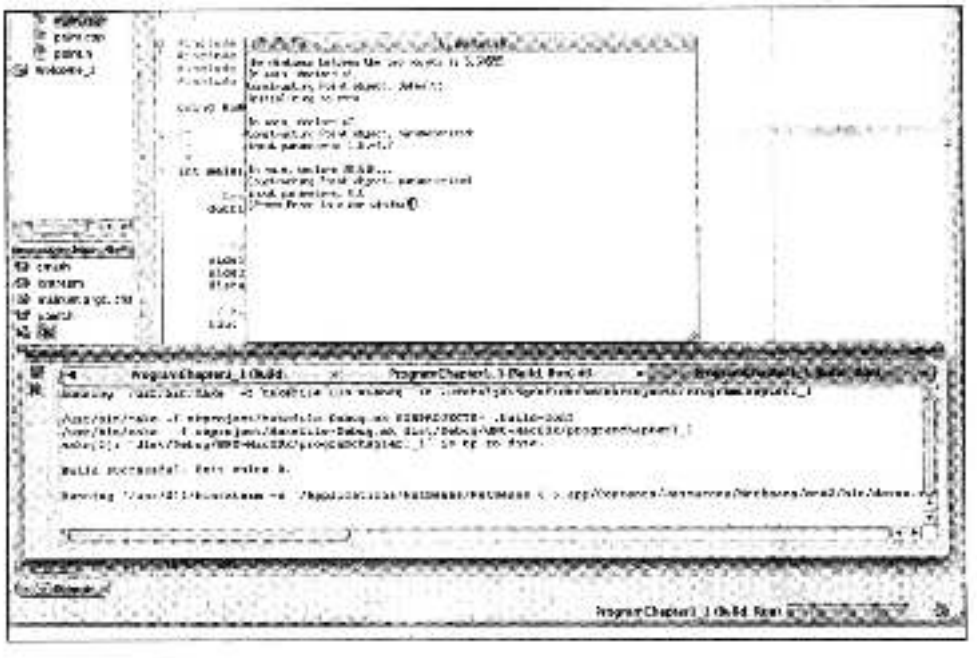


图 2.14

从输出中我们看到，三个 Point 对象都已经被初始化了。

修改

这些问题与本节中用来说明 NetBeans 用法的工程 ProgramChapter1_1 相关。

1. 注释掉 main.cpp 中的 using 指令（//using namespace std;），重新构建工程。不要去掉 using 指令的注释，修改所有构建时报出的错误，然后编译运行工程。
2. 注释掉 #include"PointH" 指令，构建工程。出现了哪些错误？如何更正错误？
3. 删除 main.cpp 中一条单独语句的分号，构建工程，出现了哪些错误？

2.7 包含在 C++ 标准库中的基本函数

除了加、减、乘、除外，工程上的应用通常还需要其他的算术计算。例如，许多表达式都需要使用对数、指数和三角函数。本节我们将讨论在标准 C++ 库中定义的可用数学函数和字符函数。例如，下面的语句计算了角度 theta 的 sine 值并将结果存入对象 b 中。

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

sin() 函数将参数 theta 视作弧度形式。如果 theta 是角度形式，我们需要首先使用一条语句将它转换成弧度形式（180 度 =π 弧度）。

```
const double PI = acos( -1.0);
...
theta_rad = theta*PI/180.0;
b = sin(theta_rad);
```

上面的转换过程还可以表示成下面的形式：

```
b = sin(theta*PI/180.0);
```

像 sin（theta）这样的函数引用表示为一个单独的值。函数名的圆括号中包含了函数的输入，称为函数参数（function argument）。函数可以不包含参数、一个参数或多个参数，这取决于函数的定义。如果函数有一个以上的参数，那么需要十分注意参数的顺序。某些函数还要求参数具有指定的单位。如在三角函数中，都假定参数是以弧度为单位。大多数数学函数都假定函数参数是 double 类型的值；如果传递的参数类型不是 double，那么在函数执行前参数会先被提升为 double 类型。

函数引用还可以作为另一个函数引用的参数。例如，下面的语句计算了 x 的绝对值的对数值：

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

当使用一个函数来计算另一个函数的参数时，要确保每个函数的参数都用括号包含。这种嵌套的函数形式称作函数复合（composition of functions）。

现在我们讨论一下在工程计算中常用的几类函数，其他函数将在讨论相关主题时再提及。附录 A 中包含了标准 C++ 库中函数的更多信息。

2.7.1 基本的数学函数

基本的数学函数包含了诸如计算绝对值、计算平方根等常用的计算函数。此外，还包括一组用于取整的函数。这些函数假定所有的参数类型均为 double 类型，函数的返回值也是 double 类型。如果传递的参数类型不是 double，将会发生按照 2.2 节中所述规则进行的转换过程。在引用标准 C++ 库中的数学函数时，需要包含下面的预处理指令：

```
#include <cmath>
```

我们列出了关于这些函数的一些简短描述。

fabs(x) 该函数计算 x 的绝对值。

sqrt(x) 该函数计算 x 的平方根，其中 $x \geq 0$。

pow(x, y) 该函数计算 x 的 y 次幂，如果 $x = 0$ 且 $y \leq 0$ 或者 $x < 0$ 而 y 不是整数，会发生错误。

ceil(x) 该函数将 x 向上取整，返回大于 x 、最接近 x 的整数值。如 ceil(2.01)等于3。

floor(x) 该函数将 x 向下取整，返回小于 x 、最接近 x 的整数值。如 floor(2.01)等于2。

exp(x) 该函数计算 e^x 的值，其中 e 为自然对数，近似等于 2.718 282。

log(x) 该函数计算 x 以自然对数 e 为底的对数值，若 $x \leq 0$ 则出错。

log10(x) 该函数返回 $\log_{10}x$ 的值，即 x 以 10 为底的常用对数值，若 $x \leq 0$ 则出错。

记住一个负值点0的对数是不存在的，因此如果你使用一个负值作为对数函数的参数将会发生执行错误。

练习

计算下面的表达式。

```
1. fabs(-2.6)                          2. ceil(-2.6)
3. pow(2.0,-2)                         4. sqrt(floor(15.7))
5. abs(-10+2.5)                        6. floor(ceil(10.8))
7. log10(100) + log(10.00)             8. acos(pow(-2,5.0))
```

2.7.2 三角函数

三角函数（trigonometric function）也包含在 cmath 之中，这些函数的参数均为 double 类型，返回值也均为 double 类型。此外，如前所述，三角函数将所有的角度表示均视作以弧度为单位。为了将弧度转换成角度，或者将角度转换为弧度，可以使用下面的转换：

```
const double PI = acos(-1.0);
...
angle_deg = angle_rad*(180/PI);
angle_rad = angle_deg*(PI/180);
```

sin(x) 该函数计算 x 的正弦值，其中 x 为弧度值。

cos(x) 该函数计算 x 的余弦值，其中 x 为弧度值。

tan(x) 该函数计算 x 的正切值，其中 x 为弧度值。

asin(x) 该函数计算 x 的反正弦值，其中 x 必须在 [-1, 1] 中；函数返回一个在 $[-\pi/2, \pi/2]$ 的弧度值。

acos(x) 该函数计算 x 的反余弦值，其中 x 必须在 [-1, 1] 中；函数返回一个在 $[0, \pi]$ 的弧度值。

atan(x) 该函数计算 x 的反正切值，函数返回一个在 $[-\pi/2, \pi/2]$ 的弧度值。

atan2(y, x) 该函数计算 y/x 的反正切值，函数返回一个在 $[-\pi, \pi]$ 的弧度值。

注意，atan 函数总是返回一个在第一或第四象限的角度，而 atan2 则返回一个可位于任意象限的角度，这取决于 x 和 y 的符号。因此，在许多应用程序中都采用 atan2 函数来取代 atan 函数。

其他的三角和反三角函数可以使用下面的公式来计算 [10]：

$$\sec x = \frac{1}{\cos x} \qquad a\sec x = a\cos\left(\frac{1}{x}\right)$$

$$\csc x = \frac{1}{\sin x} \qquad a\csc x = a\sin\left(\frac{1}{x}\right)$$

$$\cot x = \frac{1}{\tan x} \qquad a\cot x = a\cos\left(\frac{x}{\sqrt{1+x^2}}\right)$$

在三角函数中使用角度表示而不是弧度表示是程序中一个常见的错误。

练习

给出练习 1 ～ 3 中用于计算指定值的赋值语句，假定所有的对象都已经被声明并赋予了合适的值。同时假定有以下常量声明：

```
const double g = 9.8;
const double PI = acos(-1.0);
```

1. 速率计算：

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

2. 长度收缩：

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

3. 位于半心圆柱体截面中的参考平面到圆柱体重心的距离

$$\text{center} = \frac{38.1972\,(r^3 - s^3)\sin\alpha}{(r^2 - s^2)\,\alpha}$$

给出练习 4 ～ 6 中各语句所对应的公式。

4. 电子振荡频率：

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

5. 抛射范围：

```
range = (v0*v0/g)*sin(2*(1*a));
```

6. 斜坡底部圆盘的速度：

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

*2.7.3 双曲函数

双曲函数（hyperbolic function）是有关自然指数 e^x 的函数，反双曲函数是有关自然对数 $\ln x$ 的函数。这类函数在设计诸如某些数字滤波器这样的专业应用时很有用。在标准 C++ 库中包含下文所述的几个双曲函数。在使用双曲函数时，也需要包含头文件 cmath。

`sinh(x)` 该函数计算 x 的双曲正弦值，等价于 $\frac{e^x + e^{-x}}{2}$。

`cosh(x)` 该函数计算 x 的双曲余弦值，等价于 $\frac{e^x - e^{-x}}{2}$。

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

其他的双曲函数和反双曲函数可以使用下面所列的关系式进行计算 [10]：

$$\coth x = \frac{\cosh x}{\sinh x}$$

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

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

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

$$\operatorname{asinh} x = \ln\left(x + \sqrt{x^2-1}\right) \quad (|x| \geq 1)$$

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

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

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

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

许多双曲函数和反双曲函数对于参数的取值范围都有很严格的要求。如果是从键盘键入参数，应当提醒用户对其的范围限制要求。在下一章中，我们将介绍 C++ 中的一些语句，这些语句可以帮助你判断在程序执行过程中的值是否在适合的范围中。

练习

给出计算下面有关 x 的值的赋值语句（假设 x 的值都在计算时所要求的范围之内）。

1. $\coth x$
2. $\operatorname{sech} 3x$
3. $\operatorname{csch} 4x$
4. $\operatorname{acot} 6x$
5. $\operatorname{acosh} x$
6. $\operatorname{acsch} x$

2.7.4 字符函数

标准 C++ 库中包含了许多用于处理字符数据的预定义函数。这些函数可以分为两类：一类用于对字符进行大小写的转换，另一类用于字符的比较。在使用这些函数时，需要包含下面的预处理指令：

```
#include <cctype>
```

下面的语句将存储在 ch 对象中的字符从小写变成大写，并将结果存储在字符对象 chupper 中：

```
ch_upper = toupper(ch);
```

如果 ch 是一个小写字母，则 toupper() 函数将返回对应的大写字符；否则，该函数返回 ch。注意，该函数没有改变参数 ch。

使用字符比较函数时，若比较结果为真，则返回一个非零值，否则返回零。下面的语句中调用了函数 isdigit()。如果字符 ch 的值为数字字符（0…9），则 isdigit() 函数返回一个非零值，若 ch 不是数字字符，则返回 0。

```
digit = isdigit(ch);
```

附录 A 中给出了有关这些函数的简短说明。

2.8 解决应用问题：速率计算

本节我们将完成一个与汽车性能相关的应用程序。涡轮发动机已经使用了几十年，它将喷气式引擎的动力、可靠性与推进器的效率结合起来。对于早期的活塞推进器引擎来说，它是一个重大的进步。但是，它的应用范围限制在小型客机上，因为它的动力不如大型客机使用的涡扇式喷气发动机。无导管风扇（Unducted Fan, UDF）引擎使用了更先进的推进器技术，它缩小了涡轮引擎和涡扇式引擎之间的性能差距。新材料、叶片形状、高转速使得 UDF 驱动的飞机可以飞得像使用涡扇式引擎一样快，并且更加节省燃料。同时，UDF 比传统的涡轮发动机的噪声更小。

在一次 UDF 驱动的飞机的飞行测试中，飞行员将动力水平设为 40 000N（牛顿），这可以使质量为 20 000kg 的飞机获得 180m/s 的初速度。然后将引擎的动力水平设为 60 000N，这时飞机开始加速。当飞机的速度增加时，空气动力阻力会与飞行速度的平方成正比。渐渐地，当 UDF 引擎的推力刚超过阻力时，飞机将会达到一个新的初速度。下面的公式用于计算飞机从离开取得状态到达到新的初速度时刻（约在第 120s）的速度和加速度的估算值。

$$\text{Velocity} = 0.000\,01 * \text{time}^3 - 0.004\,88 * \text{time}^2 + 0.757\,95 * \text{time} + 181.3566$$

$$\text{Acceleration} = 3 - 0.000\,062 * \text{velocity}^2$$

函数曲线如图 2.15 所示，可以看到，当飞机达到新的初速度时，加速度接近于 0。

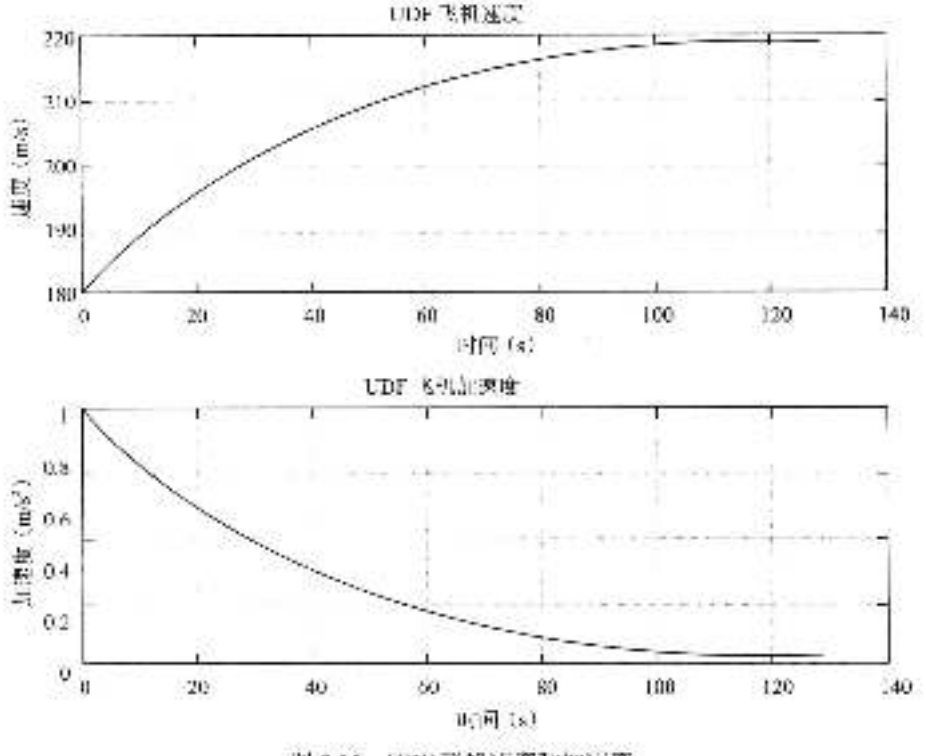


图 2.15 UDF 飞机速度和加速度

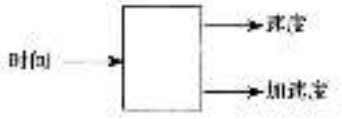
编写一个程序，要求用户输入一个代表从动力笔开始增长直到现在已逝去的时间量（以秒为单位），计算到该时间时相应的飞机速度和加速度。

1. 问题描述

计算在动力水平发生变化后飞机新的速度和加速度。

2. 输入/输出描述

下面的应用说明程序的输入是一个时间值，输出是新的速度值和加速度值。可以使用内建数据类型 double 来表示这些值。



3. 用例

假设输入的时间值为 50s，使用计算速度和加速度的公式，我们可以计算得到下面的值：

速度 = 208.3 m/s

加速度 = 0.31 m/s^2

4. 算法设计

算法设计的第一步是将问题的解决方案分解为一组顺序执行的步骤：

分解提纲

1）读取新的时间值

2）计算对应的速度和加速度值

3）打印新的速度和加速度

因为这个程序很简单，我们可以直接将上面的分解步骤转换为 C++ 形式。

```
/*-------------------------------------------------------------*/
/*  Program chapter2_6                                         */
/*                                                             */
/*  This program estimates new velocity and                    */
/*  acceleration values for a specified time.                  */

#include<iostream>  //Required for cin,cout
#include<iomanip>   //Required for setprecision(), setw()
#include<cmath>     //Required for pow()
using namespace std;

int main()
{
   //  Declare objects.
   double time, velocity, acceleration;

   //  Get time value from the keyboard.
   cout << "Enter new time value in seconds: \n";
   cin >> time;

   //  Compute velocity and acceleration.
   velocity = 0.03001*pow(time,3)   0.30488*pow(time,2)
            + 3.73795*time + 181.3366;
   acceleration = 3 - 0.000062*velocity*velocity;
   //  Print velocity and acceleration.
   cout << fixed << setprecision(3);
   cout << "Velocity = " << setw(10)
        << velocity << " m/s" << endl;
```

```
    cout << "Acceleration = " << setw( 14)
         << acceleration << "m/s^2" << endl;

    // Exit program.
    return 0;
}
/*--------------------------------------------------------------------*/
```

5. 测试

我们首先使用用例中的数据来测试程序。交互过程如下：

```
Enter new time value in seconds:
60
Velocity = 208.304 m/s
Acceleration =   0.510 m/s^2
```

因为计算结果与用例中的计算结果一致，我们可以采用其他的时间值来测试该程序。如果计算结果与用例不一致，我们应当确定错误来自用例还是来自程序。

修改

这些问题与本节设计用来计算速度和加速度值的程序相关。

1. 输入不同的时间值，直到找到一个速度在 210 ~ 211m/s 之间的结果为止。
2. 输入不同的时间值，直到找到一个加速度在 0.5 ~ 0.6m/s^2 之间的结果为止。
3. 修改程序，将键入值的单位从秒变为分钟。记住，公式中的时间是以秒为单位的。
4. 修改程序，使输出结果以英尺/秒和英尺/秒^{2}的形式表示（1 米 = 39.37 英寸）。

2.9 系统限制

在 2.2 节中，我们给出了在 Microsoft Visual C++ 6.0 编译器中使用的整型和浮点型数值的最大值表。为了打印出在我们使用的系统中的一个类似表格，可以使用下面的程序。注意程序中包含了三个头文件：包含 iostream 头文件是因为在程序中使用了 cout；包含 climits 头文件是因为它包含了与整型类型相关的范围信息；包含 cfloat 头文件是因为它包含了与浮点类型相关的范围信息。在附录 A 中包含了更多与系统相关的常量和限制值信息。

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*  Program Chapter2_2                                                */
/*                                                                    */
/*  This program prints the system limitations.                       */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include<iostream>
#include<climits>
#include<cfloat>
using namespace std;
int main()
{
    // Print integer type maxima.
    cout << "short maximum: " << SHRT_MAX << endl;
    cout << "int maximum: " << INT_MAX << endl;
    cout << "long maximum: " << LONG_MAX << endl << endl;

    // Print float precision, range, maximum.
    cout << "float precision digits: " << FLT_DIG << endl;
    cout << "float maximum exponent: "
         << FLT_MAX_10_EXP << endl;
    cout << "float maximum: " << FLT_MAX << endl << endl;
```

```
    //  Print double precision, range, maximum.  /
    cout << "double precision digits: " << DBL_DIG << endl;
    cout << "double maximum exponent: "
         << DBL_MAX_10_EXP << endl;
    cout << "double maximum: " << sizeof(double) << endl << endl;

    //  Print long precision, range, maximum.  /
    cout << "long double precision: " << LDBL_DIG << endl;
    cout << "long double maximum exponent: "
         << LDBL_MAX_10_EXP << endl;
    cout << "long double maximum: " << sizeof(long double) << endl;

    //  Exit program.
    return 0;
}
/*                                                                    */
```

本章小结

本章我们讨论了编写简单 C++ 程序用到的计算和打印语句，还给出了在程序运行时接收键入的语句。计算语句包括标准的算术操作和变量可用于工程解决方案的数学函数。

关键术语

abbreviated assignment（缩写赋值）
ANSI code（ANSI 码）
argument（参数）
assignment statement（赋值语句）
associativity（结合性）
binary codes（二进制码）
binary operator（二元操作符）
case sensitive（大小写敏感）
cast operator（类型转换操作符）
character（字符）
comment（注释）
composition（复合）
constant（常量）
declaration（声明）
exponential notation（指数表示法）
expression（表达式）
field width（域宽）
floating-point value（浮点值）
garbage value（垃圾值）
hyperbolic function（双曲函数）
identifier（标识符）
initial value（初始值）
keyword（关键字）
linear interpolation（线性插值）
mantissa（小数部分）
math function（数学函数）
memory snapshot（内存快照）
modulus（模）
multiple assignment（多重赋值）
object（对象）
overflow（上溢）
parameter（参数）
postfix（后缀）
precedence（优先级）
precision（精度）
prefix（前缀）
preprocessor directive（预处理指令）
prompt（提示）
range（范围）
scientific notation（科学记数法）
standard C++ library（标准 C++ 库）
statement（语句）
symbolic constant（符号常量）
system dependent（系统相关）
trigonometric function（三角函数）
truncate（截断）

unary operator（一元操作符）　　whirespace（空白）

underflow（下溢）

C++ 语句总结

用于包含标准C++库文件的预处理指令

包含语句

一般形式：

```
#include<filename>
```

示例：

```
#include <iostream>
#include <cmath>
#include <string>
```

类型声明语句

一般形式：

```
datatype identifier [, identifier];
```

示例：整型声明

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

示例：浮点类型声明

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

示例：字符和字符串声明

```
char ch;
string name;  //Requires #include<string>
```

符号常量的声明

一般形式：

```
const datatype identifier = expression;
```

示例：

```
const double PI = acos(-1.0);
const int MAX_SIZE = 100;
```

赋值语句

一般形式：

```
identifier = expression;
```

示例：

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

键盘输入语句

一般形式：

```
cin >> identifier [>> identifier];
```

示例：

```
cin >> hours;
cin >> minutes >> seconds;
```

从键盘输入字符

一般形式：

```
cin.get(identifier);
```

示例：

```
cin.get(ch);
```

屏幕输出语句

一般形式：

```
cout << expression [<< expression];
```

示例：

```
cout << "The area is " << area << " square feet." << endl;
```

程序退出语句

一般形式：

```
return integer;
```

示例：

```
return 0;
```

注意事项

1. 在程序中使用注释以增加可读性，同时在程序中写注释说明执行步骤。
2. 使用空行和缩进表明程序的结构。
3. 如果可能的话，在为对象命名时尽量表明它的单位。
4. 符号常量常用来表示一些工程上的常量，如π，它们都应该使用大写形式，以便于识别。
5. 在算术操作符和赋值操作符周围使用一致的空白风格。
6. 在复杂的表达式中使用圆括号增加可读性。
7. 长的计算表达式应该分为几条语句来完成。
8. 在程序的输出中，确保输出的数值都包括了相应的单位。
9. 使用用户提示信息来描述从键盘输入值时应注意的事项，如输入值的单位等。

调试要点

1. 记住 C++ 语句及声明必须以分号结尾。
2. 预处理指令不用分号结尾。
3. 如果可能的话，避免混合类型，这可能导致潜在的信息丢失。
4. 在长表达式中使用圆括号，确保正确的计算顺序。
5. 使用 double 精度或扩展精度来避免出现上溢或下溢问题。
6. 如果输入值的类型与 cin 语句中要求的变量类型不匹配，可能会出错。
7. 确保 cin 对象与 >> 操作符一起使用。
8. 确保 cout 对象与 << 操作符一起使用。
9. 在嵌套函数引用中，每个函数的参数都应该用圆括号括起来。

10. 记住对数函数不能使用负数作为参数。
11. 确保在三角函数中使用弧度值。
12. 记住，许多反三角函数对双并两数对输入值都有范围限制。
13. 记住，字符数字的整数表示形式与纯粹的数值数字的整数表示形式是不同的。

习题

判断题

1. 程序的执行从 main 函数开始。
2. C++ 不是大小写敏感的。
3. 声明可以放在程序的任何地方。
4. 语句和声明必须以分号结束。
5. 整数除法的结果是一个取整后的结果。

指出下面声明语句的正误，如果是错误的，请更正。

6. `int i, j, k,`
7. `float f1(11), f2(202.00);`
8. `DOUBLE D1, D2, D3;`
9. `float a_(a2);`
10. `int x, m m;`

多选题

11. 下面（　　）不是 C++ 关键字。

(a) `const`　　(b) `goto`
(c) `static`　　(d) `when`
(e) `unsigned`

12. 在声明中，类型声明与对象名字是用（　　）分隔的。

(a) 句号　　(b) 空格
(c) 分号　　(d) 以上都不是

13. 下面（　　）声明正确地将 x，y，z 定义为 double 类型的对象。

(a) `double x, y, z;`　　(b) `long double x,y,z`
(c) `double x=y=z;`　　(d) `double X, Y, Z`

14. 在 C++ 中，操作符 % 用于计算（　　）。

(a) 整数除法　　(b) 浮点除法
(c) 整数除法的余数　　(d) 浮点数除法的余数
(e) 以上都不是

15. 下面（　　）赋值结果为 0。

(a) `result = 9%2 - 1;`　　(b) `result = 8%5 - 1;`
(c) `result = 2 - 3%2;`　　(d) `result = 2 - 6%2;`
(e) `result = 2 - 8%5;`

给出下面每组语句执行后相应的内存快照。

16.
```
int x1;
...
x1 = 3 + 9%5 - 5;
int x(1), z(2);
...
z = z/++x;
```

```
17. double a(3,8), z;
    int n(2), y;
    ...
    x = (y*a/n)+2;
```

给出问题 18 ~ 20 中每组语句的输出。

```
18. float value_1(3.78263);
    ...
    cout << "value 1 = " << value_1;
19. double value_4(55.65032);
    ...
    cout << scientific << "value 4 = " << value_4;
20. int value_5(7/50);
    ...
    cout << "value 5 = " << fixed << value_5 << endl;
```

编程题

转换问题。这组问题涉及将一个值从一种单位表示转换为另一种单位表示。每个程序都应该提示用户输入值的单位，并打印出以新单位表示的值。

21. 写一个程序将英里转换为公里。(1 mi = 1.609 344 0 km)
22. 写一个程序将公里转换为英里。(1 mi = 1.609 344 0 km)
23. 写一个程序将英镑转换为千克。(1 kg = 2.205 lb)
24. 写一个程序将牛顿转换为英镑。(1 lb = 4.448 N)
25. 写一个程序将华氏度转换为兰氏度。(TF = TR − 459.67 兰氏度)
26. 写一个程序将摄氏度转换为兰氏度。(TF = TR − 459.67 兰氏度以及 TF = (9/5) TC + 32 华氏度)
27. 写一个程序将绝对温标转换为华氏度。(TR = (9/5) TK 以及 TF = TR − 459.67 兰氏度)

面积和体积问题。下面问题都是根据用户的输入计算相关的面积或体积。每个程序都应该提示用户所要输入的对象信息。

28. 写一个程序计算边长为 a 和 b 的矩形面积 A。($A = a * b$)
29. 写一个程序计算底和高分别为 b、h 的三角形面积 A。($A = \frac{1}{2}(b * h)$)
30. 写一个程序计算半径为 r 的圆面积 A。($A = \pi r^2$)
31. 写一个程序计算角度为 u (弧度制)、半径为 r 的扇形面积 A。($A = r^2\theta/2$)
32. 写一个程序计算角度为 d (角度制)、半径为 r 的扇形面积 A。($A = r^2\theta/2$，其中 θ 是弧度)
33. 写一个程序计算半长短轴分别为 a 和 b 的椭圆面积 A。($A = \pi a * b$)
34. 写一个程序计算半径为 r 的球表面积 A。($A = 4\pi r^2$)
35. 写一个程序计算半径为 r 的球体积 V。($V = (4/3)\pi r^3$)
36. 写一个程序计算半径为 r、高度为 h 的圆柱体体积 V。($V = \pi r^2 h$)

氨基酸分子重量问题。氨基酸由氧元素、碳元素、氮元素、硫元素和氢元素组成，如表 2.8 所示。各元素的原子量如下：

元素	原子量
氧	15.9994
碳	12.011
氮	14.006 74
硫	32.066
氢	1.007 94

37. 写一个程序计算并打印出甘氨酸的分子量。

表 2.6 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
天冬氨酸	4	4	1	0	5
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

38. 写一个程序计算并打印出谷氨酸和谷氨酸盐的分子量。

39. 写一个程序，让用户分别键入组成某种氨基酸的五种元素的原子个数，计算并打印出相应氨基酸的分子量。

40. 写一个程序，让用户分别键入组成某种氨基酸的五种元素的原子个数，计算并打印出该氨基酸中原子的平均质量。

以 b 为底的对数。为了计算 x 以 b 为底的对数值，可以使用下面的关系式计算。

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

41. 写一个程序，读入一个正数，并计算它以 2 为底的对数值。例如 8 以 2 为底的对数值为 3，因为 $8=2^3$。

42. 写一个程序，读入一个正数，计算并打印它关于以 8 为底的对数值。例如 64 以 8 为底的对数值为 2，因为 $8^2=64$。

第 3 章

Engineering Problem Solving with C++, 3e

控制结构：选择

工程挑战：全球变化

海水主要有 3.5% 来自火山爆发和岩石分解的各种物质（盐、金属和气体）的水。最咸的海水是位于赤道附近的大西洋，这里的蒸发率高于降水率。海水盐度是海水中所溶解的矿物质的测量值。氯在海水成分组成中占比约为 55%，钠占之约为 30.6%。其下的组成成分包括硫酸盐（7.7%）、镁（3.7%）、钙（1.2%）和钾（1.1%）。在海洋中各个地点的盐度都是不同的，但是通常都在 33‰ 一 38‰ 之间，或者说在 3.3% 一 3.8% 之间。盐度通常通过仪器测量海水的电导性来得到；水中溶解的矿物质越多，导电性越好。在本章的后面部分，我们将讨论盐度和冰点温度之间的关系，并使用线性插值来计算对应于给定盐度的海水冰点温度。

教学目标

本章我们所讨论的问题解决方案中包括：

- 判断条件表达式的真假
- 在程序中使用分支的选择结构
- 算法的设计，使用流图和伪代码描述算法

3.1 算法设计

在第 2 章中我们设计的 C++ 程序比较简单。算法的步骤是顺序的，包含了一般的从键盘读取信息、计算新信息和打印新信息。在解决工程问题时，大部分解决方案都需要更复杂的步骤，因此我们在解决问题的过程中需要扩展算法的设计。

自顶向下的设计

自顶向下的设计以顺序的方式给出了问题解决方案的宏观描述。然后我们通过对这一全局描述不断地重新定义、细化，直到所有的步骤可以详细到能够转化为程序语句为止。我们使用在第 1 章和第 2 章中的分解提纲来给出问题解决方案的第一个定义。这个提纲是按照顺序的步骤来书写的，可以使用流图表示，也可以使用分步提纲。对于十分简单的问题，比如第 2 章中的问题，我们可以直接将分解提纲转换成 C++ 语句：

分解提纲

1）读取新的时间值。

2）计算对应的速率和加速度值。

3）打印新的速率和加速度。

但是对于大多数问题而言，我们都需要将分解提纲细化成更细节的描述。这种处理过程称为分治（divide-and-conquer）策略，因为我们将问题不断地分解为更小的问题。为了描述这种逐步的细化过程，我们使用伪代码或流程图来进行描述。

我们使用伪代码（pseudocode）或流程图（flowchart）来完成提纲到具体步骤的细化过程。伪代码使用类自然语言的语句来描述算法中的步骤，流程图则使用图示来描述算法步骤。图 3.1 中给出了大部分算法中的基本步骤以及对应的伪代码和流程图表示形式。

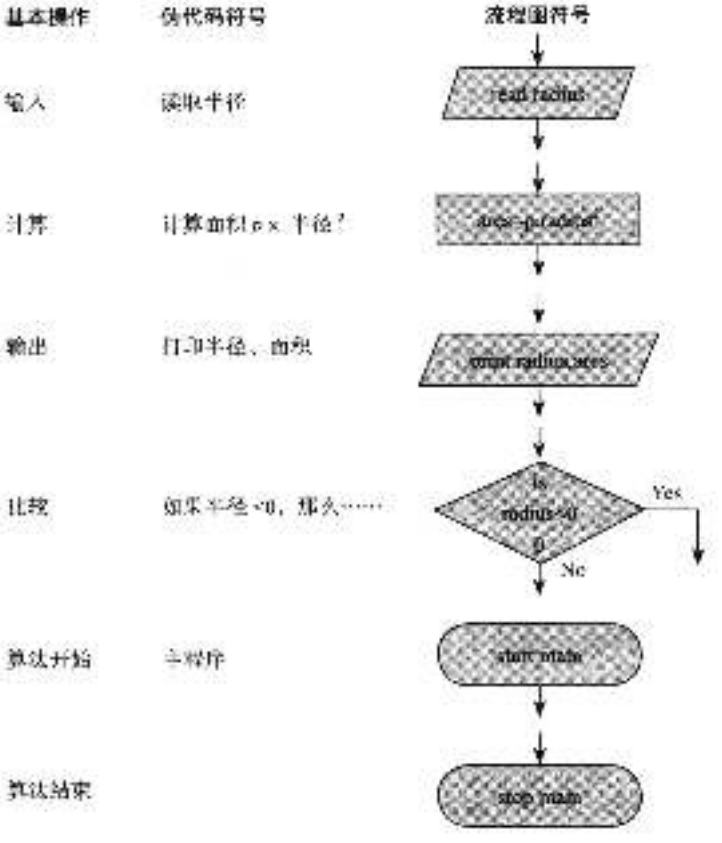


图 3.1 伪代码和流程图表示形式

伪代码和流程图是帮助我们确定解决问题的步骤顺序的工具。这两种工具都是经常使用的，但在同一个问题中一般不同时使用这两种工具。为了给出这两种工具的例子，在后面的问题解决方案中，有的使用伪代码，其他则使用流程图；选择伪代码或流程图通常都与个人有关，并无例行规则。有时为了设计复杂问题的解决方案，我们需要完成几个层次的伪代码或流程图；这就是本节前面所提到的逐步细化过程。分解提纲、伪代码和流程图是解决方案的工作模型，因此都不唯一。对于同一个解决方案，每个人都会有不同的分解提纲、伪代码或流程图描述，就像不同的人针对同一问题所开发的 C++ 程序都不同一样。

3.2 结构化编程

一个结构化的程序是一个使用简单控制结构来组织问题解决方案的程序。这里的简单结构通常定义为顺序（sequence）、选择（selection）或循环（repetition）。顺序结构包含了若干个顺次执行的步骤；选择结构则包含一组由条件真假来决定执行其中步骤的步骤集合。条件为真时某些步骤将被执行而条件为假时则执行其他步骤；重复结构包含一组在条件为真时就一直执行的步骤集合。现在我们讨论顺序和选择结构，并将使用伪代码和流程图来给出相应

的例子。循环结构将在第4章进行讨论。

在顺序结构中包含的步骤都是顺次执行的。在第2章中开发的程序都是顺序结构的。例如，图3.2中所示的计算无导管引擎飞机的速率和加速度的伪代码和流程图。

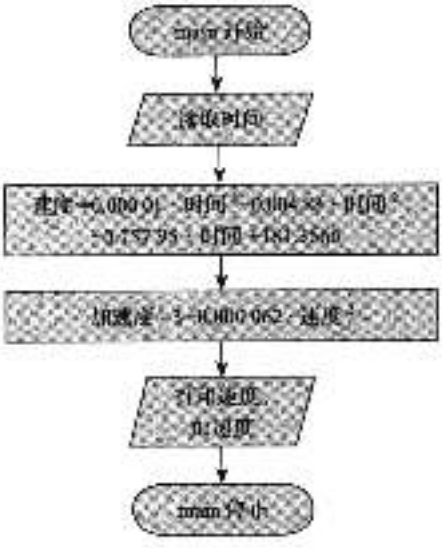


图3.2 2.8节中飞机的速度和加速度的流程图和伪代码

3.2.1 伪代码

```
main: read time
set velocity to  0.0004*time^3 - 0.0048*time^2 +
                 0.7795*time + 181.2566
set acceleration to 3 - 0.00062*velocity^2
print velocity and acceleration
```

选择结构包含了一个可以计算真假的条件。如果条件为真，某些语句将被执行；如果条件为假，那么另一些语句将被执行。例如，假定我们已知某个分数的分子和分母，在计算除法之前，我们希望知道分母是不是接近于0。因此，我们的测试条件为“分母接近于0”。如果条件为真，我们将打印消息来提示不能计算该分数的值。如果条件为假，意味着分母不接近于0，则可以计算并打印出分数的值。在定义该条件时，我们需要定义“接近于0”，对于这个例子，我们假定接近于0意味着绝对值小于0.0001。图3.3给出了这种结构的流程图表示，伪代码定义如下：

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

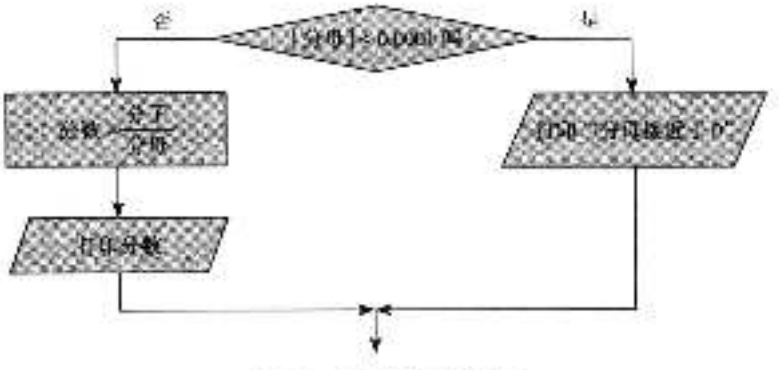


图3.3 选择结构的流程图

可以看到，在上面的结构中，当条件为假时所执行的也是一个顺序结构（计算并打印分数的值）。

在本章后面的内容中，我们将介绍完成选择结构和循环结构相关的C++语句，并使用这些结构开发示例程序。

3.2.2 可选方案的评估

对于同一个问题，通常都有多种解决方法。在大多数情况下都不存在一个最优的解决方案，但是有某些解决方案要优于其他方案。对于有经验的人而言，选择一个好的解决方案显得相对简单，在本书中我们将给出对选择好的解决方案有帮助的因素的一些例子。例如，一个好的解决方案一定是一个具有可读性的方案，因此，好的解决方案不一定是最简短的，因为简短的解决方案通常可读性都不太好。我们将努力避免使用一些精巧但难以理解的步骤来缩短程序。

当你开始设计某个问题的解决方案时，最好尝试思考多种解决方式。勾画出不同解决方案的分解提纲和伪代码或流程图，然后选择你认为最容易的一种，将其转化为C++语言。对于一些算法而言，它更适合采用某种程序语言来实现。因此你也应当选择一种适宜用C++来实现的解决方案。在某些情况下，还要考虑到其他一些因素，如执行速度和内存需求。

3.3 条件表达式

因为选择和循环结构都用到了条件（condition），因此在介绍实现选择和循环结构的语句之前，我们必须先讨论条件。条件是一个可以计算出真假的表达式，它由带有关系操作符（relational operator）的表达式组成，条件还可以包含逻辑操作符（logical operator）。本节我们将介绍关系操作符和逻辑操作符，并讨论它们出现在同一个条件中时的计算次序（evaluation order）。

3.3.1 关系操作符

下面列出了可用于比较C++表达式的关系操作符：

关系操作符	描述
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于
!=	不相等

关系操作符两边都可以使用空白，但是在由两个字符组成的操作符中间不能出现空白，如==。

下面给出了条件的示例：

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

在这些条件中，若给定了标识符的值，我们就可以计算出每个条件的真假。例如，若a等于5，b等于8.4，那么a <b为真。若x等于2.3，而y等于4.1，那么x+y >= 10.5为假。如果denominator等于20.0025，那么fabs（denominator）< 0.0001也为假。

为了增强可读性，我们在关系操作符周围添加了空格，而在算术操作符周围没有添加空格。

在C++中，一个真条件被赋予值1，而一个假条件则被赋予值0。因此下面的表达式是合法的：

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

如果 b > c，那么 d 将被赋值为 1；否则 d 的值将为 0。也可以使用单个值替换条件。例如下面的语句：

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

如果 a 的值为 0，那么条件为假；若 a 是一个非 0 值，那么条件为真。因此，在前面的语句中，若 a 非 0，那么 count 的值将自增。

3.3.2 逻辑操作符

逻辑操作符可以用于条件中，但是逻辑类作符是用来比较条件而不是比较表达式的。C++ 支持三种逻辑操作符（logical operator）：和（and）、或（or）、非（not）。逻辑类作符使用下面的符号表示：

逻辑操作符	符号
and	&&
or	||
not	!

例如，考虑下面的条件：

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

关系操作符的优先级要高于逻辑操作符，因此，这个条件读作“a 小于 b 且 b 小于 c”。为了使条件更易读，我们在逻辑操作两边加入了空格，但是没有在关系操作符两边加入空格。若给定 a、b、c 的值，我们就可以计算出条件的真假。例如，若 a 等于 1，b 等于 5，c 等于 8，那么条件为真。若 a 等于 −2，b 等于 9，c 等于 2，那么条件为假。

如果 A 和 B 是条件，那么可以使用逻辑操作符生成新的条件 A && B、A || B、!A 和 !B。当且仅当 A 和 B 同时为真时 A && B 才为真。当 A 和 B 中有一个为真时，A || B 就为真。! 操作符将条件值取反，因此，当 A 为假时，!A 为真；当 B 为假时，!B 为真。表 3.1 中对这些定义进行了总结。

表 3.1 逻辑操作符

A	B	A && B	A || B	!A	!B
假	假	假	假	真	真
假	真	假	真	真	假
真	假	假	真	假	真
真	真	真	真	假	假

表 3.1 给出了这四种条件的真值表（truth table）。

```
A && B, A || B, !A, !B
```

真值表给出了在给定了布尔操作数的值时各个条件的真假值。在表 3.1 中有两个布尔操作数 A 和 B，因此可以形成 $2^2 - 4$ 种不同的布尔操作数的值的组合。每增加一个布尔操作数，条件表达式的组合数就会翻倍。例如下面的条件：

```
A && B || B && C
```

表达式的值取决于三个布尔操作数 A、B、C 的值，因此会出现表 3.2 中所示的 2^3=8 种组合。

表 3.2

A	B	C	A && B || B && C	A	B	C	A && B || B && C
0	0	0	0	1	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	1	0	1
0	1	1	1	1	1	1	1

下面的程序说明了使用逻辑操作符和关系操作符生成表 3.2 中的真值表的过程。

```
/*-------------------------------------------------------------*/
/*  Program chapter3_1 generates a truth table                 */
/*  for the condition:                                         */
/*  A && B || B && C                                           */

#include<iostream>
using namespace std;

int main()
{
  //Declare and initialize objects
   bool A(false), B(false), C(false);

  //Print table header
  cout << " TABLE 3.2\n A\tB\tC\t\tA && B || B && C"
       << endl;
  cout << "_____"
       << endl;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;

  //Toggle C
  C = !C;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;

  //Toggle B and C
  B = !B;
  C = !C;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;

  //Toggle C
  C = !C;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;

  //Toggle A, B and C
  A = !A;
  B = !B;
  C = !C;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;

  //Repeat the pattern for B and C...

  //Toggle C
  C = !C;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;
  //Toggle B and C
  B = !B;
  C = !C;
```

```
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;

  //Toggle C
  C = !C;
  cout << A << '\t' << B << '\t' << C << "\t\t\t"
       << (A && B || B && C) << endl;
  return 0;
}
```

当带有逻辑操作符的表达式被执行时，C++ 将只计算得到表达式的值所必须要计算的部分，这称作短路（short circuiting）。例如，若 A 为假，那么 A && B 也为假，因此没有必要计算 B 的值。类似地，若 A 为真，那么 A || B 也为真，同样没有必要计算 B。为了验证表 3.2 的正确性，你必须意识到“&&”操作符的优先级高于“||”操作符。下一节我们将讨论操作符的优先级。

修改

1. 修改程序 chapter3_1，用它生成条件 A&&B&&C 的真值表。
2. 修改程序 chapter3_1，用它生成条件 A||B||C 的真值表。
3. 修改程序 chapter3_1，用它生成条件 A&&!B 的真值表。

3.3.3 优先级和结合性

一个条件可以包含多个逻辑操作符，如下所示：

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

从优先级而言，从高到低依次是 !、&&、|，但是可以使用圆括号来改变计算次序。在上面的例子中，表达式 b==c 和 b==5.5 最先被计算。假定 b 等于 3，c 等于 5，那么两个表达式都为假，因此表达式 b==c || b==5.5 为假。当对这个条件使用“!”操作符后，将得到一个真条件。

条件可以同时包含算术操作符、关系操作符以及逻辑操作符。表 3.3 中给出了条件中各个元素的优先级和结合次序。

表 3.3 算术、关系、逻辑操作符的操作优先级

优先级	操作	结合性
1	( )	与最接近的结合
2	+- -- - - !(类型)	从右向左（一元）
3	* / %	从左向右
4	+ -	从左向右
5	< <= > >=	从左向右
6	= = !=	从左向右
7	&&	从左向右
8	|	从左向右
9	= += -= *= /= %=	从右向左

练习

确定练习 1 ~ 8 中条件的真假。假定下面的对象都已经声明，且初始化如下：

a = 5.5 b = 1.5 k = 3

1. a < 10.0+k
2. a-b > =6.5
3. k!=a-b
4. b-k > a
5. !(a==3*b)
6. -k<-k-6
7. a<10&&a>5
8. fabs(k)>3 | k < b-a

3.4 选择语句：if 语句

if 语句许我们测试条件，并根据测试结果（条件为真或假）来执行相应语句。C++ 包含两种 if 语句——简单的 if 语句和 if else 语句。C++ 还包含了 switch 语句，它允许测试多个条件，根据测试条件的真假执行不同的语句。

3.4.1 简单的 if 语句

if 语句的简单形式具有下面的通用形式：

```
if (condition)
    statement 1;
```

如果条件为真，我们就执行语句 1，否则我们将跳过语句 1。

在 if 语句中所包含的语句应当缩进书写，这样有利于清楚地通过这些语句表现出程序结构。

如果我们希望在条件为真时执行多条语句（或者一个顺序结构），则应该使用语句块（statement block）。语句块是由一对大括号所包含的一组语句。大括号的书写位置也有一定的风格，常用的两种方式如下所示：

```
风格 1
if (condition)
{
   statement 1;
   statement 2;
   ...
   statement n;
}
```

```
风格 2
if (condition) {
   statement 1;
   statement 2;
   ...
   statement n;
}
```

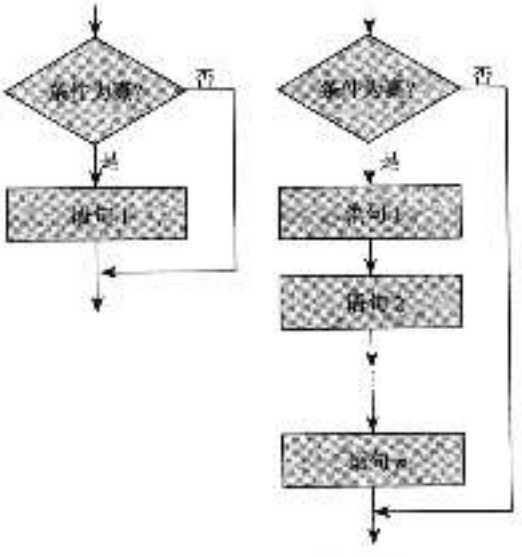


图 3.4 选择语句的流程图

在本书的解决方案中，我们采用第一种风格约定。因此，两个大括号都各自占一行。虽然这样会使程序长一些，但也更容易看出是否有括号丢失了。图 3.4 中给出了简单 if 语句控制流的流程图，包括执行单一语句和执行一个语句块两种情况。

if 语句：让语句允许我们测试条件，然后根据条件的真假确定要执行的语句。

语法

```
if (条件)
语句;
[else
语句;]
```

```
if (条件)
{
语句块;
}
[else
{
语句块;
}]
```

示例

```
if (letter && x > 0)
{
  sum += x;
  ++count;
}
```

```
if(x > y)
{
  --c1;
  --x;
}
else if(x < y)
{
  ++c2;
  --y;
}
else
{
  --c3;
}
```

关于if语句的一个例子如下所示：

```
#include<iostream>   //required for cout
using namespace std;
int main()
{
  int a, count(0), sum(0);
  cin >> a;
  if (a < 10)

     ++count;
     sum = a;

  cout << (double)sum/count;
  return 0;
}
```

如果a小于50，那么count将自增1，且a的值被加到sum中，否则这两条语句都被跳过。

if语句块中所包含的语句可以是任意的C++语句，也包括其他的if语句。下面的例子说明了这种嵌套if语句（nesting of if statement）的形式。

```
if(count < 50)
{
    ++count;
    sum = sum + a;
    if(count < 40)
    {
      ++a;
    }
}
```

若count小于50，我们将count加1，并将a加到sum上。此外，若count小于40，我们还将a的值增1。若count不小于50，我们就跳过所有语句。为了可读性要求，我们将每个if语句块中的语句都增加了缩进。

3.4.2 if/else语句

if/else语句允许我们在条件为真时执行一个语句块，而在条件为假时执行不同的语句块。if/else语句的简单形式如下所示：

```
if (condition)
   statement 1;
else
   statement 2;
```

语句1和语句2可以用多个语句块代替。语句1或语句2也可以是空语句（empty statement），仅有一个分号存在。如果语句2是空语句，那么这个if/else语句将被看作一个简单的if语句。也存在语句1为空语句的情况，但这些情况都可以被重写为简单的if语句形式。例如，下面的两个语句是等价的：

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

考虑下面的if/else语句：

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

在这个例子中，如果距离 d 小于等于 30，则速率将按照第一种方式计算，否则速率就按照第二种方式计算。图 3.5 中给出了 if/else 语句的流程图。

另一个 if/else 语句的例子是：

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

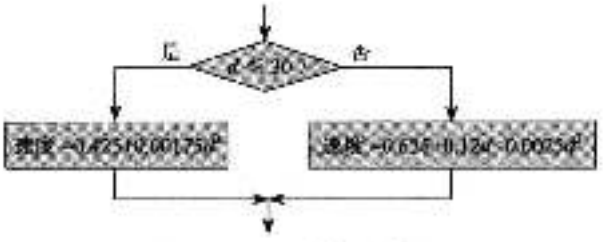
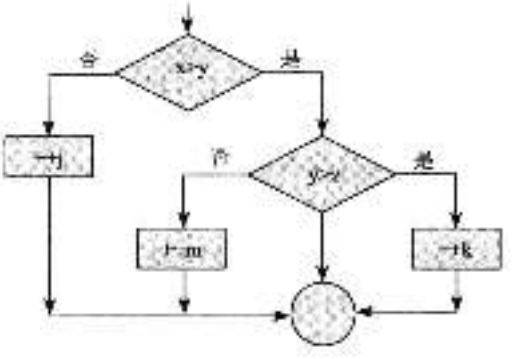


图 3.5 if/else 语句的流程图

在这个例子中，我们检查对象 denominator 的绝对值。如果这个值接近于 0，我们打印一条消息，说明我们不能完成这个除法。如果 denominator 的值不接近于 0，我们则计算并打印出 x 的值。这条语句的流程图在图 3.3 中已经给出。

考虑下面的一组嵌套 if/else 语句：

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



当 x>y 且 y<z 时，k 的值将会增加。当 x>y 且 y>=z 时，m 的值将会增加。当 x<=y 时，j 的值会增加。在认真加入缩进后，这条语句可以直接书写如下，假如我们现在去掉里层 if 语句的 else 部分，而保持相同的缩进，那么语句将会变成下面的形式：

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

这样看起来是当 x<=y 时，j 的值将增加，但这是不正确的。C++ 编译器会将 else 语句与语句块中最靠近它的 if 语句进行匹配。因此，不论如何使用缩进，前面的语句都与下面的语句执行结果一致：

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

因此，j 的值是在 x>y 且 y>=z 时增加。如果我们希望在 x<=y 时增加 j 的值，应当使用

大括号将里层的 if 语句包含起来：

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

为了避免使用 if/else 语句时出现混淆和错误，应该习惯使用大括号清晰地定义各个语句块。

C++ 允许使用条件操作符（conditional operator）来替换 if/else 语句。这个条件操作符是一个三元操作符，因为它有 3 个参数：条件、条件为真时要执行的语句、条件为假时要执行的语句。该操作符用问号将条件与执行语句隔开，问号跟在条件之后，同时使用冒号将两个执行语句隔开。为了说明实际的使用情况，可以看下面的两条等价语句：

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

条件操作符（书写为“?:”）在赋值操作符执行之前被计算，如果表达式中有一个以上的条件操作符，那么它们按照从右到左的顺序进行结合。

在本节中，我们介绍了选择语句中用于比较值的几种方式。当我们比较浮点数时需要格外小心。正如我们在第 2 章中所看到的，由于在二进制和十进制的转换中，浮点值有时与我们的预期会有细微的差别。例如，在本节的前面，我们没有将 denominator 与 0 比较，而是使用一个条件来判断 denominator 的绝对值是否小于一个较小的值。类似地，如果我们想知道 y 是否接近值 10.5，应当使用条件 fabs（y − 10.5）<= 0.001 而不是 y == 10.5。综上所述，不要对浮点值使用相等操作符。

第二点要注意的是在对整数使用相等操作符（==）时。常见的错误就是将赋值操作符（=）与逻辑操作符（==）混淆。看下面的 C++ 程序：

```
#include<iostream>
using namespace std;

int main()
{
  int x(4), y(5);
  if (x = y)
  {
     cout << x << " is equal to " << y << endl;
  }
  return 0;
}
```

编译器在编译上述代码或者操作系统执行上述程序时都不会出现错误消息，但是程序的输出结果是：

```
5 is equal to 5
```

这个 bug 的出现是因为程序中使用赋值操作符替换了 if 语句中的相等操作符。由于将 y 的值赋给了 x，使得 y 非 0，从而条件为真，于是在 if 语句块中 cout 语句被执行了。如

果 y 被初始化为 0 而不是 5，那么语句块将不会被执行。事实上每个程序员都会不止一次犯这个错误，因此如果你发现程序输出不符合你的预期，检查一下所有使用了相等操作符的条件。

练习

画出练习 1～7 中表示执行步骤的流程图，然后给出对应的 C++ 语句。假定所需的对象都已经声明且被赋予了合适的值。

1. 如果时间大于 15.0，将时间值增加 1.0。
2. 当 poly 的平方根小于 0.5 时，打印出 poly 的值。
3. 如果 volt_1 和 volt_2 之间的差超过 10.0，打印出 volt_1 和 volt_2 的值。
4. 如果 den 的值小于 0.05，将 result 置为 0；否则，将 result 的值设为 den 除以 num 的值。
5. 如果 x 的自然对数大于等于 3，则将 time 设为 0，并将 count 减 1。
6. 如果 dist 小于 50.0，且 time 大于 10.0，则将 time 的值增加 2.；否则将 time 的值增加 2.5。
7. 如果 dist 大于等于 100.0，则将 time 的值增加 2。如果 dist 的值在 50～100 之间，则将 time 的值加 1；否则，将 time 的值增加 0.5。

3.5 数值方法：线性插值

从实验中或者在观察物理现象的过程中收集数据，对设计问题解决方案而言至关重要。这些数据点通常可以被视作某个函数 $f(x)$ 上的坐标点。我们经常使用这些数据点来估计函数 $f(x)$，进而使用 $f(x)$ 来计算那些非已知数据集中的数据值。例如，假设我们有数据点 $(a, f(a))$ 和 $(c, f(c))$。如果我们想估计 $f(b)$ 的值，其中 $a < b < c$。我们可以先算出连接 $f(a)$ 和 $f(c)$ 的直线，然后使用线性插值来获得 $f(b)$ 的值。如果我们假设点 $f(a)$ 和 $f(c)$ 由一个三次多项式的图线连接，我们就可以使用三次样条插值方法来获得 $f(b)$ 的值。大多数插值问题都可以使用这两种方法之一来解决。图 3.6 包含了连接 6 个数据点的折线和三次曲线。需要清楚的是得到的函数值取决于我们所选择的插值类型。本节我们将讨论线性插值。

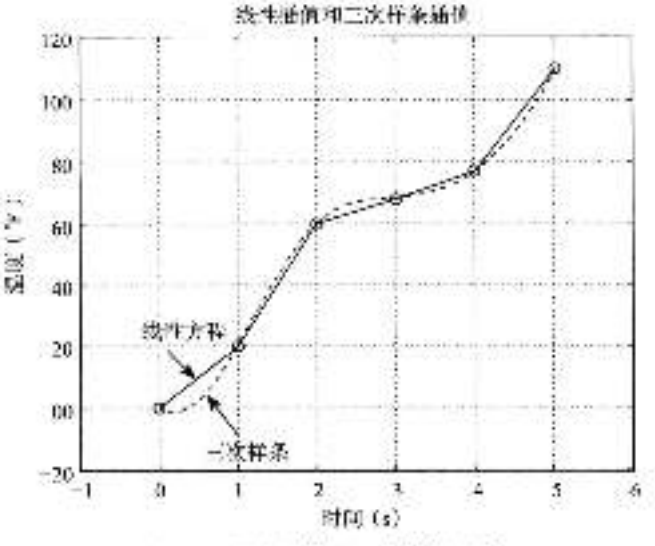


图 3.6 线性插值和三次样条插值

图 3.7 中给出了任意两个数据点 $f(a)$ 和 $f(c)$。如果我们假设两点间的函数表示是一条直线，那么可以使用来自相似三角形的公式计算任意点 $f(b)$：

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

回想一下，我们还假设了 $a < b < c$。

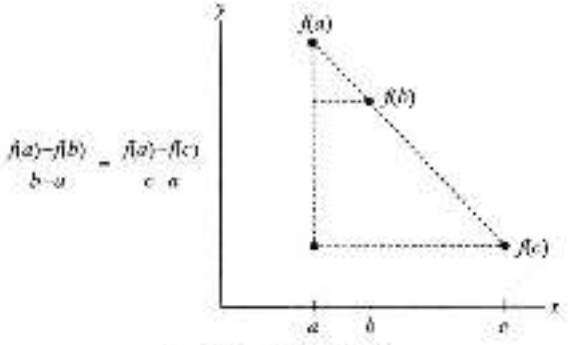


图 3.7 相似三角形

为了说明使用插值等式的公式，假设我们有一组取自用于赛车的新型引擎气缸罩的温度测量值。图 3.8 中标绘出了这些点，并使用直线将它们连接起来。具体数据列出如下：

时间（s）	温度（下）
0.0	0.0
1.0	20.0
2.0	60.0
3.0	68.0
4.0	77.0
5.0	110.0

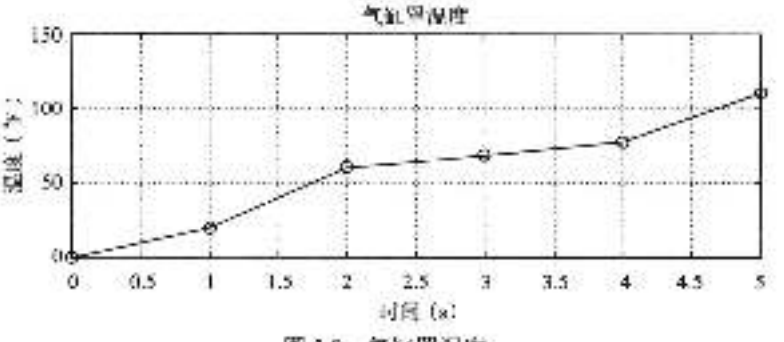


图 3.8 气缸罩温度

假如我们想计算 2.6 秒时相应的温度值，此时的情况如下：

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

使用插值公式，我们可以得到：

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

$$=60.0+\frac{2.6-2.0}{3.0-2.0}[68.0-60.0]$$

$$=64.8$$

在这个例子中，我们使用线性插值找出了对应某个特定时间的温度值。我们还可以交换温度和时间的角色，即将温度标绘在 x 轴，时间标绘在 y 轴。在这种情况下，如果我们有一组位于某个特定温度之上和之下的数据点，则可以使用相同的过程来计算对应于这个特定温度的时刻值。

练习

假设我们有下面一组数据点，它们被标绘在图 3.9 中。

时间（s）	温度（下）
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

1. 使用线性插值，用你的计算器计算下列时刻的温度值：
   0.3、1.25、2.36、4.48
2. 使用线性插值，用你的计算器计算x位于下列温度值的时刻：
   81、96、100、106
3. 假设练习 2 要求你计算对应于 110 华氏度时的时刻值。是什么让这个问题变得复杂了？对应于 110 华氏度的时刻值有多少个？使用线性插值找出每个对应的时刻值。（可以参考图 3.10，在图 3.10 中将温度标绘在 x 轴、将时刻标绘在 y 轴。）

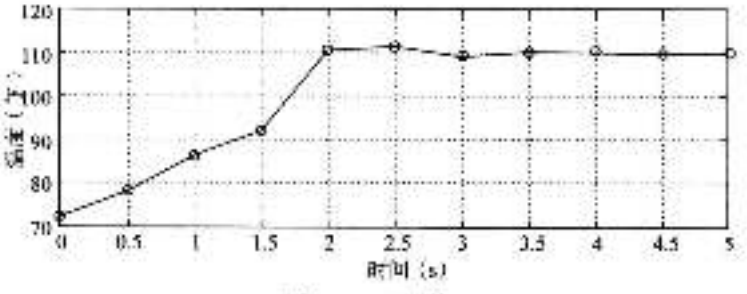


图 3.9 温度值

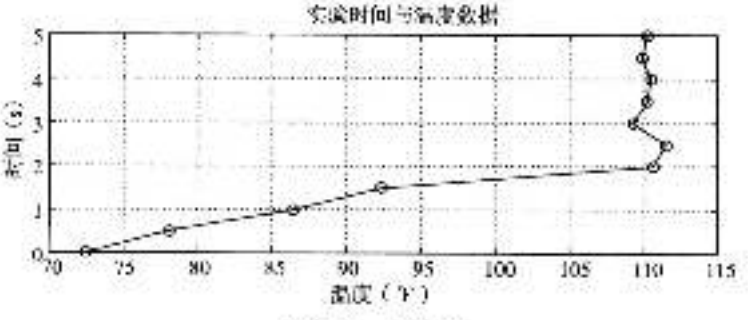


图 3.10 时刻值

3.6 解决应用问题：海水的冰点

本节我们将使用线性插值来解决一个与海水盐度相关的问题。回想一下本章开头所说的，海水盐度是海水中所溶解的矿物质的测量值。在海洋中各个地点的盐度都是不同的，但

是通常都在 33‰ ~ 38‰ 之间，或者说在 3.3% ~ 3.8% 之间。

盐度通常通过仪器测量海水的电导性来得到；水中溶解的矿物质越多，导电性越好。盐度的测量对于寒冷地区尤为重要，因为这些地区的海水冰点取决于该地区的盐度。盐度越高，海水的冰点越低。下面的表中包含了一些盐度测量值以及对应的冰点，图 3.11 中标绘出了这些值。

盐度（ppt）	冰点温度（℉）
0（淡水）	32
10	31.1
20	30.1
24.7	29.6
30	29.1
35	28.6

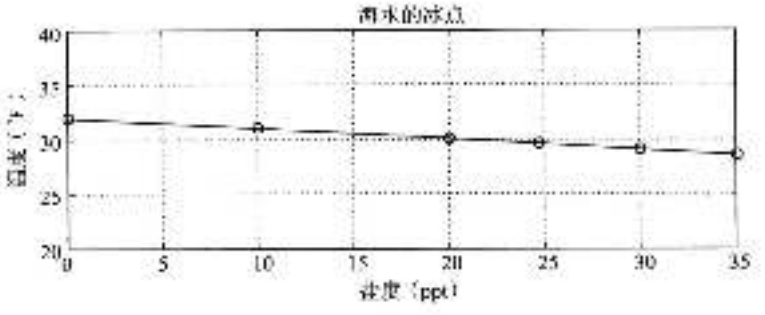


图 3.11 海水的冰点温度

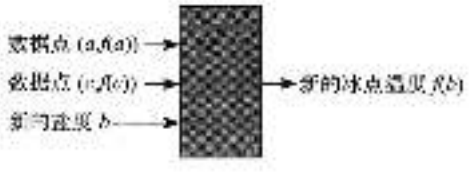
假如现在我们想使用线性插值来确定已测定过盐度值的水的冰点。写一个程序，允许用户输入两个点的数据，以及在这两个点之间的盐度值。程序应当验证所键入的盐度值是否在两个输入的数据点的盐度值之间。如果输入数据合法，那么程序应当计算出对应的冰点温度。若输入数据非法，则程序应当打印出错误消息和非法的数据。

1. 问题描述

使用线性插值计算对应于给定盐度值的新的冰点温度。

2. 输入/输出描述

下图说明程序的输入包括两个连续的点 $(a, f(a))$ 和 $(c, f(c))$，以及新的盐度值 b。输出是新的冰点温度。



3. 用例

假如我们希望计算盐度值为 33‰ 时的冰点温度。从数据中我们可以看到这个点在 33‰ 和 35‰ 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，可以计算出 $f(b)$：

$$f(b)=f(a)+(b-a)/(c-a)\cdot(f(c)-f(a))$$
$$=29.1+3/5\cdot(28.6-29.1)$$
$$=28.8$$

如同所期望的，计算出来的值落在 $f(a)$ 和 $f(c)$ 之间。

4. 算法设计

算法设计的第一步是将问题解决方案分解为一组顺序执行的步骤：

分解提纲

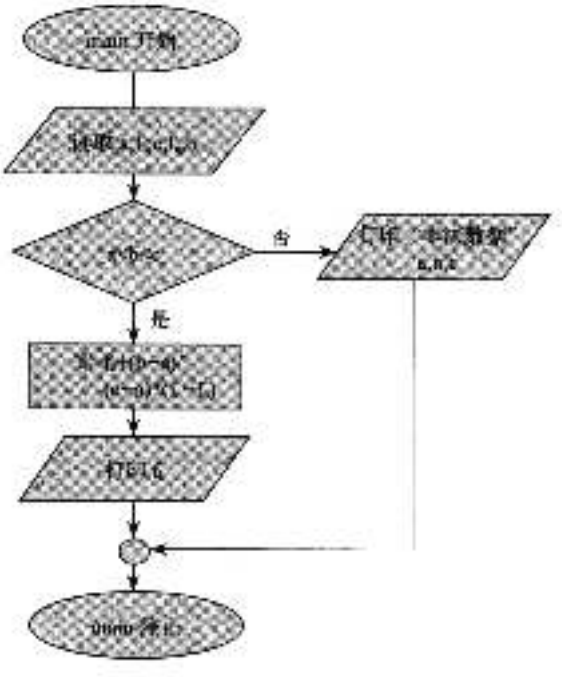
1）读取相邻的数据点和新的温度值；

2）验证输入；

3）计算新的冰点温度；

4）打印新的冰点温度。

分解提纲的第二步需要一个选择结构。如果键入的温度值不在两个数据点的温度值之间，应当打印出错误消息和错误的数据，并且终止程序。下面使用流程图对提纲步骤进行了细化。



流程图中的步骤已经很详细，可以直接转换成 C++ 语句，我们将使用一个 if 语句来实现这个选择结构。

```
/*------------------------------------------------------------------------*/
/*  Program chapter3_2                                                    */
```

```
/*                                                                    */
/*  This program uses linear interpolation to                         */
/*  compute the freezing temperature of seawater.                     */

#include<iostream> //Required for cin, cout, endl
#include<iomanip>  //Required for fixed, setprecision()
using namespace std;
int main()
{
  // Declare objects
  double a, f_a, b, f_b, c, t_c;

  // Prompt and get user input from the keyboard.
  cout << "Use ppt for salinity values." << endl
       << "Use degrees F for temperatures." << endl
       << "Enter first salinity and freezing temperature: \n";
  cin >> a >> f_a;
  cout << "Enter second salinity and freezing temperature: \n";
  cin >> b >> f_b;
  cout << "Enter new salinity: \n";
  cin >> c;
  if( c < a && c < b)
  {
     cout  << "Invalid data: " << a  << ","
           << b << "," << c << endl;
  }
  else
  {
    // Use linear interpolation to compute
    // new freezing  temperature.
    f_c = f_a + (b-a)/(c-a)*(f_c - f_a);
    // Print new freezing temperature to the screen.
    cout << " New freezing temperature in degrees F: "
         << fixed  << setprecision(1) << f_c  << endl;
  }// end else

  // Exit program.
  return 0;
}
```

5. 测试

我们首先使用用例中的数据对程序进行测试。交互过程如下：

```
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
```

计算所得的值与用例相匹配，因此我们可以使用其他的时刻值对程序进行测试。如果新的系数值与用例的结果不匹配，我们就需要检查错误是发生在用例中还是程序中。

修改

这些问题与本节中使用线性插值计算冰点的程序有关。

1. 使用程序确定与下面的盐度千分值（单位：千分之一）相对应的冰点温度：

   1　　8.5　　19　　23.5　　28.0　　30.5

2. 修改程序，使程序将冰点温度转换成摄氏度表示，并打印出来。（$T_F = 9/5T_C+32$，这里 T_F 表示华氏温度，T_C 表示摄氏温度。）
3. 假如程序所使用的数据包含的值是没有要表示的，那么需要修改程序吗？解释原因。
4. 修改程序，使程序可以使用插值的方法计算得到盐度值，而不再计算冰点值。（你可能需要参考一下图 3.12，其将冰点温度标绘在 x 轴上，将盐度值标绘在 y 轴上。）

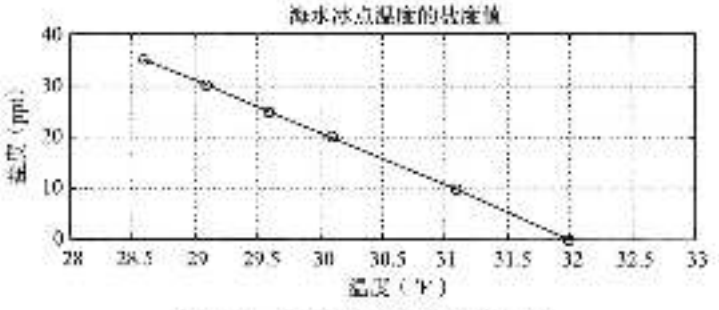


图 3.12　对应于海水冰点的盐度值

3.7　选择语句：switch 语句

switch 语句用于对多个选择进行决策。特别地，它经常用来替换嵌套的 if/else 语句。在给出有关 switch 语句的一般性讨论之前，我们先看一下使用嵌套 if/else 语句的例子，然后给出等价的 switch 语句表示。

假设我们从一个大型机械装置的传感器中读取了某个温度值。我们希望在控制台上打印出消息，以通知操作员温度状态。如果状态码为 10，温度太高，应当关闭机器；如果状态码为 11，操作员应该每 5 分钟检查一下温度；如果状态码为 13，操作员应当打开排风扇；对于所有其他的状态码，说明设备都处于正常模式。我们可以使用下面的嵌套 if/else 语句来打印正确的消息。

```
if (code == 10)
{
   cout << "Too hot - turn equipment off." << endl;
}
else
{
   if (code == 11)
   {
      cout << "Caution - recheck in 5 minutes." << endl;
   }
   else
   {
      if (code == 13)
      {
         cout << "Turn on circulating fan." << endl;
      }
      else
      {
         cout << "Normal mode of operation." << endl;
      }
   }
}
```

下面是采用 switch 的等价语句：

```
switch (code)
{
   case 10:
      cout << "You must ↑ turn equipment off." << endl;
      break;
   case 11:
      cout << "Caution - recheck in 1 minutes." << endl;
      break;
   case 13:
      cout << "Turn on circulating fan." << endl;
      break;
   default:
      cout << "Normal temperature range." << endl;
      break;
}
cout << code << endl;
```

break 语句会使程序接着执行 switch 语句之后的语句（在本例中是 cout << code << endl;），因此会跳过括号中剩下的其他语句。

将复杂的 if/else 语句转换成 switch 语句有时候并非总是很简单。但是当转换成 switch 语句之后，后者通常更易读。同样也很容易确定 switch 语句中的语句分组。

在 switch 语句中会根据控制表达式（controlling expression）来选择执行那些语句，控制表达式必须是一个整型表达式或者字符型的表达式。在下面给出的一般形式中，case 标签（label 1，label 2，…）决定了要执行哪些语句，因此在某些语言中，这种结构被称作 case 结构（case structure）。所执行的语句是标签值等于控制表达式的值所对应的语句。case 标签必须是一个值唯一的常量，如果有两个或以上的 case 标签具有相同的值就会出错。当控制语句的值不等于任何 case 标签值时，default 语句所对应的语句将会被执行；default 语句是可选的。

```
switch (controlling expression)
{
   case label_1:
      statements
   case label_2:
      statements
   ...
   default:
      statements
}
```

switch 结构中的语句通常都包含 break 语句。当执行 break 语句时，程序就会从 switch 结构的执行过程中跳出，而继续执行 switch 结构之后的语句。如果没有 break 语句，那么从所选择的标签开始以下的所有语句都将会被执行。

虽然 default 语句在 switch 结构中是可选的，但是我们推荐在结构中包含它，因为这样可以清楚地看到在没有 case 标签与控制表达式匹配时所要执行的步骤。我们还在 default 语句中使用了 break，以确保程序接下来将继续执行 switch 之后的语句。

对于相同的语句使用多个 case 标签是合法的，如：

```
switch (up code)
{
   case 'X':
   case 'B':
      cout << "Narrow operating range." << endl;
```

```
        break;
    case 'M':
        cout << "Maintenance needed." << endl;
        break;
    default:
        cout << "Error in code value." << endl;
        break;
}
```

当对同一语句使用一个以上的 case 标签时，就如何使用逻辑或操作符将多个 case 条件组合起来计算。对于这个例子而言，当 op_code 等于 'N' 或 'R' 时将执行第一条语句。

练习

将下面的嵌套 if/else 语句转换为 switch 语句。

```
if (rank==1 || rank==2)
{
    cout << "Lower division" << endl;
}
else
{
    if (rank==3 || rank==4)
    {
        cout << "Upper division" << endl;
    }
    else
    {
        if (rank==5)
        {
            cout << "Graduate student" << endl;
        }
        else
        {
            cout << "Invalid rank" << endl;
        }
    }
}
```

3.8 使用 IDE 构建 C++ 解决方案：NetBeans

在本节中，我们将使用 NetBeans 开发一个 C++ 解决方案。我们的 C++ 程序完成货币转换。进行货币转换程序的伪代码如下：

```
Pseudocode
main: read amount in dollars, currency code
      case currency code E
          convert dollars to euros
      case currency code P
          convert dollars to Mexican pesos
      case currency code S
          convert dollars to British pounds sterling
      print equivalent currency
```

伪代码中的步骤已经足够详细，可以使用 NetBeans 将其转换为 C++ 语句。

NetBeans

NetBeans 是由 Oracle 公司开发的 IDE，可以免费下载。它使用 Java 编写，只要安装了 Java 虚拟机（Java Virtual Machine，JVM）便可以运行在任何平台之上，包括 Windows、

Mac OS、Linux 以及 Solaris 等。当启动 NetBeans 时，会出现如图 3.13 所示的欢迎界面。

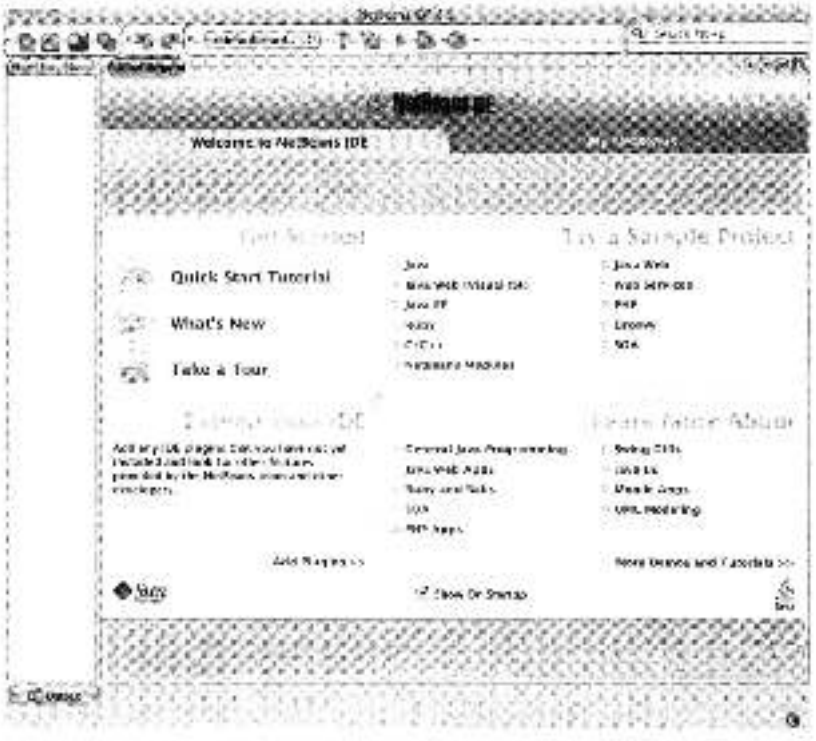


图 3.13

为了创建一个新的 C++ 解决方案，我们必须首先创建一个新工程。在欢迎界面上，选择链接“C/C++”，将会出现如图 3.14 所示的新建工程窗口。

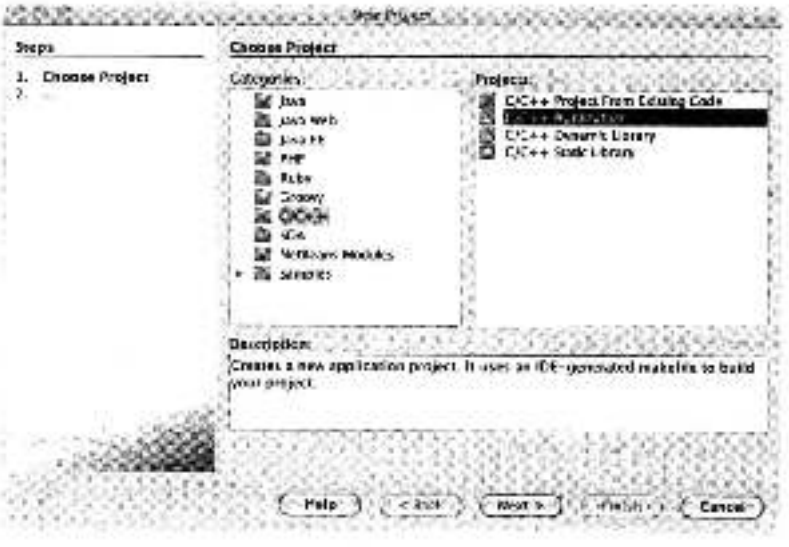


图 3.14

从 Categories 中选择 C/C++，并从 Projects 中选择 C/C++ Application，然后点击 Next 按钮。这时将会出现一个新的窗口来为工程命名并选择目录，给工程命名的窗口如图 3.15 所示。

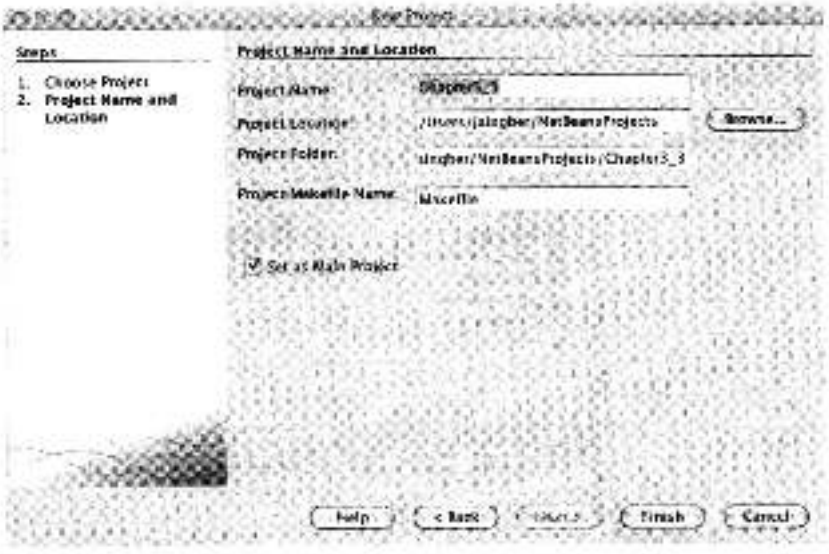


图 3.15

选定目录，并将文件命名为 Chapter3_3，然后点击 Finish 按钮。这样你的 NetBeans 工程就创建完成了。下一个出现的窗口是 Project 窗口。Project 窗口提供了访问工程文件和 NetBeans 工具的入口，如图 3.16 所示。

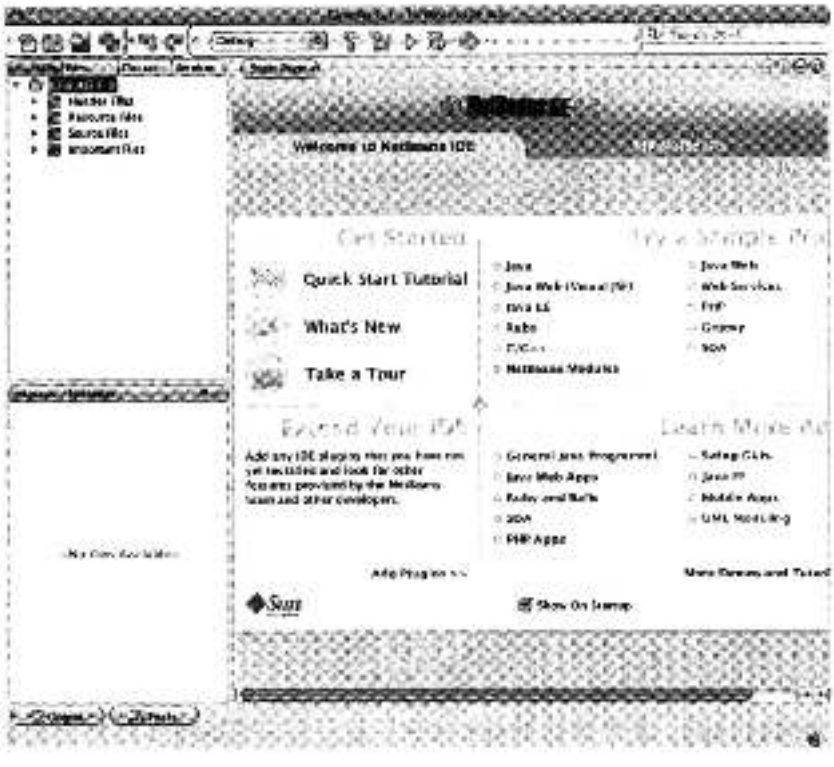


图 3.16

为了创建一个 C++ 程序，从左边的窗口中选择 Chapter3_3，然后从窗口顶部的

NetBeans File 菜单中选择 New File，这时将出现如图 3.17 所示的添加新文件的窗口。

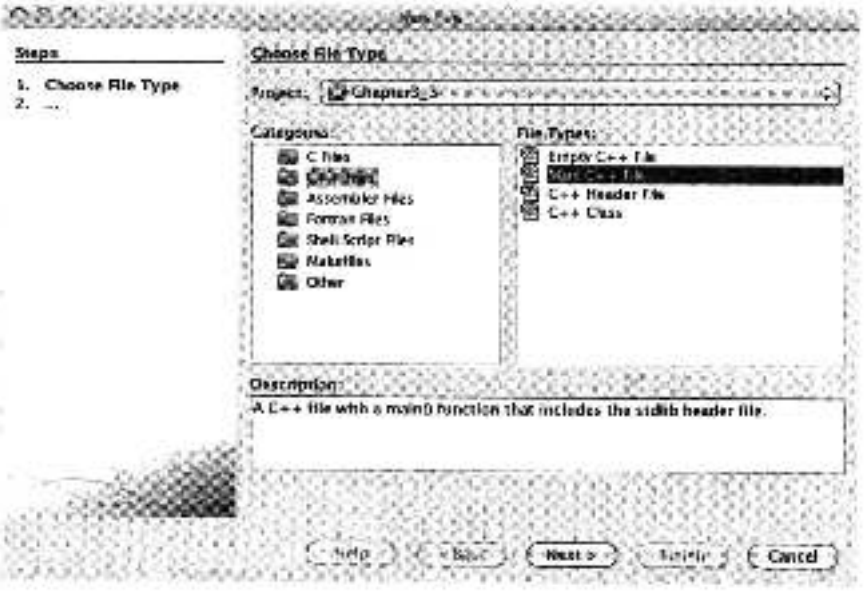


图 3.17

从 Categories 中选择 C++ Files，并从 File Types 中选择 Main C++ File，这时会出现一个新窗口，在该窗口中可以为新文件命名并选择其存放路径。新文件命名窗口如图 3.18 所示。

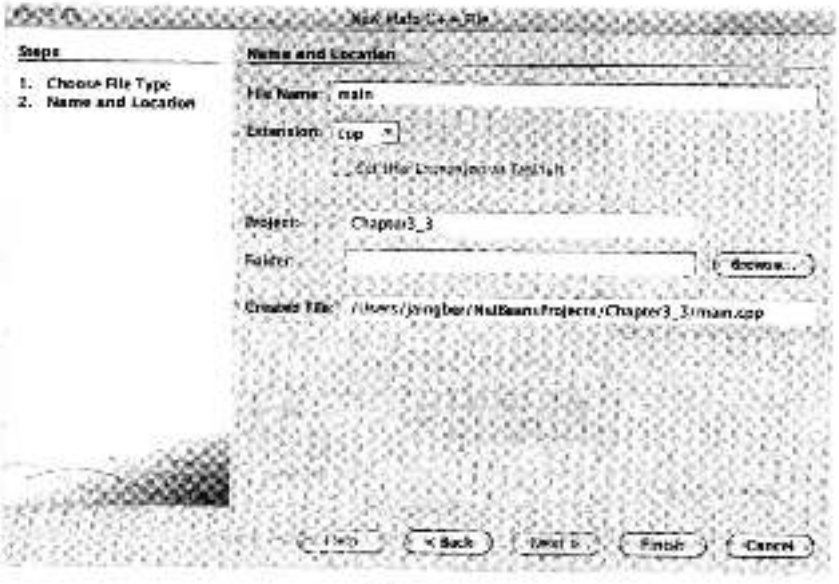


图 3.18

将文件命名为 main，然后单击 Finish 按钮，这时会出现如图 3.19 所示的编辑器窗口。我们可以看到一些基本的语句已经被添加到文件 main.cpp 中了。

现在我们在 main.cpp 中编写货币转换的 C++ 伪代码实现。代码和完成编辑后的文件窗口如图 3.20 所示。

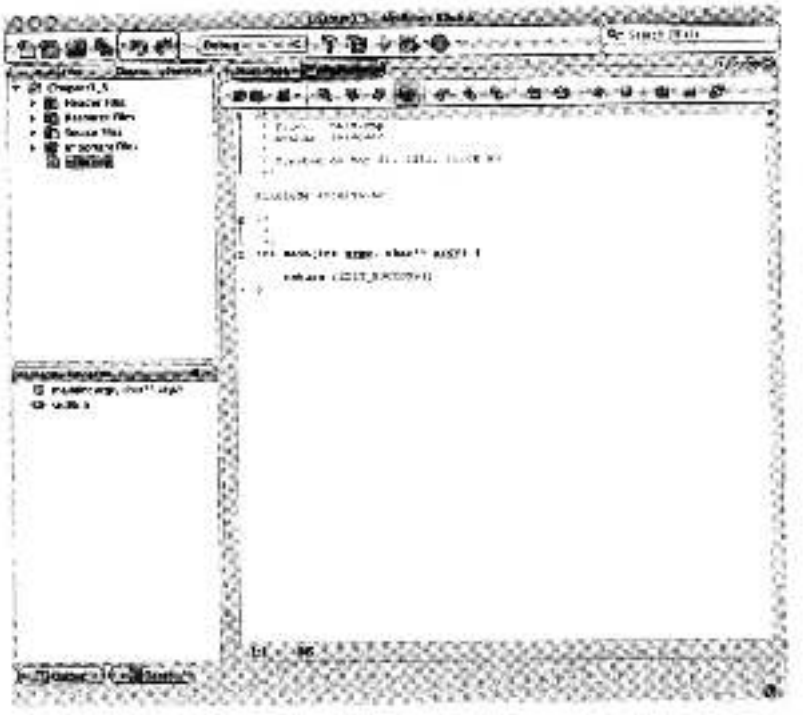


图 3.19

```
/*
 * File:   main.cpp
 * Author: jsinghe
 *
 * Created on May 21, 2011, 10:08 AM
 */
#include <stdlib.h>
#include <iostream>
using namespace std;

/*-------------------------------------------------------------------------*/
/*  Program chapter3_3
 *
 *  This program performs currency conversion from dollars to
 *  E => euros
 *  P => pesos
 *  S => pounds sterling
 */
int main(int argc, char** argv)
{
  double dollars, equivalentCost;
  char currencyCode;
  const double ECONVERTION(0.7041), PCONVERTION(11.6325),
      SCONVERTION(0.6144);
  //Prompt user for input
  cout << "enter dollar amount"  << endl;
  cin >> dollars;
  cout << "enter currency code:\n"
       << "E -> Euros\nP => Mexican Pesos\nS -> British Pounds
          Sterling\n" ;
  cin >> currencyCode;
```

```
switch(toupper(currencyCode))
{
    case 'D':
        cout << "converting dollars to euros..\n" ;
        equivalentCurr = dollars*EDOLLARS;
        break;
    case 'P':
        cout << "converting dollars to pesos..\n" ;
        equivalentCurr = dollars*PCONVERTION;
        break;
    case 'E':
        cout << "converting dollars to pounds sterling..\n" ;
        equivalentCurr = dollars*ECONVERTION;
        break;
    default:
        cout << currencyCode << "not supported at this time\n" ;
        equivalentCurr = dollars;
}
cout << "Equivalent amount: "<< equivalentCurr << endl;

return (EXIT_SUCCESS);
}
```



图 3.20

从 NetBeans Run 菜单中选择 Build Main Project 就可以对工程进行编译了。构建窗口如图 3.21 所示。我们可以看到构建成功了。



图 3.21

为了执行程序，从 NetBeans Run 菜单中选择 Run Main Project，将会出现如图 3.22 所示的输出窗口。输入要求的美元数量和货币代码，验证输出。



图 3.22

修改

1. 查找当前对欧元、墨西哥比索和英镑的汇率，并修改程序 chapter3_3 中的常量，以反映出当前对应的货币值。
2. 修改程序 chapter3_3，使其可以完成从美元到港币的兑换。
3. 修改程序 chapter3_3，将其中的 switch 语句换成 if 语句。

3.9 为自定义数据类型定义操作符

我们已经考察了算术操作符和关系操作符，并使用它们和内建数据类型一起构建条件表达式。但是，这些操作符并不是为自定义数据类型所定义的，如 Point 类型。回顾一下第 2 章的内容，只有赋值操作符是编译器为自定义数据类型定义的。如果对于其他操作符有合理的定义，我们也可以为自定义数据类型定义附加的操作符。为自定义数据类型定义操作符被称作操作符重载（operator overloading）。操作符重载为一个已经存在的操作符提供了新的定义，它允许自定义数据类型和内建数据类型一样以相同的方式使用这些操作符。只能为自定义数据类型重载操作符，不能为内建数据类型重载操作符。

本节我们将为第 2 章中的 Point 类添加两个二元操作符。我们将为 Point 类重载算术操作符（ - ），该操作符被定义为返回平面上两点间的距离；同时重载关系操作符（==），它被定义为两个操作符相应的数据成员相等时返回真。为了说明重载操作符的语法，我们在 Point 类声明中添加两个方法声明。

```
/*-------------------------------------------------------------*/
/* Point class chapter3_/                                      */
/* Filename: Point.h                                           */
/*-------------------------------------------------------------*/
class Point
{
  //Type declaration statements
  //Data members
  private:
  double xCoord, yCoord; //Class attributes

  public:
  //Declaration statements for class methods
  //Constructors for Point class
```

```
  Point(); //default constructor
  Point(double x, double y); //parameterized constructor

  //Overloaded operators
  double operator-(const Point & rhs) const;
  bool operator ==(const Point& rhs) const;
};
/*-------------------------------------------------------*/
```

注意我们在每个操作符声明前面都加上了 const 修饰符。使用 const 修饰符可以防止重载操作符修改操作数的数据成员。括号里的 const 修饰符用于保护右边操作数的数据成员。在声明末尾的 const 操作符保护左边操作数的数据成员。任何使用这些方法时尝试对对象数据成员的修改都会导致编译错误。

为了完成重载操作符的定义，必须将这两个方法添加到类实现中。更新后的类实现如下所示。

```
/*-------------------------------------------------------*/
/* Class Implementation for Point                        */
/* filename: Point.cpp                                   */
#include "Point.h"        //Required for Point
#include <iostream>       //Required for cout
#include <cmath>          //Required for sqrt() and pow()
using namespace std;

/*-------------------------------------------------------*/
/* Parameterized constructor                             */
Point::Point(double x, double y)
{
  //Input parameters x,y
  cout << "Constructing Point object, parameterized: \n";
  cout << "Input parameters: " << x << "," << y << endl;
  xCoord = x;
  yCoord = y;
}

/*-------------------------------------------------------*/
/* Default constructor                                   */
Point::Point()
{
   cout << "Constructing Point object, default: \n";
   cout << "Initializing to zero" << endl;
   xCoord = 0.0;
   yCoord = 0.0;
}
/*-------------------------------------------------------*/
/* This method returns the distance between two          */
/* points in a plane.                                    */
double Point::operator -(const Point& rhs) const
{
  double diffX, diffY, distance;
  diffX = rhs.xCoord - xCoord; //(x2-x1)
  diffY = rhs.yCoord - yCoord; //(y2-y1)
  distance = sqrt( pow(diffX,2) + pow(diffY,2) );
  return distance;
}
/*-------------------------------------------------------*/
/* This method returns:                                  */
/*   true if two points are equal                        */
/*   false if two points are not equal.                  */
```

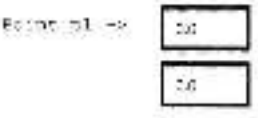
```
/* equal means that all corresponding data                    */
/* members are equal.                                         */
bool Point::operator ==(const Point& rhs) const
{
  if(rhs.xCoord == xCoord &&
     rhs.yCoord == yCoord )
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

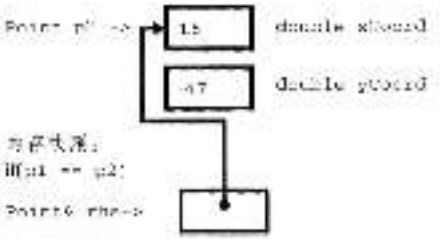
我们已经为 Point 类完成了两个二元操作符的定义。下面的程序 chapter3_3 中测试了这些新操作符。在跟踪主程序执行时，我们将使用内存快照说明重载操作符的用法。

```
/*                                                            */
/* Program chapter3_3                                         */
/*                                                            */
/* This program illustrates the use of the                    */
/* programmer-defined data type Point                         */

#include <iostream>  //Required for cout
#include "Point.h"   //Required for Point
using namespace std;

int main()
{
  //Declare and initialize objects.
  Point p1;
  Point p2(1.1, -4.4);

  //Test operators
  if( p1 == p2)
  {
    cout << "p1 is equal to p2" << endl;
  }
  else
  {
    cout << "Distance between p1 and p2 is" << p1 - p2
         << endl;
  }

  return 0;
}
/*                                                            */
```

内存快照：

```
Point p1;
Point p2(1.1, -4.4);
```





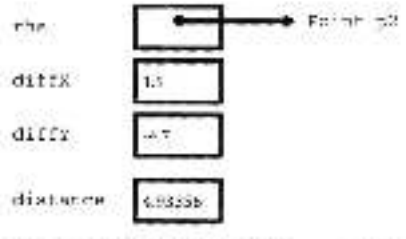
在表达式 p1 == p2 中，p1 是调用对象（calling object），调用对象是调用方法的对象。对于重载的二元操作符，左操作数总是调用对象。当 operator ==（const Point & rhs）方法被调用时，从内存快照中可以看到定义了 rhs 并引用了和 p2 所引用的相同对象，rhs 的数据类型为 Point&。操作符（&）被称作引用操作符，rhs 被称为 Point 类型的引用对象。我们将在第 6 章中详细讨论引用操作符和调用对象。对于现在而言，重要的是注意 rhs 和 p2 应用的是同一对象。

类方法可以通过名字来访问调用对象的数据成员。因此，当下面的语句执行时，

```
if(rhs.xCoord == xCoord &&
    rhs.yCoord == yCoord )
{
  return true;
}
else
{
  return false;
}
```

xCoord 和 yCoord 引用了调用对象 p1 的数据成员，因此括号内的条件表达式可以化为（1.5 == 0.0 && -4.7 == 0.0）。该表达式为假，因此在 else 语句块中的 return 语句将被执行，main( ) 中的 if 语句将返回 false。因为 main( ) 中的 if 语句返回 false，所以在 else 语句块中的 cout 语句将被执行。

```
内存快照：
cout << "Distance between p1 and p2 is " << p1 - p2
     << endl;
```



表示距离的值被返回给 cout 语句，并被打印到标准输出上。程序 chapter3_3 的一次测试运行的输出如下所示：

```
Constructing Point object, default:
Initializing to zero
Constructing Point object, parameterized:
Input parameters: 1.5, 4.7
Constructing Point object, parameterized:
```

```
input parameters: 0.0
Distance between p1 and p2 is 4.93355
```

后面我们学习关于 C++ 的新特性时，将继续对 Point 类进行设计开发。

练习

1. 创建方法 bool Point::operator == (const Point & rhs) const 的流程图。
2. 写出方法 double Point::operator - (const Point& rhs) const 的伪代码。

修改

1. 将下面的声明添加到程序 chapter3_3 中：

```
const Point ORIGIN(0.0, 0.0);
```

然后修改程序，测试 p1 和 ORIGIN 是否相等。

2. 向程序 chapter3_3 中添加下面的语句：

```
p2 = ORIGIN;
```

并加入问题 1 中关于 ORIGIN 的声明，然后修改程序，测试 p2 和 ORIGIN 是否相等。

3. 将下面的赋值语句添加到程序 chapter3_3 中：

```
ORIGIN = p2;
```

当编译程序时，会得到什么消息？

本章小结

本章我们讨论了条件的使用，以及如何通过 if 语句和 switch 语句来选择合适的语句来执行，这些选择结构在大多数程序中都被用到。

关键术语

case structure（case 结构）
condition（条件）
controlling expression（控制语句）
data file（数据文件）
decomposition outline（分解提纲）
divide and conquer（分治）
flowchart（流程图）
logical operator（逻辑操作符）
pseudocode（伪代码）
relational operator（关系操作符）
selection（选择）
sequence（顺序）
statement block（语句块）
stepwise refinement（逐步求精）
test data（测试数据）
top-down design（自顶向下设计）
validation and verification（确认和验证）

C++ 语句总结

if 语句

```
if (temp > 100)
   cout << "Temperature exceeds limit" << endl;
```

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;
```

条件语句

```
temp>100 ? cout << "Caution \n" : cout << "Normal \n";
```

switch 语句

```
switch (my_code)
{
   case 'a':
   case 'r':
      cout << "Normal Operating range \n";
      break;
   case 'x':
      cout << "Maintenance needed \n";
      break;
   default:
      cout << "Error in code value \n";
      break;
}
```

break 语句

```
break;
```

注意事项

1. 在简单的条件中的逻辑表达式里的关系操作符周围使用空格；在复杂的条件中，在逻辑操作符周围使用空格，但不要在关系操作符周围使用空格。
2. 在语句块或选择结构中对语句使用缩进。如果有复杂的嵌套语句，则对于每个嵌套的语句集合都要相对前一语句使用缩进。
3. 在复杂的语句结构中，尽量使用括号，让结构更加清晰。
4. 在 switch 语句中使用 default 语句，用以强调在没有标签被匹配时应采取的行动。

调试要点

1. 当发现并更正程序中的错误时，就开始了不断的调试步骤。特别地，应当用所有的测试数据集来重新测试程序。
2. 确保在相等的条件中使用的是关系操作符“==”而不是赋值操作符“=”。
3. 包含语句块的括号各自占一行；这将帮助你避免忽略它们。
4. 不要对浮点值使用“==”操作符；相反，要使用一个近似值与期望值进行比较。
5. 在更正错误时要不断地重新编译程序；更正一个错误可以消除很多错误消息。
6. 当调试循环时，使用 cout 语句将关键对象的内存快照打印出来。记住，使用 endl 操纵符而不要使用‘\n’，这样可以确保在语句执行之后立即将值打印出来。

习题

判断题

1. 如果条件的值为 0，那么条件将被计算为假。
2. 如果条件的值不等于 0 也不等于 1，那么这是一个不合法的条件。
3. 表达式 $a == 2$ 用于判断 a 的值是否等于 2，表达式 $a=2$ 是将值 2 赋给对象 a。
4. 逻辑操作符“&&”和“||”具有相同的优先级。
5. 关键词 else 总是与最近的 if 语句匹配，除非使用了括号来定义语句块。

语法题

给出下面语句中的语法错误。假设所有的对象都已经被定义为整数。

6.
```
switch (sqrt (x))
{
    case 1:
      cout << "Too low. \n";
      break;
    case 2:
      cout << "Correct range. \n";
      break;
    case 3:
      cout << "Too high.\n";
      break
}
```

多选题

7. 考虑下面的语句：

```
int i=100, j=0;
```

下面（　　）语句是正确的。

(a) i<3　　(b) !(j<1)

(c) (i>0) || (j>50)　　(d) (j<i) && (i <= 10)

8. 如果 a1 为真，a2 为假，那么下面（　　）语句为真。

(a) a1 && a2　　(b) a1 || a2

(c) !(a1 || a2)　　(d) !a1 && a2

9. 下面（　　）操作符是一元操作符。

(a) .

(b) ||

(c) ==

10. 表达式(!((3-4*3) < 5 && (6/4 > 3)))是（　　）。

(a) 真　　(b) 假

(c) 不合法　　(d) 以上都不是

问题 11 ~ 13 与下面的语句相关：

```
int sum(0), count(4);
if(sum <= count)
    sum += count;
cout << "sum = " << sum << endl;
```

11. 如果这些语句被执行，你将在屏幕上看到（　　）。

(a) sum = 0　　(b) sum = 4

(c) sum = 8　　(d) 没有输出

12. 在这些语句执行后，count 的值是（　　）。

(a) 0　　(b) 4

(c) 5　　(d) 一个不可预测的整数

13. 语句“sum += count;”执行了（　　）次。

(a) 0　　(b) 1

(c) 2　　(d) 4

内存快照问题

给出下面语句执行后对应的内存快照。

```
14. int a = 20;
    if(a > 0)
       if(a >= 1000)
          a = 0;
       else
          a += 2;
    else
       a += 10;
15. char ch = '+';
    switch(ch)
    {
      case '+':
         cout << "addition\n";
         break;
      case '-':
         cout << "subtraction\n";
         break;
      case '*':
         cout << "multiplication\n";
         break;
      case '/':
         cout << "division\n";
         break;
      default:
         cout << "other\n";
    }
```

布尔表达式

16. 写一个程序，从标准输入接收三个布尔变量 a、b、c 的值，计算下面条件

```
! (a&&b&&c) && !(a ||b||c)
```

的真假。以下面的格式输出结果：

```
! (xx&&xx&&xx) && !(a||b ||c)

is false (if the condition is false) OR
is true (if the condition is true).
```

17. 写一个程序，从标准输入接收三个布尔变量 a、b、c 的值，计算下面条件

```
!(a ^ b) && c
```

的真假，检查输入数据的错误，以下面的格式输出结果：

```
!(xx||xx)&&xx

is false (if the condition is false) OR
is true (if the condition is true).
```

下面的问题与本章中生成真值表的程序 chapter3_1 有关。

18. 修改程序，判断条件是否是恒真的。(提示：恒真就是总为真，如果组成条件的某一部分为假，那么这个条件就不是恒真的。)
19. 修改程序，判断条件的值是否是矛盾的。
20. 修改程序，判断条件的值是否是可变的

控制结构：循环

工程挑战：数据收集

气象气球基于从高层大气中收集数据。气球中充满了氦气，氦气的密度与周围空气密度存在差异，当气球外的空气刚好可以支持气球重量时，气球将会达到一个平衡点。白天太阳加热了气球，使它升到一个新的平衡点；晚上气球冷却，它的高度下降。气球可以监测周围空气的温度、压力、湿度和化学组成以及其他属性。在收集环境数据时，气象气球在空中可能仅停留几小时，也可能停留长达几年。当气球中的氦气耗尽或被释放后，气象气球将返回地球。

教学目标

本章我们所讨论的问题解决方案中包括：

- 循环结构：当条件为真时允许我们重复执行一组步骤的集合
- 使用流图和伪代码进行循环结构的算法设计和描述
- 三种常见的输入循环形式

4.1 算法设计

我们在第 3 章设计的算法中使用了选择结构，这可以在程序执行的过程中提供若干可选执行路径。本章中我们将使用循环结构设计算法并编写 C++ 程序。循环结构允许我们在条件为真时重复（或者循环）执行一组步骤。例如，如果我们想计算对应于时间值 0，1，2，…，10 秒的一组速率值，我们不希望开发出这样一种顺序结构，即针对时刻 0 给出一条速率计算语句，针对时刻 1、时刻 2 等再分别给出一条速率计算语句。虽然在这种情况下只需要 11 条语句，但是如果我们要计算更长时间段内的速率值的集合则可能需要数百条语句。如果我们使用循环结构，则可以设计一个解决方案，将时间初始值置为 0，在时间值不超过 10 时，我们计算并打印对应时间的速率值，同时将时间值加 1。当时间值大于 10 时，退出结构。

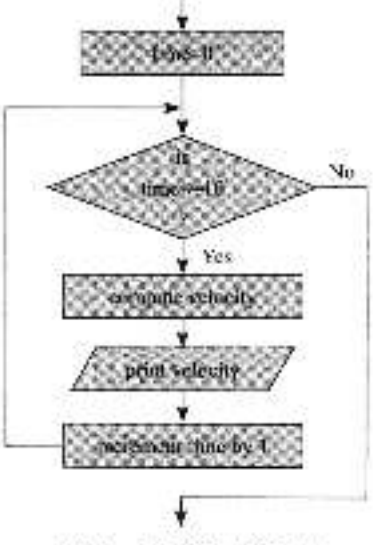


图 4.1 循环结构的流程图

伪代码和流程图描述

循环结构的算法可以用伪代码或流程图描述。图 4.1 中包含了前述计算并打印不超过时间值 10 的速率值的循环结构流程图，在每次循环结束时时间值增加 1，循环结构的伪代码描述如下：

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

循环用于实现重复结构。C++中包括三种不同的循环结构：while循环、do/while循环、for循环。C++还支持两种附加语句来扩展循环的性能：break语句（在switch语句中也用到了它）和continue语句。在本章后面，我们将介绍这些循环结构，并通过示例程序说明每种循环结构的用法。

4.2 循环结构

在C++中最常用的循环结构是while循环。

4.2.1 while循环

while循环的一般形式如下：

```
while (condition)
   statement;
```

while语句：当指定的条件为真时，while语句允许程序重复执行一个语句块。

语法

```
while(条件)
    语句;
```

```
while(条件)
{
    语句块
}
```

示例

```
while(!isspace(ch))
{
  strings += ch;
}
```

```
while(x > y)
{
  ++z;
  --x;
}
```

首先计算循环条件。如果条件为假，那么循环内的代码块将被跳过，while循环之后的语句将被执行。如果条件为真，那么循环内的代码块将被执行，并在执行后再次计算循环条件。这一过程将持续到循环条件为假时终止，图4.2给出了这一过程的流程图。

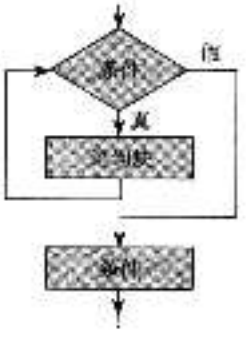


图4.2 while语句的流程图

程序chapter4_1使用while循环实现了图4.1中的流程。该程序和程序的输出结果如下

所示。

```
/*************************************************/
/*  Program chapter4_1                           */
/*  This program computes and prints a velocity  */
/*  for time values in the range of: 0 <= time <  10 */

#include<iostream> //required for cout
using namespace std;

int main()
{
  // Declare and initialize objects
  const double V0(0.0); //initial velocity m/s
  const double A(2.20); //constant acceleration m/sec
  double time, velocity;

  //print heading
  cout << " Time, s\t\tvelocity, m/s\n" ;

  time = 0;
  while(time <= 10.3)
  {
    velocity = A*time + V0;
    cout << time << "\t\t" << velocity << endl;
    ++time;
  }
  return 0;
}
```

程序输出：

```
Time, s    Velocity, m/s
0          0
1          2.337
2          5.074
3          7.811
4          10.548
5          13.285
6          16.022
7          18.759
8          21.496
9          24.233
10         26.97
```

循环中的语句必须修改在循环条件中用到的对象，否则循环条件将永远不会改变，这样会导致永远不能执行循环中的语句或者永远无法退出循环。如果在 while 循环中的循环条件总为真，那么将产生一个无限循环（infinite loop）。

大部分系统对于一个程序可以使用的执行时间总量都有一个预定义的上限，当超过这个上限时将会产生执行错误。其他系统要求用户输入一个特殊的字符序列，如控制键与字符 c 的组合（缩写为 <ctrl>-c），用以停止或中断程序的执行。几乎每个人在写程序时都会由于不注意而包含一个无限循环，因此请确保你知道系统中用来终止程序的特殊字符序列。

当你尝试在包含循环的程序中找出错误时，这里我们给出两条有用的调试建议。当编译较长的程序时，通常不会出现大量的编译错误。我们建议在每次更正一到两个明显的语法错误后就重新编译你的程序，而不是在更正所有错误后再编译。一个错误经常会产生几条错误消息。某些错误消息可能描述了一些并不在你程序中的错误，因为最原始的错误误导了编译器。

第二条建议与循环内的错误有关。当你希望确定循环内的步骤是否与你预期相符时，在循环中加入打印关键对象的内存快照的 cout 语句。如果出现了错误，你将会有大量信息用来确定是什么地方出了错。记住在 cout 语句中使用 endl 操纵符，以确保在每条调试语句之后输出缓冲区能够立即被打印到屏幕上。

下面的伪代码和程序使用 while 循环生成了一个从角度到弧度的转换表。角度值从 0 度开始，每次增加 10 度，直到 360 度。

细化的伪代码

```
main:   set degrees to zero
        while degrees <= 360
                convert degrees to radians
                print degrees, radians
                add 10 to degrees
```

C++ 程序

```
/*      ------------------------------------------------------------*/
/*  Program chapter4_2                                              */
/*  '                                                               */
/*  This program prints a degree-to-radian table                    */
/*  using a while loop structure.                                   */

#include<iostream>  //Required for cout
#include<iomanip>   //Required for setw()
using namespace std;

const double PI = 3.141593;

int main()
{
   //  Declare and initialize objects.
    +  degrees(0);
   double radians;

   //  Set formats.
   cout.setf(ios::fixed);
   cout.precision(6);

   //  Print radians and degrees in a loop.
   cout << "Degrees to Radians \n";
   while (degrees <= 360)
   {
      radians = degrees*PI/180;
      cout << setw(6) << degrees << setw(10) << radians << endl;
      degrees += 10;
   }

   //  Exit program.
   return 0;
}
/*------------------------------------------------------------------*/
```

程序输出的前几行如下：

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

为了进一步说明 while 循环，我们给出了前二次循环的程序跟踪情况和内存快照。注意 while 循环中的角度值是在角度值增加 10 之前被打印的。

程序跟踪

```
main()
步骤 1: int degrees(0);
步骤 2: double radians;
步骤 3: while(degrees <= 360)
        {
         radians = degrees*PI/180;
         cout << setw(6) << degrees << setw(10)
              << radians << endl;
         degrees += 10;
        }
```

内存快照

步骤 1：

```
integer degrees  0
```

步骤 2：

```
double radians   ?
```

步骤 3：

第一次测试为真

```
integer degrees  10
double radians   0.0
```

第二次测试为真

```
integer degrees  20
double radians   0.174533
```

第三次测试为真

```
integer degrees  30
double radians   0.349066
```

...

我们看到，degrees 在声明时被初始化为 0，因此在第一次条件测试时，条件 degrees <= 360 为真，那么 while 循环中定义的语句块将被执行。在语句块中，转换表的第一行被打印出来，degrees 的值加 10。当语句块执行完之后，控制分支返回到循环条件，进行第二次条件测试。此时条件仍为真，语句块将被执行，此时 degrees 的值为 10。这样的重复过程直到 degrees 的值为 370 时停止，因为 370 大于 360，循环条件为假，while 循环终止。while 循环终止后将使控制分支推进至后续语句块的第一条语句，在这个例子中为 "return 0;" 语句。注意图 4.2 中的直线箭头说明了与 while 循环相关的分支。

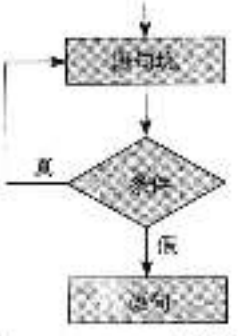


图 4.3 do/while 的流程图

4.2.2 do/while 循环

do/while 循环与 while 循环类似，不同的是它的条件测试是在循环的末尾而不是在循环开始进行，如图 4.3 所示。在循环末尾进行条件测试确保了 do/while 循环至少被执行一次，而若条件开始即为假则 while 循环可能一次都不会被执行。do/while 的一般形式如下所示：

```
do
{
   statements;
} while (condition);
```

do/while 语句： do/while 语句允许程序在指定的条件为真时重复执行一段语句。在 do/while 语句中的语句或者代块至少被执行一次。

语法

```
do                                  do
    语句;                           {
while(条件);                            语句块;
                                    }while(条件);
```

示例

```
do                                  do
    cin.get(ch);                    {
while (ch != '\n');                     x = a*i + b;
                                        cout << i << " "<< y << endl;
                                        i += 0.5;
                                    }while(i<10);
```

下面的伪代码和程序使用 do/while 循环替代 while 循环打印角度/弧度转换表：

细化的伪代码

```
main: set degrees to zero
      do
          convert degrees to radians
          print degrees, radians
          add 10 to degrees
      while degrees <= 360
```

```
/*-------------------------------------------------------------------*/
/*  Program Chapter4_3                                               */
/*                                                                   */
/*  This program prints a degree-to-radian table                     */
/*  using a do-while loop structure.                                 */

#include<iostream>  //Required for cout
#include<iomanip>   //Required for setw()

const double PI = 3.141593;

int main()
{
   //  Declare and initialize objects.
   int degrees(0);
   double radians;

   //  Set forms.
   cout.setf(ios::fixed);
   cout.precision(5);

   //  Print degrees and radians in a loop.
   cout << "Degrees to Radians \n";
   do
   {
      radians = degrees*PI/180;
      cout << setw(6) << degrees << setw(10) << radians << endl;
```

```
        degrees += 10;
    } while (degrees <= 360);

    // Exit program.
    return 0;
}
//-------------------------------------------------------------------------
```

练习

给出练习 1～4 中打印在标准输出上的输出结果。如果 cout 语句没有被执行，请说明原因。

1.
```
int count(1);
while(count < 5)
{
  ++count;
}
cout << count << endl;
```

2.
```
int count(1);
while(count < 5)
{
   count;
}
cout << count << endl;
```

3.
```
int count(10);
while(count >= 0)
{
  count -= 2;
}
cout << count << endl;
```

4.
```
int count(0);
do
{
  count = count +5;
} while (count >=10)
cout << count << endl;
```

修改

1. 修改程序 chapter4_1，使用 do/while 循环替换 while 循环。验证输出是否相同。
2. 修改程序 chapter4_1，在其中加入必要的 C++ 语句提示用户输入初始速率值和加速度常量。使用这些值计算并打印速率。
3. 修改程序 chapter4_1，在其中加入必要的 C++ 语句提示用户输入初始和终止时间的值。使用这些值计算并打印速率。

4.2.3 for 循环

许多程序都要求循环基于某个在每次循环中递增（或递减）相同数量的计数器。当计数器达到特定值时就退出循环。这种类型的循环可以使用 while 循环实现，但使用 for 循环实现则更简单。for 循环的一般形式如下：

```
for (expression_1; expression_2; expression_3)
{
   statements;
}
```

第一条语句用于初始化循环控制变量（loop-control variable），语句 2 指明了循环继续的条件，语句 3 则说明了每次循环执行后循环控制对象需要做的修改。

注意语句 1 只会执行一次。条件（语句 2）将被测试 1 到 $n+1$ 次，语句 3 将被执行 0～n 次，这里 n 是循环中语句块被执行的次数。下面给出了语法声明形式，图 4.4 给出了相应的流程图。

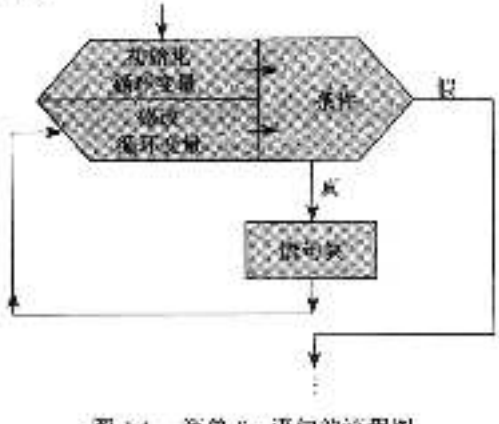


图 4.4 简单 for 语句的流程图

for 语句： for 语句允许程序基于一个计数器的值来重复执行一个语句块，该计数器的值在每次循环中都具有相同的修改量。

语法

```
for (初始化; 条件; 修改)
    语句;
```

```
for (初始化; 条件; 修改)
{
    语句块
}
```

示例

```
for (int count=1; count< 10;
++count)
    sum = sum + count;
```

```
for(int i counter; i>0; --i)
{
    cin >> degrees;
    radians = degrees * PI/180
    cout << degrees_<< " "
         << radians << endl;
}
```

例如，如果我们想执行 10 次循环，变量 k 的值从 1 ~ 10 增长，每次增加 1，我们可以使用下面的 for 循环结构：

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

在这个例子中，在第一条语句里声明并初始化了变量 k，这样 k 可以在 for 循环的语句块中被引用。

如果我们想执行与对象 n 的值相关的循环，n 的值从 20 减小至 0，每次减小 2，则可以使用下面的循环结构：

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

在这种形式中，变量 n 在第一条语句中被赋予一个初始值，但是必须在 for 语句之前被声明。这个 for 循环还可以写成下面的形式：

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

这两种形式都是合法的，但是 expression_3 一般使用缩写形式，因为这样更简洁。

下面的表达式计算了 for 循环将要执行的循环次数：

$$\text{floor}\left(\frac{\text{最终值} - \text{初始值}}{\text{增量}}\right)+1$$

如果值为负，那么循环不会执行。因此，如果 for 循环是下面的结构：

```
for (int k=5; k<=85; k+=4)
{
    statements;
}
```

那么其执行次数为：

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

这里 k 的值是 5、9、13 等，直到值增加到 81。当 k 的值为 85 时，循环将不再执行，因为此时循环条件不再为真。考虑下面嵌套的 for 循环语句：

```
for(int k=1; k<=3; ++k)
{
     for(int j=0; j<2; ++j)
     {
       ++count;
     }
}
```

外层 for 循环将执行三次。外层循环每次循环执行时，内层的 for 循环将执行两次，因此变量 count 将增加 6 次。

下面的伪代码和程序使用 for 循环打印了角度 / 弧度转换表，之前使用 while 循环也实现过。可以看到对于这个问题，使用 while 循环和使用 for 循环时的伪代码是一致的。

```
Refinement in Pseudocode
main:   set degrees to zero
        while degrees <= 360
                convert degrees to radians
                print degrees, radians
                add 10 to degrees
```

```
/*------------------------------------------------------------*/
/*  Program Chapter4_6                                        */
/*                                                            */
/*  This program prints a degree to radian table              */
/*  using a for loop structure.                               */

#include<iostream>  // Required for setw()
#include<iomanip>  // Required for cout
using namespace std;

const double PI = 3.141593;

int main()
{
   //  Declare the objects.
   double radians;

   //  Set formats.
   cout.setf(ios::fixed);
   cout.precision(6);

   //  Print degrees and radians in a loop.
   cout << "Degrees to Radians \n";
   for (int degrees=0; degrees<=360; degrees+=10)

     radians = degrees*PI/180;
     cout << setw(6) << degrees << setw(10) << radians << endl;
   }
   //  Exit program.
   return 0;
}
/*--------------------------------------------------------------*/
```

for 循环中的初始化语句和变量修改语句可以包含多条语句，在下面的 for 循环中，就对两个对象进行了初始化和更新：

```
for (int k=1, j=5; k<=10; k++, j--)
{
   sum_j += k;
   cnt_k -= j;
}
```

当使用一条以上语句时，它们之间使用逗号隔开，执行时将从左至右执行。逗号操作符（comma operator）在操作符优先级中最低，将最后执行。

练习

确定练习 1 ~ 5 中的 for 循环执行次数。

1.
```
for (int k=5; k<=20; k++)
{
statement(s);
}
```

2.
```
for (int k=5; k<=20; ++k)
{
statement(s);
}
```

3.
```
for (int count=3; count<14; count++)
{
statement(s);
}
```

4.
```
for (int k=2; k>=10; k)
{
statement(s);
}
```

5.
```
for (int time=13; time>=5; time--)
{
   statement(s);
}
```

6. 在嵌套的 for 循环执行之后 count 的值是多少？

```
int count(0);
for(int k=1; k<4; k++)
{
        for(int j=5; j>0; j--)
        {
                count++;
        }
}
```

4.3 解决应用问题：GPS

全球定位系统（Global Positioning System，GPS）由 24 颗分布在全世界上空的可提供位置、速度和时间信息的卫星组成。GPS 接收机使用最少从 4 颗卫星接收到的数据就可以精确确定接收者的位置。

为了确定 GPS 接收机的位置，从卫星广播接收到的时间信息被用来确定广播的传输时间。传输时间，或者说在卫星与接收机之间传输数据所花费的时间，用于计算卫星和接收机之间的距离，这一距离称为伪距（pseudorange）。假定数据以光速传输，那么伪距 p 被定义为：

$$p=(t_r-t_b)C$$

其中，t_r 是数据接收的时刻，t_b 是数据被广播的时刻，C 是光速，t_r-t_b 是传输时间。

卫星的位置和伪距确定了以卫星为中心、以伪距为半径的范围，如图 4.5 所示。

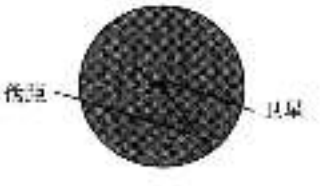


图 4.5 以卫星为中心的截面视图

GPS 接收机的位置就在这个范围的某个位置。使用从 4 颗卫星接收到的位置和时间信息，就可以通过 4 个范围的交集确定接收机的位置[9]。

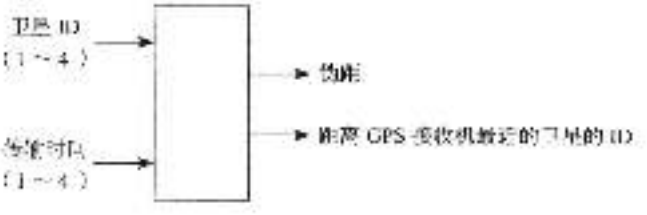
假设已知 4 颗卫星相对于 GPS 接收机的传输时间。编写一个程序提示用户输入卫星的 ID 和每颗卫星的传输时间。程序应当计算并打印出每颗卫星的范距和距离 GPS 接收机最近的卫星的 ID。

1. 问题描述

给定 4 颗卫星的 ID 和广播传输时间，计算并打印出每颗卫星的伪距。还要打印出距离 GPS 接收机最近的卫星的 ID。

2. 输入 / 输出描述

下面的草图表明程序的输入包括了 4 组数据（卫星 ID、传输时间）。输出是每颗卫星的伪距以及距离接收机最近的卫星的 ID。



3. 用例

假定第一颗卫星的 ID 为 23，其对应的传输时间为 0.001 秒；那么第一颗卫星的伪距计算应为：传输时间 *c – 299792m。

4. 算法设计

算法设计的第一步是将问题解决方案分解为一组顺序执行的步骤：

分解提纲

对于每颗卫星：

1）提示用户输入卫星 ID 和传输时间；

2）输入 ID 和传输时间；

3）计算伪距；

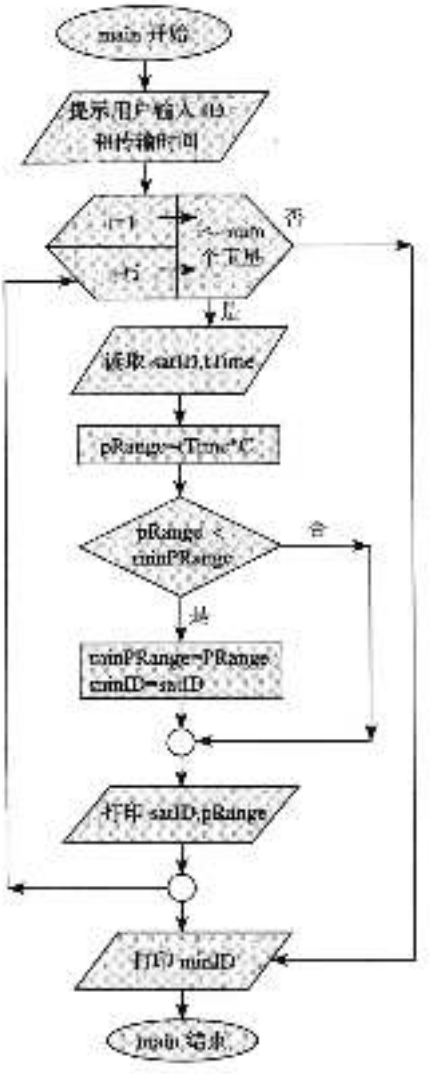
4）确定卫星是否距离接收机最近；

5）打印 ID 和伪距；

6）打印距离接收机最近的卫星 ID。

步骤 1～5 对于每颗卫星都需要重复进行，这意味着一个计数控制循环 for 循环。当我们细化算法时，将对步骤 4 予以特别的关注。为了确定哪一颗卫星距离接收机最近，我们必须追踪已经计算出的伪距中的最小值。每次计算出一个伪距，我们都要将这个值与最小伪距进行比较。如果新计算出的值更小，那么我们就更新最小伪距，并且记录对应的卫星 ID。我们将在 for 循环中嵌入行为向来实现步骤 4 中所需要的选择结构。细化后的算法流程图如下所示。

9 网址 http://en.wikipedia.org/wiki/Trilateration 上的文章 "Trileration" 给出了确定球面交集的数学生节。



流程图中的步骤已经足够详细，可以转换成 C++ 语句：

```
/*****************************************************************/
/* Program Chapter4_5                                            */
/*                                                               */
/* This program computes and prints the pseudorange for          */
/* 4 satellites.  It also prints the number of the satellite     */
/* that is closest to the receiver.                              */

#include<iostream>  //Required for cout
#include<cfloat>    //required for DBL_MAX
using namespace std;

int main()
{
  //Declare and initialize objects
  const double C(299792458.0);  //meters per second
  const int NUMBER_OF_SATELLITES(4);
  int satID, minID;
```

```
double tTime, pRange, minPRange(DBL_MAX);

//Prompt user for input
cout << "Enter id and transit time for "
     << NUMBER_OF_SATELLITES << " satellites:\n"
     << "Use whitespace to separate the values(ie: 25 0.00257)\n"
     << endl;

for(int i=1; i<=NUMBER_OF_SATELLITES; ++i)
{
  cin >> satID >> tTime;
  pRange = tTime*C;

  //Check for closest satellite
  if(pRange < minPRange)
  {
    minPRange = pRange;
    minID = satID;
  }
  cout << "Satellite " << satID << " has a pseudorange of "
       << pRange << " m\n" << endl;
}
//Output ID of closest satellite
cout << "\nSatellite " << minID
     << " is closest to GPS receiver." << endl;
return 0;
}
/*****************************************************************/
```

5. 测试

使用用例中的数据和其他三组卫星数据一起作为输入，我们得到了下面的程序输出：

```
Enter id and transit time for 4 satellites:
Use whitespace to separate the values(ie: 25 0.00257)

23 0.001
Satellite 23 has a pseudorange of 299792 m

25 0.00237
Satellite 25 has a pseudorange of 770467 m

18 0.00529
Satellite 18 has a pseudorange of 1.5859e+06 m

20 0.00176
Satellite 20 has a pseudorange of 527535 m

Satellite 23 is closest to GPS receiver.
```

对于卫星 23 所计算出的伪距与我们用例中的计算结果匹配，同时也是距离 GPS 接收机最近的卫星。

修改

这些问题与本节所开发的程序相关。

1. 修改程序，使用传输时间而不是伪距来找出距离接收机最近的卫星。
2. 修改程序，找出距离接收机最远的卫星。
3. 修改程序，处理来自 5 颗而不是 3 颗卫星的输入。

4.4 break 和 continue 语句

在前一章中，我们在 switch 语句中曾经使用过 break 语句。break 语句也可以用在本章介绍的任一循环结构中，它将从包含自身的那层循环立即退出。与之相反的是，continue 语句用于跳过当前执行的这次循环中剩下的语句，然后开始执行循环结构中的下一次循环。因此，在一个 while 循环或者 do/while 循环中，如果循环中的语句要再次执行，那么循环条件的判断是在 continue 语句执行之后进行的。在 for 循环中，循环控制对象修改时，已计算循环是否继续的条件以确定循环中的语句是否被再次执行。在出现错误的条件时，break 和 continue 语句各自在从单次循环过程或整个循环结构中跳出都是很有用的。

为了说明 break 和 continue 语句的不同，我们来看下面这个从键盘读取数值的循环：

```
sum = 0;
for (int k = 1; k < 20; ++k)
{
   cin >> x;
   if (x > 10.0)
      break;
   sum += x;
}
cout << "Sum = " << sum << endl;
```

这个循环最多从键盘读取 20 个值。如果所有 20 个值都不超过 10.0，那么这段语句就会计算出这些值的和并将其打印出来。但是如果有一个值超过 10.0，那么 break 语句就会导致控制流从循环中跳出，然后执行 cout 语句。因此，最后打印的和只是前面输入的那些不超过 10.0 的值的和。

现在看看对前面的循环做了一些变化的代码：

```
sum = 0;
for (int k=1; k<=20; ++k)
{
   cin >> x;
   if (x > 10.0)
      continue;
   sum += x;
}
cout << "Sum = " << sum << endl;
```

在这个循环中，如果所有的值都不超过 10.0，那么将打印出这 20 个值的和。但是，如果某个值大于 10.0，那么在这一次循环中 continue 语句将使控制流跳过本次循环中的剩余语句，继续执行下一次循环。因此，最后打印出的和将是这 20 个值中不超过 10.0 的值的和。

4.5 结构化输入循环

当从键盘或数据文件中读取数据时通常都会用到循环。数据文件将在第 5 章中讨论。本节我们介绍三种常见的输入形式。我们将对从标准输入读取数据时会遇到的不同的循环实现方式进行说明。

4.5.1 计数器控制循环

计数器控制（counter-controlled）循环用于在输入数据前已知所要读取的数据数量的情况。将要输入的数据数量从键盘读取并存储在一个计数器之中。

然后这个计数器将被用来控制输入循环的次数。图 4.6 中给出了这种循环结构的流程图。

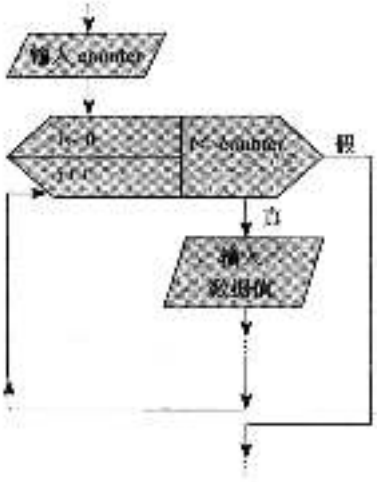


图 4.6 计数器控制循环

这种循环使用 while 循环或 for 循环实现较为简单。在下面的例子中我们使用 for 循环计算了从键盘输入的一组考试成绩的平均值：

```
/*-------------------------------------------------------------*/
/*  Program chapter4_5                                         */
/*  This programs finds the average of a set of exam scores.   */

#include<iostream>//Required for cin, cout
using namespace std;

int main()
{
// Declare and initialize objects.
   double exam_score, sum(0), average;
   int counter;

// Prompt user for input
   cout << "Enter the number of exam scores to be read ";
   cin >> counter;
   cout << "Enter " << counter << " exam scores separated "
        << " by whitespace ";

// Input exam scores using counter-controlled loop.
   for(int i=1; i<=counter; i++)
   {
      cin >> exam_score;
      sum = sum + exam_score;
   }

// Calculate average exam score.
   average = sum/counter;
   cout << counter << " student s took the exam.\n";
   cout << "The exam average is " << average << endl;

// Exit program
```

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

4.5.2 标志控制循环

标志控制循环（sentinel-controlled loop）通常用于在输入的数据中存在某个特别的数据值来标识数据的结束。这个值必须不同于正常输入的数据。这种循环结构的流程图如图 4.7 所示。

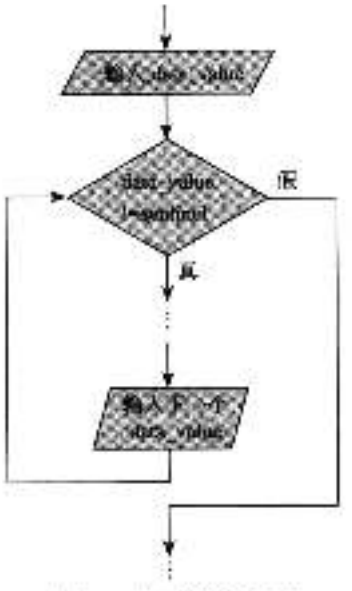


图 4.7 标志控制循环

我们将使用 while 循环来实现这种结构。为了计算一组考试成绩的平均值，我们也可以使用这种结构，在下面的例子中我们将使用负值作为标志。

```
/*-----------------------------------------------------------------*/
/*  Program Chapter4_7                                             */
/*  This programs finds the average of a set of exam scores.       */
#include<iostream>//Required for cin, cout
using namespace std;

int main()
{
// Declare and initialize objects.
   double exam_score, sum(0), average;
   int count(0);

// Prompt user for input.
   cout << "Enter exam scores separated by whitespace.\n";
   cout << "Enter a negative value to indicate the end of data. ";

// Input exam scores using sentinel controlled loop.
   cin >> exam_score;
   while(exam_score >= 0)
   {
      sum = sum + exam_score;
      ++count;
      cin >> exam_score;
   }
```

```
// Calculate average exam score.
   average = sum/count;
   cout << count << " students took the exam.\n";
   cout << "The exam average is " << average << endl;

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

4.5.3 数据终止循环

数据终止循环（end-of-data loop）是读取输入数据时最灵活的一种循环。每当有新的数据可用时，循环中的语句就会继续执行。不需要任何有关输入数据数量的先验知识，也不需要标志值。当遇到数据的末尾时，循环的执行就会终止。回想一下，cin 就是一种 istream 类型的对象，cin 可以通过 eof() 函数来判断是否已经到了数据的末尾。函数 eof() 是 istream 类的成员，如果调用对象已经读到了数据的末尾，它将返回 true。图 4.8 给出了这种循环结构的流程图。使用 while 循环可以很简单地实现数据终止循环。

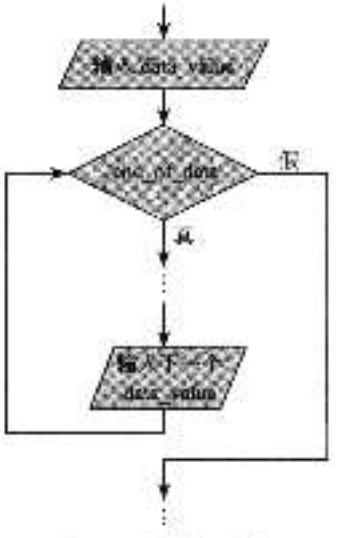


图 4.8 数据终止循环

下面的例子同样是计算从键盘输入的一组考试成绩的平均值：

```
/*-------------------------------------------------------------*/
/*  Program Chapter4_7                                         */
/*  This programs finds the average of a set of exam scores.   */

#include<iostream>//Required for cin, cout
using namespace std;
int main()
{
// Declare and initialize objects.
double exam_score, sum(0), average;
int count(0);

// Prompt user for input.
cout << "Enter exam scores separated by whitespace. ";
```

```
// Input exam scores using end of data loop.
cin >> exam_score;
while(!cin.eof())
{
   sum = sum + exam_score;
   ++count;
   cin >> exam_score;
}

// Calculate average exam score.
average = sum/count;
cout << count << " students took the exam.\n";
cout << "The exam average is " << average << endl;

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

在程序 chapter4_7 中，第一条输入语句尝试从键盘读取一条考试成绩。如果读取到数据，那么 eof() 函数将返回 false，后面的语句块将被执行。注意这里控制 while 语句的表达式里的“!”操作符。while 循环中的最后一条语句是另一条从键盘读取下一条考试成绩的输入语句。这是一个正确的数据终止控制循环。当没有读取到数据末尾时，循环将继续执行。函数 eof() 直到读取到数据末尾时才会返回 true。因此，如果没有使用正确的数据终止循环结构，在读取数据和对数据计数时常常会出错。当针对标准输入使用数据终止循环时，你需要知道在你的系统上什么样的字符序列被当做数据终止的标志。许多 Unix 和 Linux 系统将 <ctrl>+ d 字符序列作为数据终止标志。

4.6 解决应用问题：气象气球

气象气球用于在大气层的不同高度收集温度和压力数据。由于气球中的氢气比气球外的空气密度小，气球会一直上升。当气球上升时，周围的空气也变得稀薄，因此气球会上升得越来越慢，直到达到某个平衡点为止。在白天，阳光会加热气球中的氢气，这将导致氢气膨胀，密度变小，因此气球升得更高。在晚上，气球中的氢气冷却，密度变大，气球将下降到一个更低的高度。第二天太阳再次加热氢气，气球上升。在这段时间内，这个过程会产生一组高度的测量值，这组测量值可以使用一个多项式来近似。

假定这个多项式为 $H(t)=-0.12t^4+12t^3-380t^2+4100t+220$，表示在放飞气球后的首个 48 小时内的海拔或高度（单位：米），这里 t 的单位为小时。与之相应的气球速率（单位：米/秒）多项式为 $v(t)=-0.48t^3+36t^2-760t+4100$。

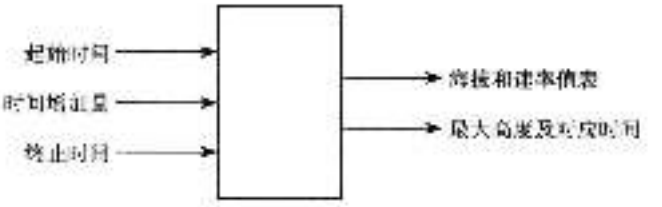
以米和米/秒为单位，打印出这个气象气球的海拔高度和速率表。让用户输入开始时间，表格每行的时间递增值，直到终止时间，这里所有的时间值必须小于 48 小时。除了打印出高度表，还需要打印出最高海拔和对应的时刻。

1. 问题描述

使用表示气球海拔高度和速率的多项式，以米和米/秒为单位，打印出海拔高度和速率表，同时找出最大高度和对应的时刻。

2. 输入/输出描述

输入包括用户输入的表格起始时间、时间增加量和终止时间。输出包括海拔和速率值的表格，以及最高海拔和对应的时刻，I/O 图如下所示。我们可以使用内建数据类型 double 表示输入、输出对象。



3. 用例

假设起始时刻是 0 小时，时间增加量为 1 小时，终止时间是 5 小时。为了得到正确的单位，我们需要将速率值除以 3600，以得到单位为米/秒的速率值。使用计算器，我们可以得到下面的值：

时间	海拔 (m)	速率 (m/s)
0	220.00	1.14
1	3951.88	0.94
2	6994.08	0.76
3	9414.28	0.59
4	11 277.28	0.45
5	12 645.00	0.32

我们也可以找出表格中最高海拔为 12 645.00 米，发生时刻为第 5 小时。

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤：

分解提纲

1）获取用户输入以确定表中的时间值；

2）生成并打印出转换表格，找出最大高度和对应的时刻；

3）打印出最大高度和对应的时刻。

分解提纲的第二步需要使用循环来生成表格，同时跟踪最大高度。当我们细化大纲时，会将步骤 2 描述得更加详细，同时还需要认真思考如何找出最大高度。回顾一下用例，一旦将表格打印出来，就能很轻松地看出最大高度。但是，在计算机计算并打印表格时，它还不能一次得到所有数据，它只拥有表格中当前行的信息。因此，为了跟踪得到最大值，我们需要使用一个独立的对象来存储最大值。每次我们计算出一个新的高度，就要将它与最大值进行比较。如果新的值更大，我们就使用新的值作为最大值。同时我们还需要跟踪对应的时刻值。下面是细化后的伪代码：

```
Refinement in Pseudocode
main: read initial, increment, final values from keyboard
        set max_height to zero
        set max_time to zero
        print table heading
        set time to initial
        while time <= final
```

```
                compute height and velocity
                print height and velocity
                if height > max_height
                        set max_height to height
                        set max_time to time
                add increment to time
        print max_time and max_height
```

伪代码中的步骤现在已经十分详细，可以转换成C++语言。注意我们在cout语句中将速率值从米/小时转换成米/秒。

```
/*                                                              */
/*  Program Chapter4_6                                          */
/*                                                              */
/*  This program prints a table of height and                   */
/*  velocity values for a weather balloon.                      */

#include <iostream>//Required for cin, cout
#include <iomanip>//Required for setw()
#include <cmath>//Required for pow()
using namespace std;

int main()
{
//  Declare and initialize objects.
   double initial, increment, final, time, height,
          velocity, max_time(0), max_height(0);
   int loops;

//  Get user input.
   cout << "Enter initial value for table (in hours) \n";
   cin >> initial;
   cout << "Enter increment between lines (in hours) \n";
   cin >> increment;
   cout << "Enter final value for table (in hours) \n";
   cin >> final;

   //  Print report heading.
   cout << "\n\nWeather Balloon Information \n";
   cout << "Time    Height    Velocity \n";
   cout << "(hrs)   (meters)  (meters/s) \n";

   // Set formats.
   cout.setf(ios::fixed);
   cout.precision(2);

   //  Compute and print report information.
   //  Determine number of iterations required.
   //  Use integer index to avoid rounding error.
   loops = (int)( (final - initial)/increment );

   for (int count=0; count<=loops; ++count)
   {
      time = initial + count*increment;
      height = -0.12*pow(time,4) + 12*pow(time,3)
               -300*time*time + 4100*time + 220;
      velocity = -0.48*pow(time,3) + 36*time*time
               - 600*time + 4100;
      cout << setw(6) << time << setw(10) << height
           << setw(10) << velocity/3600 << endl;
```

```
        if (height > max_height)
        {
            max_height = height;
            max_time = time;
        }
    }
    //  Print maximum height and corresponding time.
    cout << "\nMaximum balloon height was " << setw(8)
         << max_height << " meters \n";

    cout << "and it occurred at " << setw(6) << max_time
         << " hours \n";

    /*  Exit program.  */
    return 0;
}
/*--------------------------------------------------------------*/
```

5. 测试

如果我们使用用例中的数据，与程序的交互输出清如下所示：

```
Enter initial value for table (in hours)
0
Enter increment between lines (in hours)
1
Enter final value for table (in hours)
5

Weather Balloon Information

Time      Height       Velocity
(hrs)     (meters)     (meters/s)
0           220.00     1.14
1          3951.88     0.94
2          6894.08     0.76
3          9474.28     0.59
4         11277.28     0.45
5         12645.00     0.32 Maximum balloon height was
                       12645.00 meters, and it occurred at
                       5.00 hours.
```

图 4.9 中标绘出了 48 小时内的气球海拔和速率。从图中我们可以看到哪些时间段气球在上升或下降。

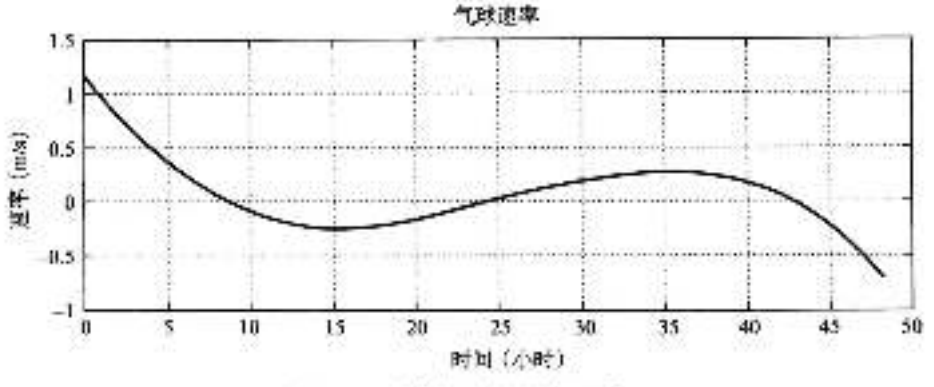


图 4.9 气象气球的海拔和速率

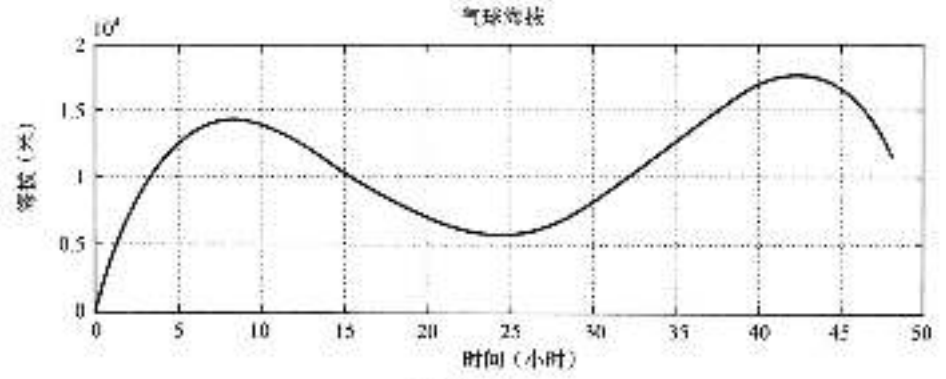


图 4.9 （续）

修改

这些问题与本节所开发的打印气象气球信息表的程序有关。

1. 使用程序生成 2 小时内每 10 分钟的气象气球信息表。起始时间从气球放飞 4 小时后开始。
2. 修改程序，在程序中添加对终止时间的检查，确保它大于起始时间。如果不大于起始时间，则要求用户重新输入完整的报告信息。
3. 本节所给的公式只是相对于时间从 0 一 48 小时有足够的准确度。修改程序，使得用户可以输入一个不超过 48 小时的时刻值对，打印相应的消息。同时，检查用户输入，确保输入在正确的范围内。如果有输入错误，要求用户重新输入完整的报告信息。
4. 如果在两个时刻都出现了相同的最大高度，则这个程序将打印出第一次出现的最大高度。修改程序，使其输出最后一次出现的最大高度。

4.7 使用 IDE 构建 C++ 解决方案：Microsoft Visual C++

在本节中我们使用 Microsoft Visual C++ 来完成一个 C++ 解决方案，用于说明嵌套控制结构的用法。我们的程序实现了一个分治算法，用于猜测一个在 1 一 20 的数字。在每次循环中，分治算法通过确定一个高的值或者低的值将数字可能处在的位置划分为两个范围，所猜测的数字总是在这个范围的中间。

当数字被猜中或者范围的高值小于低值时，程序终止。如果每个问题用户都没能回答正确，就会发生第二种情况。我们的猜数字算法的伪代码如下：

```
Pseudocode
main: print greeting to user
      while(!done && high >= low)
         guess = (high + low)/2
         print guess
         if guess equals number
            print number of trys required to guess number
            done = true
         else if guess is larger than number
            increment  number of tries
            high = guess - 1
         else if guess is smaller than number
             increment number of tries
             low  = guess + 1
      end while
```

伪代码中的步骤已经足够详细，可以转换成C++代码。我们将使用ceil()函数和自然对数函数log()来预测使用我们的分治算法将要猜测的次数。

```
/*****************************************************************/
/* Program chapter4 5                                            */
/* This program guesses a number between 1 and 20 inclusive.     */

#include<iostream>  //Required for cout, cin
#include<cmath>     //Required for ceil, log
using namespace std;

int main()
{
  // Declare and initialize objects
  int high(20), low(1), guess, count(1), ceiling;
  bool done(false);
  char answer;
  const char YES('Y'), NO('N');

  // Print greeting to user
  ceiling = ( ceil(log( (double)high ) ) + 1 );
  cout << "Think of a number between "  << low << "and"  << high
       << "and I will guess it in\n"  << ceiling
       << "or fewer trys. Just answer yes( or n(o) to my questions.\n"
       << "Are you thinking of a number? "  << endl;
  cin >> answer;
  switch(toupper(answer))
  {
  case YES:
    // While number has not been guessed
    while(!done && high >= low)
    {
      guess = (high + low)/2;
      cout << " Are you thinking of "  << guess << '?' << endl;
      cin >> answer;
      switch(toupper(answer))
      {
      case YES:
        cout << " I guessed it in "  << count << "  trys."  << endl;
        if(count > ceiling) cout << " Good pick.."  << endl;
        done = true;
        break;
      case NO:
        //Must guess again.
        ++count;
        cout << " Is "  << guess << "  Target?"  << endl;
        cin >> answer;
        if(toupper(answer) == YES) high = guess - 1;
        else low = guess + 1;
        break;  //case NO
      default:
        cout << " Don't support "  << answer << endl;
        done = true;
      } //end switch
    } //end while
    break;
  case NO:
    cout << " OK..Goodbye. "  << endl;
    break;
  default:
```

```
    cout << " Joni support " << answer << endl;
  }
  return 0;
}//end main
```

Visual C++

Visual C++是Microsoft为C和C++程序的设计开发所推出的IDE。当启动Visual C++应用程序时，将出现一个启动页面，类似图4.10所示。

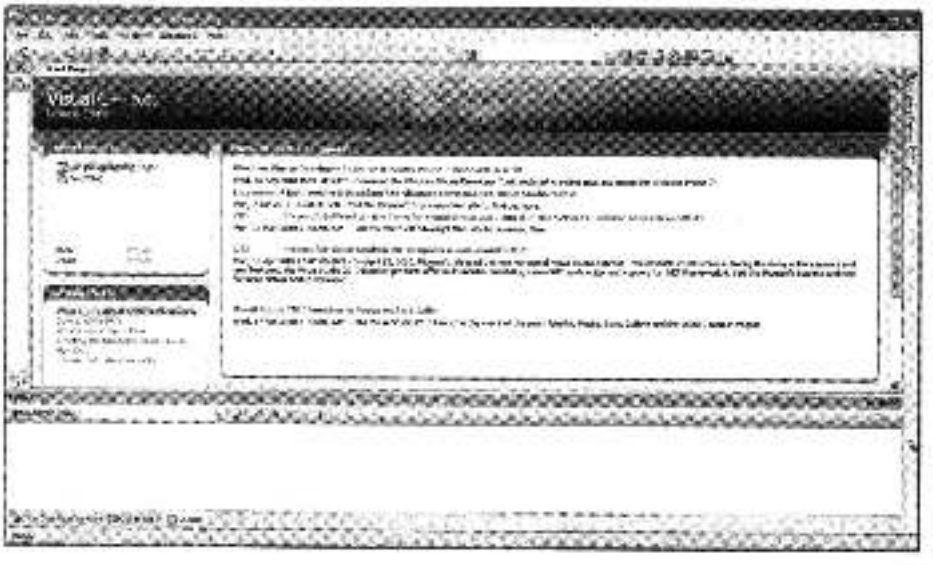


图 4.10

为了创建一个新的工程，从窗口顶端的File菜单中选择New Project，或者从Recent Projects窗口中选择Create: Project，这时将出现一个新的工程界面，如图4.11所示。

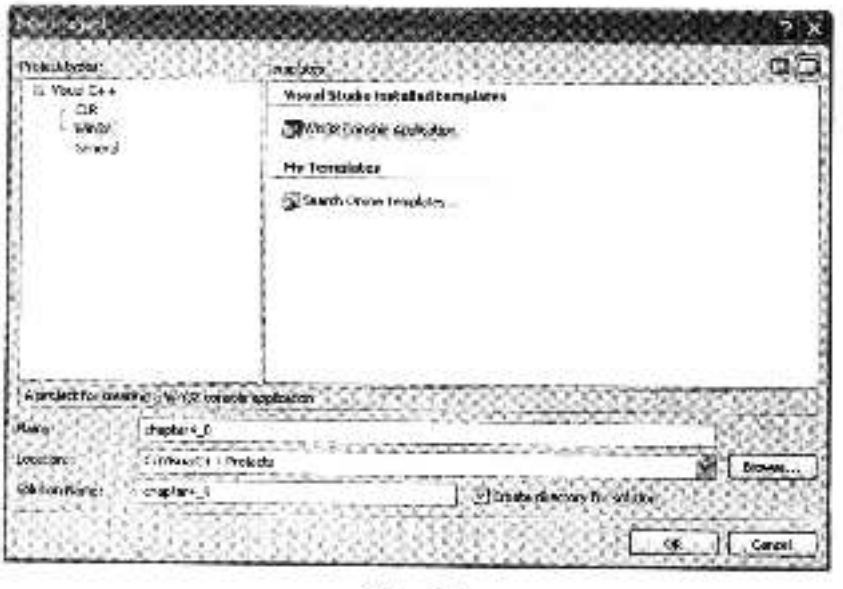


图 4.11

在本例中，我们将选择 Win32 Console Application。我们将文件命名为 chapter4_8，并点击窗口底部的 OK 按钮，这时将出现一个新的 Win32 Application Wizard 窗口，如图 4.12 所示。

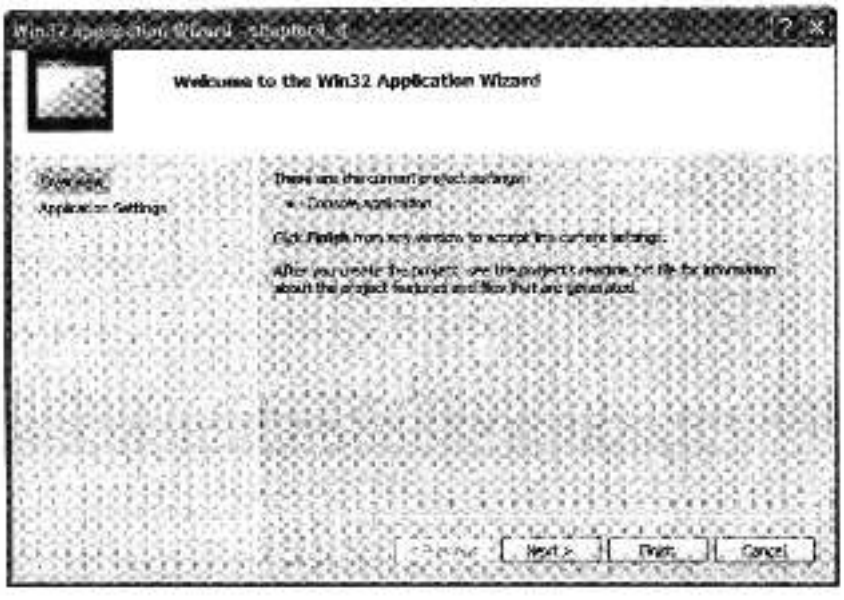


图 4.12

如果此时点击 Finish 按钮，那么将创建一个带有预编译头的新工程。为了简单起见，在本例中，我们希望创建一个空工程，而不想加入附加的头文件，因此点击窗口底部的 Next 按钮，将出现如图 4.13 所示的结束界面。

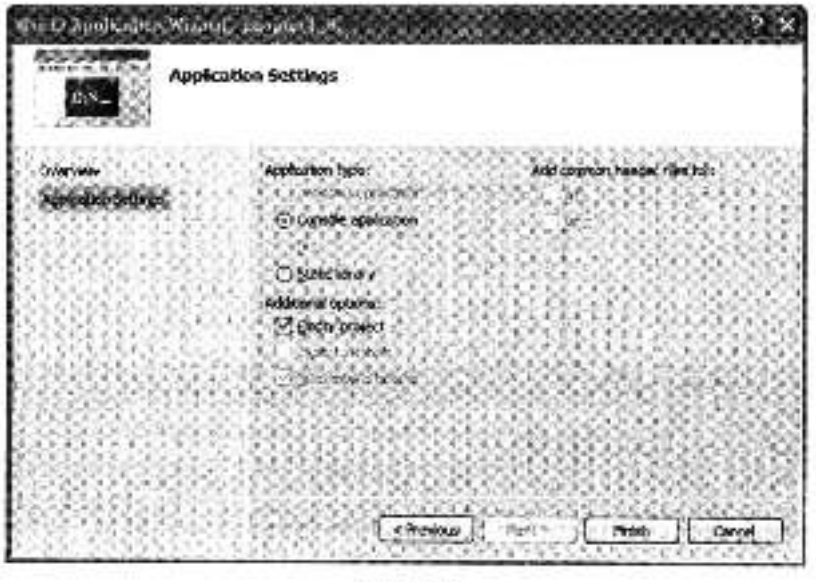


图 4.13

在图 4.13 界面中，选择 Console application 和 Empty project，然后点击 Finish 按钮。这

样就创建了一个名为 chapter4_8 的新的空工程，如图 4.14 所示，它出现在 chapter4_8 Visual C++ 工程界面的左窗口中。

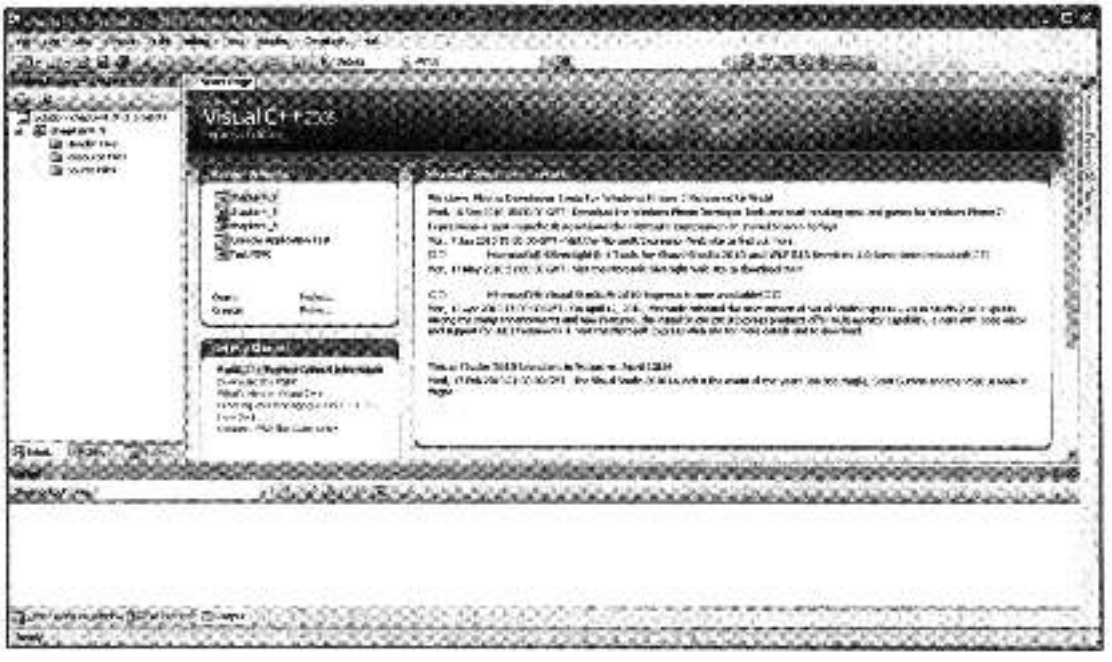


图 4.14

为了向工程中添加新文件，从窗口顶部的 Project 菜单中选择 Add New Item，将出现如图 4.15 所示的 Add New Item 窗口。



图 4.15

在左窗口中选择 Code，然后在右窗口中选择 C++ File (.cpp)，将文件命名为 main，点击 Add 按钮。这时将会打开编辑器窗口，可以输入程序 chapter4_9 的代码了，如图 4.16 所示。



图 4.16

从 Build 菜单中选择 Build Solution。如果构建成功，从 Debug 菜单中选择 Start without Debugging 以执行程序。这时会出现一个命令行窗口，在这个窗口中可以看到所有的标准 I/O。程序的运行结果在图 4.17 中可以看到。对于数字 7 只需要猜 3 次。有其他数字需要猜超过 3 次的吗？



图 4.17

Visual C++ 和 Visual Studio 集成了许多开发大型软件的工具。如果你在 Windows 环境下工作，推荐你浏览 Microsoft Visual 环境中提供的帮助材料和导学。

修改

这些问题与本节中开发的程序 chapter4_9 有关。

1. 修改程序，将所有 switch 语句替换成 if 语句。
2. 修改程序，将 while 循环替换成 do/while 循环。
3. 修改程序，使得程序可以指定范围从 1 ~ 100 的数字。

本章小结

在本章中，我们介绍了使用循环来重复执行一组语句。这些循环可以使用 while 循环、do/while 循环或者 for 循环来实现。在大多数程序中都用到了这些循环结构。

关键术语

counter-controlled loop（计数器控制循环）
divide and conquer（分治）
end-of-data loop（数据终止循环）
flowchart（流程图）
for loop（for 循环）
iteration（迭代）
loop（循环）
loop-control variable（循环控制变量）
pseudocode（伪代码）
repetition（重复）
sentinel-controlled loop（标志控制循环）
while loop（while 循环）

C++ 语句总结

while 循环

```
while (degrees <= 360)
{
   radians = degrees*PI/180;
   cout << degrees << '\t' << radians << endl;
   degrees += 10;
}
```

do/while 循环

```
do
{
   radians = degrees*PI/180;
   cout << degrees << '\t' << radians << endl;
   degrees += 10;
} while (degrees <= 360);
```

for 循环

```
for (int degrees=0; degrees<=360; degrees+=10)
{
   radians = degrees*PI/180;
   cout << degrees << '\t' << radians << endl;
}
```

break 语句

```
break;
```

continue 语句

```
continue;
```

注意事项

1. 位于语句块或者循环内部的语句要注意缩进。如果存在嵌套的循环或者复合语句，那么被嵌套的语句相对其前一层的语句要使用缩进。
2. 使用大括号标识出每个循环的循环体；每个大括号都独占一行，这样可以清楚地看出循环体。

调试要点

1. 当调试循环时，使用 cout 语句将关键对象的内存快照打印出来。记住，使用 endl 操作符而不要使用 '\n'。这样可以确保在语句执行之后立即将值打印出来。
2. 一个无限循环的形成比你想象的要简单得多，请确保你了解当出现一个无限循环时，在你的系统上用于终止其执行的特殊字符序列。

习题

判断题

1. break 语句用于立即从循环中退出。
2. continue 语句将强制开始下一次循环。
3. 在标志终止循环中需要 break 语句。
4. 使用 while 循环来实现数据终止循环较为简单。
5. 计数器控制循环常常用来统计从标准输入流取的数据的行数。
6. 为了调试循环，我们可以在程序中使用 cout 语句以类对象的内存快照。

语法题

标出下面语句中的语法错误。假设所有的对象都已经被定义为整数。

7. `for (b=1, b<25, b++)`
8. `while (k =1)`
9. `int count(0);`
   `do`
   `{`
   `cout << count << endl;`
   `++count;`
   `} while(count < 10)`

多选题

10. 考虑下面的语句：

    ```
    int i=100, j=0;
    ```

    下面（　　）语句是正确的。

    (a) i<3　　(b) i[j]<1;
    (c) (i>0) || (j>0);　　(d) (i<=) && (i <= 10)

11. 如果 a1 为真，a2 为假，那么下面（　　）表达式为真。

    (a) a, bb &&　　(b) a1 || a2
    (c) !(a1 || a2)　　(d) !a1 && a2

12. 下面（　　）操作符是一元操作符。

    (a) !
    (b) |
    (c) &&

13. 以下表达式为（　　）。

```
((((3-4)*4) < 5 && (6/4 > 3))) is
```

(a) 真　　(b) 假
(c) 不合法　　(d) 以上都不是

问题 14 ~ 16 与下面的语句相关：

```
int sum(0), count;
for (count=0; count<4; count++)
    sum += count;
cout << "sum = " << sum << endl;
```

14. 如果这些语句被执行，你将在屏幕上看到：　　）。
(a) sum = 1　　(b) sum = 6
(c) sum = 10　　(d) 4 行的输出
(e) 错误消息

15. 在 for 循环执行后，count 的值是（　　）。
(a) 0　　(b) 4
(c) 5　　(d) 一个不可预测的值

16. for 循环被执行了（　　）次。
(a) 0　　(b) 4
(c) 5　　(d) 6

编程题

单位转换。问题 19 ~ 23 要求生成单位转换表，包括一个表头和列标题。基于要打印的数值确定小数点的位置。

17. 生成一个从弧度到角度的转换表。弧度列从 0.0 开始，且逐列增加 π/10，直至达到 2π 为止。

18. 生成一个从角度到弧度的转换表。第一行包含的值为 0 度，最后一行包含的值为 360 度。允许用户输入表格中每行的增加量。

19. 生成一个从英寸到厘米的转换表。英寸的列从 0.0 开始，以 0.5 英寸为增加量。最后一行的值为 20.0 英寸。（1 英寸 = 2.54 厘米）

20. 生成一个从英里/小时到英尺/秒的转换表。英里/小时的列从 0 开始，以 5 英里/小时为增加量。最后一行为 65 英里/小时。（1 英里 =5280 英尺）

21. 生成一个从英尺/秒到英里/小时的转换表。英尺/秒的列从 0 开始，以 5 英尺/秒为增加量。最后一行为 100 英尺/秒。（1 英里 =5280 英尺）

汇率转换。问题 24 ~ 27 要求生成汇率转换表。使用表标题和列标题。假设当前的汇率转换比例如下：

```
1 美元 ($) = 9.02 墨西哥比索
1 日元 (¥) = 0.01 美元
1 美元 ($) = 11.30 南非兰特
```

22. 生成一个从比索到美元的转换表。比索的列从 5 比索开始，增加量为 5 比索。表格打印 20 行。

23. 生成一个从日元到比索的转换表。日元的列从 1 日元开始，增加量为 2 日元。表格打印 30 行。

24. 生成一个从日元到南非兰特的转换表。日元的列从 100 日元开始，打印 25 行，最后一行的值为 10 000 日元。

25. 生成一个从美元到比索、南非兰特和日元的转换表。美元的列从 1 美元开始，增加量为 1 美元。表格打印 50 行。

温度转换。问题 28 ~ 30 要求生成温度转换表，使用下面给出的各种温度（华氏度 TF、摄氏度 TC、开氏度 TK、兰氏度 TR）转换关系。

```
TF = TK - 459.67 degrees F
TF = (9/5)TC + 32 degrees F
TK = (9/5)TC
```

26. 写一个程序生成从华氏度到开氏度的转换表，范围从 0 ~ 100 华氏度，行间增加量为 5 华氏度。在你的解决方案中使用 while 循环。
27. 写一个程序生成从华氏度到开氏度的转换表，范围从 0 ~ 200 华氏度。允许用户输入行间的华氏度增加量。在你的解决方案中使用 do/while 循环。
28. 写一个程序生成从兰氏度到摄氏度的转换表。允许用户输入温度的起始范围和行间增加量。表格打印 25 行。在你的解决方案中使用 for 循环。

木材再生。在木材管理中有一个问题，就是如何确定一个地区需保留下来不能砍伐的木材量，以保证在指定的时间内，树木再生后的木材量能够重新恢复。假设植被的再生每年以某个确定的比率进行，这取决于气候和土壤条件。再生公式表示保留木材量与再生率之间的函数关系。例如，如果在收获之后留下 100 英亩未砍伐，再生率为 0.05，那么在第一年年终将有 100 + 0.05 * 100 即 105 英亩植被。在第二年年终，将有 105 + 0.05 * 105 即 110.25 英亩植被。

29. 假设总共有 14 000 英亩，2500 英亩未砍伐，再生率为 0.02。打印出一个表格，显示每年年终的植被数量，总计 20 年。
30. 修改在问题 29 中开发的程序，以允许用户输入表格所要打印的总年数。
31. 修改在问题 29 中开发的程序，以允许用户输入英亩的数值，程序应当计算出需再生达到该数值所需要的年数。

循环。这些问题与第 3 章中的生成真值表的程序 chapter3_1 有关。

32. 使用嵌套的 for 循环重写程序 chapter3_1（在第 3 章开发的）。
33. 修改程序 chapter4_5（在本章开发的），用 do/while 循环替换 for 循环。
34. 修改程序 chapter4_5（在本章开发的），用 for 循环替换 while 循环。

第5章

Engineering Problem Solving with C++, 3e

使用数据文件

工程挑战：天气预报

气象卫星为科学家和气象学家提供信息。大量数据被收集起来用于分析，历史数据则用于测试天气预报模型。通过对数据进行分析，可以分辨出与全球气候变化有关的因素，如云层、温室气体等。例如，通过了解云层对太阳能在大气层上分布的影响，可以帮助科学家更好地理解气象模式，并有助于天气预报和全球气候变化仿真工具的开发。

教学目标

本章我们所讨论的问题解决方案中包括：

- 为输入、输出打开和关闭数据文件
- 使用常用的循环结构从文件中读取数据
- 检查输入流状态
- 从输入流错误中恢复
- 使用数值方法：线性插值

5.1 定义文件流

工程问题解决方案中经常会涉及大量的数据。这些数据可能由程序生成作为输出，或者作为输入数据被程序使用。将大量的数据打印在屏幕上或者从键盘输入大量数据通常都不是明智的选择。在这些情况下，我们使用数据文件来存储数据。这些数据文件与我们存储 C++ 程序的程序文件相似。事实上，对于 C++ 编译器而言，一个 C++ 程序文件就是一个输入数据文件，而目标程序则是它的输出文件。本节我们将讨论与数据文件交互的 C++ 语句，并给出从数据文件中生成和读取信息的例子。

5.1.1 流的类层次

在第 2 章中，我们介绍了标准流对象 cin 和 cout。这些流对象都是由编译器定义用来分别与系统的标准输入、输出设备进行交互的。在本章及后续章节中我们将用到另一种标准流对象 cerr。流对象 cerr 被编译器定义用作将流输出到系统的标准错误（standard error）输出设备。cerr 是一个非缓冲的流，这意味着输出内容将被直接打印在标准错误上。缓冲流，如 cout，其中的流数据先存入一个缓冲区中，当缓冲区刷新时，其中的数据才会被显示。要使用这些标准流，在程序中必须包含头文件 iostream。

当使用数据文件时，必须定义支持文件的读（>>）、写（<<）以及向给定文件追加内容的新的流。C++ 标准库中提供了支持文件输入、输出的文件流类（file stream class）。类 ifstream 用于定义输入文件流，可以从文件中得到流数据，类 ofstream 定义了输出文件流，可以将数据流写入文件中。图 5.1 中展示了 C++ 流类层次的一部分。

图 5.1 中的直线箭头表示 ios 继承自 ios_base，istream 和 ostream 继承自 ios，ifstream

和 ofstream 分别继承自 istream 和 ostream。我们还看到 iostream 同时继承于 istream 和 ostream，这是一个多继承（multiple inheritance）的例子。

我们将在接下来的小节中讨论类 ifstream 和类 ofstream，但首先我们要说明这些类在程序 chapter5_1 中的用法。要使用类 ifstream 和 ofstream，需要在程序中包含头文件 fstream。程序 chapter5_1 中还使用了 C++ 标准库中的函数 exit()。函数 exit() 需要一个整型参数，就像函数 main() 的返回值一样，exit() 函数将其整型参数返回给系统。因此，在下面的程序中调用 exit(1) 将会终止程序，并将 1 作为 main() 的返回值。

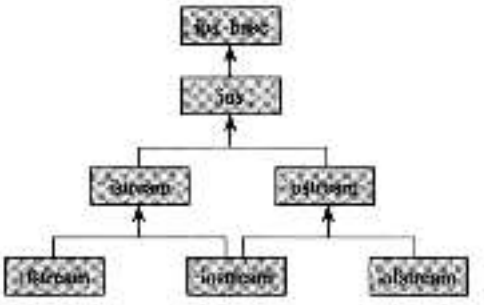


图 5.1 C++ 流类层次

```
/*-------------------------------------------------------------------*/
/* Program 5_1                                                       */
/* This program reads data pairs from the                            */
/* the file sensor.dat and writes valid data pairs                   */

/* to the file checkedSensor.dat. Valid data pairs                   */
/* may not be negative. Invalid data is written to                   */
/* to standard error(cerr)                                           */

#include<iostream> //Required for cerr
#include<fstream> //Required for ifstream, ofstream
using namespace std;

int main()
{
  //Define file streams for input and output
  ifstream fin("sensor.dat");
  ofstream fout("checkedSensor.dat");

  //Check for possible errors.
  if(fin.fail())
  {
    cerr << "could not open input file sensor.dat\n";
    exit(1);
  }
  if(fout.fail())
  {
    cerr << "could not open output file checkedSensor.dat\n";
    exit(1);
  }

  //All files are open.
  double t, motion;
  int count(0);
  fin >> t >> motion;
  while(!fin.eof())
  {
    ++count;

    //Write valid data to output file.
    if(t >= 0 && motion >= 0)
    {
       fout << t << " " << motion << endl;
    }
```

```
    //Write invalid data to standard error output.
    else
    {
       cerr << "Bad data encountered on line"
            << count << endl
            << r << " " << motion << endl;
    }

    //Input next data pair.
    fin >> r >> motion;
  }//end while
  //Close all files.
  fin.close();
  fout.close();

  return 0;
}
```

如果数据文件 sensor.dat 中包含了下面的数据：

```
0 45
1 48
2 56
3 -1
4 10
5  8
6  4
6 12
```

在这种情况下输出到标准错误的内容将打印在屏幕上，如下所示：

```
Bad data encountered on line 4
2 -1
Bad data encountered on line 6
-5 8
Bad data encountered on line 7
-6 -4
```

而数据文件 checkedSensor.dat 将包含下面的数据：

```
0 45
1 48
2 56
4 10
6 12
```

5.1.2 ifstream 类

类 ifstream 派生自类 istream，因此它继承了类 istream 的所有功能，包括输入操作符和成员函数 eof() 和 fail()，同时还有类 istream 中用来打开和关闭数据文件的成员函数。

程序中每个用于输入的数据文件都需要一个与之关联的 ifstream 对象。定义一个 ifstream 类型的对象的类型声明语句如下：

```
ifstream sensor1;
```

定义好一个 ifstream 对象后，必须将它与一个数据文件关联起来。类 ifstream 包含了一个名为 open() 的成员函数，ifstream 对象调用该函数时将文件名作为参数传递给它。这里的文件名应当是一个字符串常量或者 C 风格的字符串对象。因此，下面的语句指定了 ifstream 对象 sensor1 来操作名为 sensor1.dat 的文件，后者是我们读取信息的来源：

```
sensor1.open("sensor1.dat");
```

也可以在定义 ifstream 对象时对其进行初始化：

```
ifstream sensor1("sensor1.dat");
```

在这里当 ifstream 对象 sensor1 被创建时，open() 函数将自动被调用。

当一个数据文件用作输入时，文件本身必须存在且包含了程序所要使用的数据。如果文件不能被打开，那么将设置错误标志。此时不会产生任何错误消息，你的程序也将继续执行，但是任何从文件中读取数据的尝试都将被忽略。因此每当打开一个输入文件时，一个好的习惯就是去检查文件流对象的状态，以确保文件被成功打开了。检查文件流对象的状态的一个办法是调用对象的成员函数 fail()。如果文件打开失败，fail() 函数将返回 true，如下面的例子所示：

```
ifstream sensor1;
sensor1.open("sensor1.dat");
if( sensor1.fail() ) //open failed
{
   cerr << "File sensor1.dat could not be opened";
   exit(1); //end execution of the program
}
```

也可以直接检查文件流对象的状态：

```
ifstream sensor1("sensor1.dat");
if( !sensor1 ) //open failed
{
   cerr << "File sensor1.dat could not be opened";
   exit(1); //end execution of the program
}
```

为了表述清晰，我们通常采用第一种方式，即打开数据文件，检查文件流的状态。流状态的更多细节将在 5.5 节进行讨论。

一旦定义了输入文件流且成功地关联到一个数据文件，我们就可以像使用 cin 一样使用该文件流对象了。如果文件 sensor1.dat 中的每一行都包含一个时间和传感器值，我们就可以使用下面的语句读取一行信息，并将值分别存储到对象 t 和 motion 中：

```
sensor1 >> t >> motion;
```

注意我们使用了 sensor1 的输入操作符来从文件 sensor1.dat 中读取数值。如果数据文件中包含字符数据，则 sensor1 也可以使用 get() 函数来读取。

当从数据文件中读取信息时，将已读取信息的一部分打印出来以确保数据读取正确。

5.1.3 ofstream 类

类 ofstream 派生自类 ostream，因此它继承了类 ostream 的所有功能。同时还有类 ostream 中用于打开和关闭数据文件的成员函数。

定义一个 ofstream 类型的对象的类型声明如下：

```
ofstream balloon;
```

定义好一个 ofstream 对象后，需要将它与一个给定文件关联起来。在 ofstream 类中包含了名为 open() 的成员函数。该函数被 ofstream 对象调用，并传递一个文件名作为该函数

的参数。下面的语句表明 ofstream 对象 balloon 将会把数据写入文件 balloon.dat 中：

```
balloon.dat;
balloon.open("balloon.dat");
```

也可以在定义 ofstream 对象时对其进行初始化，如下所示：

```
ofstream balloon("balloon.dat");
```

当一个 ofstream 对象调用 open() 函数时，如果指定文件名的文件不存在，那么将创建一个以该文件名命名的新文件。如果文件已经存在，则该文件将被打开，文件中的原有内容则会被新数据所覆写。通过指定 open() 函数的第二个参数，可以使文件以追加的方式打开，即新的数据将从已有数据的末尾开始追加写入：

```
balloon.open("balloon.data", ios::app);
```

因为 ofstream 类中的 open() 函数在文件不存在时将创建一个新文件，因此在打开输出文件时很少会发生错误。基于这个理由，在后面的示例程序中打开输出文件后，我们将不检查 ofstream 对象的状态。

一旦定义了 ofstream 对象且成功地关联到一个数据文件，我们就可以像使用 cout 一样使用该 ofstream 对象了。我们可以将第 3 章所开发的计算并打印包含了时间、高度和速率数据的程序作为例子。如果我们希望修改第 3 章的程序，让它生成一个包含上述数据集的文件，我们可以使用下面的语句完成这项工作，其中的 ofstream 对象已经与输出文件 balloon.dat 相关联：

```
balloon << time << ' ' << height << ' '
        << velocity/3600 << endl;
```

空格用于分隔各个值，而 endl 操纵符用于将每组三个值写入文件后跳至新的一行。

函数 close() 用于在文件使用结束后将文件关闭；该函数由文件流对象调用。在下面的示例语句中，为了关闭两个文件，我们用到了下面的语句：

```
sensor1.close();
balloon.close();
```

如果 return 0 语句执行时文件没有关闭，那么它将自动关闭。调用 exit() 将会关闭所有的文件。

通常使用一个 string 对象来指明数据文件的名字，因为我们经常用同一个程序来处理不同的数据文件。在程序运行时，通常都需要提示用户输入数据文件的名字并将其存储在一个 string 对象中。这个字符串随后即可被 open() 使用，如下面的代码段所示：

```
...
string filename;
cout << "enter the name of the output file";
cin >> filename;
balloon.open(filename.c_str());
...
```

我们将文件名声明为一个 string 对象，这样更便于处理。但是，open() 函数要求一个 C 风格的字符串作为参数，因此 string 对象需要调用函数 c_str() 才可作为参数。函数 c_str() 是类 string 的一个成员函数，它返回一个等价于调用对象的 C 风格字符串。在本章和后续章节所有使用到文件的示例程序中，我们都将使用这一组语句。

5.2 读取数据文件

为了从数据文件中读取信息，我们必须先了解有关文件的一些细节。显而易见的是，我们要先知道文件名才能打开文件。我们必须知道存储在文件中的值的顺序和数据类型，这样才能正确地声明对应的标识符。最后，我们还需要了解文件中是否有任何特别的信息，这可以帮助我们确定文件中有多少信息。如果我们在读取完文件中所有数据之后再尝试执行一个输入语句，那么输入对象中的值将不会改变。这在我们的程序中可能导致不可预测的结果。为了避免这样的情况，我们需要知道什么时候已经读完了所有数据。

数据文件通常都是三种常见结构中的一种。一些文件生成时在第一行包含了后面信息的行数（也称作记录数）。例如，假设一个包含传感器数据的文件中有 150 组时间和传感器信息，那么在第一行只包含数值 150，之后将有 150 行包含传感器数据的行。为了从文件中读取这些数据，我们使用在第 4 章讨论过的计数器控制循环。

另一种形式的文件结构使用了尾标信号（trailer signal）或标志信号（sentinel signal）。这些信号是一些特殊的数据值，用于指示文件的最后一条记录。例如，一个带有标志信号的传感器数据文件中包含了 150 行信息，在这些信息之后是一条特殊的记录，其中时间和传感器的值均为 -999.0。这些标志信号不可以是正常数据，以避免混淆。为了从这种类型的文件中读取数据，我们使用在第 4 章讨论过的 while 循环。

第三种数据文件的结构既不包含指示后续记录行数的初始行，也不包含一个标志信号。对于这种类型的文件，我们使用文件流对象的返回值来帮助我们判断是否读取到了文件末尾。为了从这种类型的文件中读取数据，我们使用在第 4 章讨论过的数据终止循环。

现在我们将给出这样一个程序：读取传感器信息，并打印出关于信息的总结报告，这其中包含了传感器数据的数目、平均值、最大值和最小值。在下面的程序中我们将会用到上述三种不同结构的数据文件。

5.2.1 指定记录的数目

假设传感器数据文件中的第一条记录包含了一个指明后续传感器信息记录数目的整数。每条记录都包含时间和传感器数据：

```
sensor1.dat
10
0.0   132.5
0.1   147.2
0.2   140.3
0.3   137.3
0.4   163.2
0.5   158.2
0.6   169.5
0.7   148.5
0.8   137.6
0.9   135.9
```

使用一个变量控制的循环来实现我们的处理过程比较简单。处理过程的第一步是读出数据点的数目，然后使用该数目来确定要读取的时间值的数目并累积信息。在下面的程序中，第一条真实的数据值用于初始化 max 和 min 值。如果我们将 min 值初始化为 0，而所有的传感器值都大于 0，那么程序将打印出错误的最小传感器值。该解决方案的程序如下所示：

```
/*------------------------------------------------------------*/
/* Program chapter5_2                                         */
/*                                                            */
/* This program generates a summary report from               */
/* a data file that has the number of data points             */
/* in the first record.                                       */

#include <iostream> //Required for cerr, cin, cout.
#include <fstream> //Required for ifstream, ofstream.
#include <string> //Required for string.
using namespace std;

int main()
{
  // Declare and initialize objects.
  int num_data_pts, k;
  double time, motion, sum=0, max, min;
  string filename;
  ifstream sensor1;
  ofstream report;

  // Prompt user for name of input file.
  cout << "Enter the name of the input file";
  cin >> filename;

  // Open file and read the number of data points.

  sensor1.open(filename.c_str());
  if( sensor1.fail() )
  {
      cerr << "Error opening input file" << filename << endl;
      exit (1);
  }
  // Open report file.
  report.open ("sensor1report.txt");

  sensor1 >> num_data_pts;

  // Read first data pair and initial max and min.
  sensor1 >> time >> motion;
  max=min=motion;
  sum += motion;

  // Read remaining data and compute summary information.
  for (k=1; k<num_data_pts; k++)
  {
    sensor1 >> time >> motion;
    sum += motion;
    if (motion > max)
    {
       max = motion;
    }
    if (motion < min)
    {
       min = motion;
    }
  }

  // Set format flags.
  report.setf(ios::fixed);
  report.setf(ios::showpoint);
  report.precision(2);
```

```
    // Print summary information.
    report << "Number of sensor readings: "
           << num_data_pts << endl;
    report << "Average reading:           "
           << sum/num_data_pts << endl;
    report << "Maximum reading:           "
           << max << endl;
    report << "Minimum reading:           "
           << min << endl;

    // Close files and exit program.
    sensor1.close();
    report.close();
    return 0;
} //end main
//-------------------------------------------------------------------*/
```

使用文件 sensor1.dat，程序打印出来的报告如下：

```
Number of sensor readings: 10
Average reading:           149.77
Maximum reading:           160.30
Minimum reading:           132.50
```

5.2.2 标志信号

假设数据文件 sensor2.dat 包含的信息与 sensor1.dat 相同，但在文件开头并未给出合法数据记录的数量，而是最后一条记录中包含了标志信号。在文件最后一行中包含的时间值为负数，这样可以表明它不是一个合法的记录。数据文件的内容如下：

```
sensor2.dat
0.0  152.5
0.1  147.2
0.2  148.9
0.3  157.9
0.4  161.2
0.5  158.2
0.6  159.9
0.7  158.2
0.8  157.6
0.9  155.9
-99  -99
```

使用一个 do/while 结构的循环可以比较容易地描述出我们的处理过程：读取并累积信息直到读到标志信号位置。下面给出了相应的伪代码和程序段：

```
Pseudocode
main:    set sum to zero
         set number of points to 0
         read time, motion
         set max to motion
         set min to motion
         do
                  add motion to sum
                  if motion > max
                           set max to motion
                  If motion < min
                           set min to motion
                  increment number of points by 1
                  read time, motion
```

```
        while time >= 0
        set average to sum/number of data points
        print average, max, min

/*-------------------------------------------------------------*/
/*  Program chapter5_3                                         */
/*                                                             */
/*  This program generates a summary report from               */
/*  a data file that has a trailer record with                 */
/*  negative values.                                           */

#include <iostream>  //Required for cin, cout, cerr
#include <fstream>  //Required for ifstream, ofstream
#include <string>  //Required for string.
using namespace std;

int main()
{
   //  Declare and initialize objects.
   int num_data_pts(0), k;
   double time, motion, sum(0), max, min;
   string filename;
   ifstream sensor2;
   ofstream report;

   //  Prompt user for name of input file.
   cout << "Enter the name of the input file";
   cin >> filename;

   //  Open file and read the first data point.
   sensor2.open(filename.c_str());
   if(sensor2.fail())
   {
        cerr << "Error opening input file\n";
        exit(1);
   }

   //  Open report file.
   report.open("sensor2Report.txt");
   sensor2 >> time >> motion;

   //  Initialize objects using first data point.
        max = min = motion;

   //  Update summary data until trailer record read.
        do
        {
           sum += motion;
           if (motion > max)
           {
              max = motion;
           }
           if (motion < min)
           {
              min = motion;
           }
           num_data_pts++;
           sensor2 >> time >> motion;
        } while (time >= 0);
```

```
    //  Set format flags.
        report.setf(ios::fixed);
        report.setf(ios::showpoint);
        report.precision(2);

    //  Print summary information.
        report << "Number of sensor readings:   "
               << num_data_pts << endl
               << "Average reading:             "
               << sum/num_data_pts << endl
               << "Maximum reading:             "
               << max << endl
               << "Minimum reading:             "
               << min << endl;

    //  Close files and exit program.
        sensor2.close();
        report.close();
    return 0;
}  //end main
//*************************************************/
```

程序使用文件 sensor2.dat 打印出的报告与使用文件 sensor1.dat 所打印出来的报告内容是完全一致的。

5.2.3 文件结束

在每个数据文件的末尾都自动插入了一个与系统相关的文件结束指示符。函数 eof() 可以用来检测是否到达了指示符的位置。如果与数据文件相关联的文件流对象已经读取到了文件结束指示符，函数将返回 true。考虑下面的语句：

```
sum = count = 0;
cotal >> x;
while ( !data1.eof() )
{
   ++count;
   sum += x;
   data1 >> x;
}
avg   sum/count;
```

在第一条输入语句中，与数据文件相关联的输入文件流对象 data1 尝试从数据文件中读取一个值赋给 x。如果读取到了数据值，函数 eof() 将返回 0，循环结构中的语句将被执行。while 循环中的最后一条语句试图从数据文件中读取 x 的下一个值。如果还没有到达数据文件的末尾，while 循环将继续执行。当文件中没有数据可读取时，即读到了文件结束指示符时，在下一次调用函数 eof() 时将返回 1，这时被控制流将转向 while 循环之后的语句。这是数据终止循环的正确结构。在读取到文件结束指示符之前，函数 eof() 将不会返回 true。由于这个原因，在一条输入语句之后使用函数 eof() 测试文件的结尾是很重要的。

你也可以自动检测文件结束指示符，如下面语句所示：

```
while ( data1 >> x )
{
   ++count;
   sum += x;
}
```

这个 while 循环使用输入请求的返回值来控制执行过程。如果从文件 data1 中读取到了

一个数据将返回 true，若未能从文件流中读取到数据则返回 false。为了表述的清晰，在我们的例子中将使用函数 eof()。

现在我们假定数据文件 sensor3.dat 的内容与 sensor2.dat 相同，但是 sensor3.dat 不包含标志信号。在下面的程序中，我们将一直读取和累积信息，直到到达文件末尾为止：

```
/*                                                              */
/*  Program chapter5_6                                          */
/*                                                              */
/*  This program generates a summary report from                */
/*  a data file that does not have a header record              */
/*  or a trailer record.                                        */

#include <iostream> //Required for cin, cout, cerr
#include <fstream> //Required for ifstream, ofstream.
#include <string> //Required for string.
using namespace std;

int main()
{
   //  Declare and initialize objects.
   int num_data_pts(0), k;
   double time, motion, sum(0), max, min;
   string filename;
   ifstream sensor3;
   ofstream report;

   //  Prompt user for name of input file.
   cout << "Enter the name of the input file";
   cin >> filename;

   //  Open file and read the first data point.
   sensor3.open(filename.c_str());
  if(sensor3.fail())
  {
     cerr << "Error opening input file\n";
     exit(1);
  }

  //  open report file.
      report.open("sensor3Report.txt");

  //  While not at the end of the file,
  //  read and accumulate information
      sensor3 >> time >> motion;  // initial input
      while ( !sensor3.eof() )
      {
         num_data_pts++;
         if (num_data_pts == 1)
         {
            max = min = motion;
         }
         sum += motion;
         if (motion > max)
         {
            max = motion;
         }
         if (motion < min)
         {
            min = motion;
```

```
        }
        sensor3 >> time >> motion;  // input next
    }

//  Set format flags.
    report.setf(ios::fixed);
    report.setf(ios::showpoint);
    report.precision(2);
//  Print summary information.
    report << "Number of sensor readings: "
           << num_data_pts << endl
           << "Average reading:           "
           << sum/num_data_pts << endl
           << "Maximum reading:           "
           << max << endl
           << "Minimum reading:           "
           << min << endl;

//  Close file and exit program.

    sensor3.close();
    report.close();
  return 0;
} //end main
//-------------------------------------------------------------*/
```

使用文件 sensor3.dat 打印出的报告与使用文件 sensor1.dat 以及文件 sensor2.dat 打印出的报告内容是完全一致的。

在本节中，若数据文件存在且包含了所需的数据，那么程序将正常工作。如果这个程序将一直用于处理传感器数据，那么应当在其中包含用于处理所期望的文件结构的相关语句。此外，如果数据点的数目为 0，那么将会出现除 0（divide-by-zero）的问题。在打印报告之前，通过将数据点的数目与 0 比较可以避免除 0 问题的发生。

这三种文件结构在工程和科学应用中都经常用到。因此，了解你所处理的数据文件是哪种结构是非常重要的。如果做了错误的假设，那么可能会得到不正确的结果，而不是错误消息。有时候确定文件结构的唯一办法就是打印出文件的开头和末尾几行。

修改

在本章所编写的程序 chapter5_4 中，循环中包含了一个在循环第一次执行时的测试条件。当该条件为真时，max 和 min 的值将被初始化为第一组值。如果这些程序所使用的数据文件非常长，那么用于执行选择语句所需要的时间将会变得越来越长。一种避免该测试的方法就是在进入循环之前读取第一组数据并初始化对象。做出这种修改的同时还需要对程序做其他修改。

1. 修改程序 chapter5_4，将循环第一次执行时的测试条件去掉后，使得程序可以正常工作。
2. 使用一个空的数据文件来运行程序 chapter5_4。如果输入出现错误，修改程序以更正错误。

5.3 生成数据文件

生成一个数据文件与打印一个报告类似；不同的是后者将行输出到终端屏幕上，而我们要将输出写入数据文件中。但在生成数据文件之前，我们需要先确定使用哪种文件结构。在前面的讨论中，我们介绍了三种文件结构——①在文件开头给出记录数目；②在文件末尾给出标志信号用于指示数据结束；③在开头或末尾不包含任何特殊标记。

上面讨论的三种文件结构都有各自的优点和缺点。带有标志信号的文件简单易用，但是选择标志信号的时候要十分小心，不能包含出现在合法数据里的数据值。如果要在数据文件

的第一条记录里包含实际数据的行数。在生成数据文件之前我们必须知道在文件里有多少行数据。但在执行生成文件的程序之前，确定数据的行数并不容易。最容易生成的文件就是只包含有效信息的文件，不需要在文件开头或者末尾增加特别的信息。如果文件中的信息被用作绘图工具包使用，则最好使用第二种文件结构，即只包含有效信息。

我们现在给出第4章中一个程序的修改版本，原来的程序用于打印出包含气象气球的时刻、高度以及速率值的表格。修改后的程序除了打印出在屏幕上显示信息的表格外，还将这些数据（时刻、高度、速率信息）写入数据文件。可以将这个程序与程序 chapter4_8 比较：

```
/*-------------------------------------------------------*/
/*  Program chapter5_3                                   */
/*                                                       */
/*  This program generates a file of height and          */
/*  velocity values for a weather balloon. The           */
/*  information is also printed in a report              */
/*-------------------------------------------------------*/

#include <iostream> //Required for cin, cout.
#include <fstream> //Required for ofstream.
#include <iomanip> //Required for setw().
#include <cmath> //Required for pow().
#include <string> //Required for string.
using namespace std;

int main()
{
   //  Declare and initialize objects.
   double initial, increment, final, time, height,
          velocity, max_time(0), max_height(0);
   int loops, itime;
   string filename;
   ofstream balloon;

   //  Prompt user for name of output file.
   cout << "Enter the name of the output file";
   cin >> filename;

   //  Open output file
   balloon.open(filename.c_str());

   //  Get user input.
   cout << "Enter initial value for table (in hours) \n";
   cin >> initial;
   cout << "Enter increment between lines (in hours) \n";
   cin >> increment;
   cout << "Enter final value for table (in hours) \n";
   cin >> final;

   //  Set format flags for standard output.
   cout.setf(ios::fixed);
   cout.precision(2);

   //  Set format flags for file output.
   balloon.setf(ios::fixed);
   balloon.precision(2);

  //  Print report heading.
  cout << "\n\nWeather Balloon Information \n";
  cout << "Time    Height    Velocity \n";
  cout << "(hrs)   (meters)  (meters/s) \n";
```

```
    //  Determine number of iterations required.
    //  Use integer index to avoid rounding error.
    loops = (int)((final - initial)/increment);
    for (itime=0; itime<=loops; itime++)
    {
       time = initial + itime*increment;

       height = -0.12*pow(time,4) + 12*pow(time,3)
                - 380*time*time + 6100*time + 220;
       velocity = -0.48*pow(time,3) + 36*time*time
                - 760*time + 4100;

    //  Print report information to the screen.
       cout << setw(6) << time << setw(10) << height
            << setw(10) << velocity/3600 << endl;

    //  Write report information to a file.
       balloon << setw(6) << time << setw(10) << height
               << setw(10) << velocity/3600 << endl;

       if (height > max_height)
       {
          max_height = height;
          max_time = itime;
       }
    }

    //  Report maximum height and corresponding time.
    cout << "\nMaximum balloon height was "
         << setw(8) << max_height
         << " meters\nand it occurred at "
         << setw(6) << max_time << endl;

    //  Close file and exit program.
    balloon.close();
    return 0;
}
/*-----------------------------------------------------------*/
```

程序生成的数据文件中的前几行如下所示，其中的初始时刻为 0 小时，增加量为 0.5 小时，终止时间为 48 小时：

```
  0.00    220.00      1.14
  0.50   2176.49      1.04
  1.00   3951.88      0.94
...
```

这个文件的形式很易于被诸如 Matlab（附录 C 中将讨论它）的软件包用来绘制；第 4 章中的图 4.7 给出了使用该文件完成的一个图形。

修改

1. 修改程序 chapter5_5，让它生成一个文件，该文件的最后一行所包含的时间、高度和速率均为负值。
2. 修改程序 chapter5_5，让它生成一个文件，该文件的第一行指出了该数据文件中的有效行数。

5.4 解决应用问题：数据过滤器——修改 HTML 文件

被称作数据过滤器（data filter）的程序通常用于从一个数据文件中读取信息，修改文件的内容，然后将修改后的数据写到一个新文件中。假定我们在 Web 网站上找到一个含有重要信息的 HTML 文档，我们希望将其中的超文本标记语言（HyperText Markup Language，

HTML）命令移除，另存为一个纯文本文件。HTML命令（也称为标记）的一般形式如下：

```
<html command>
```

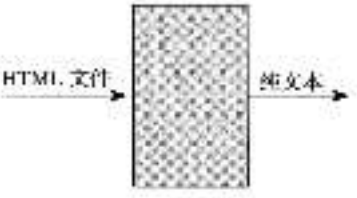
我们可以编写一个C++程序来读取HTML文件，过滤掉所有的标记，将过滤掉标记后的内容输出到一个新文件中。

1. 问题描述

写一个程序，去掉一个HTML文件中的所有标记，并将修改后的内容保存到一个新文件中。

2. 输入/输出描述

下面的草图表明程序的输入为HTML文件，输出是去除了标记的HTML文件的文本内容。



3. 用例

我们使用一个只有几行的简单HTML文件作为用例。文件的内容如下：

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT "text/html;
 charset=ISO-8859-1">
<TITLE>HomePage</TITLE>

<META NAME "GENERATOR" CONTENT="Internet Assistant for
 Microsoft Word 2.0z">
</HEAD>
<BODY BGCOLOR = "#118187" >
<H2>
<b><font size="6">
<p>
J. Togher
<br>
Accurate Solutions in Applied Physics, LLC
<br>
Albuquerque, New Mexico
</b></font>
</BODY>
</HTML>
```

去除标记后的文本如下：

```
HomePage
J. Togher
Accurate Solutions in Applied Physics, LLC
Albuquerque, New Mexico
```

4. 算法设计

算法设计的第一步是将问题解决方案分解为一组顺序执行的步骤：

分解提纲

1）从文件中读取一个字符；

2）确定字符是否是HTML标记的一部分；

3）打印出所有不是HTML标记的字符；

分解提纲中的第一步包含了一个用于从文件中读取每个字符的循环。循环退出的条件是测试是否到了文件末尾。我们的程序有两种不同的状态。在一种状态下，我们将从文件中读取文本（非标记），然后将我们所读取的每个字符输出。在另一种状态下，我们将读取一个标记，这些字符不会被打印。我们将使用一个布尔类型的对象来跟踪我们处于何种状态。如果我们处于文本状态，字符'<'将标志标记的开始。如果我们处于标记状态，字符'>'将标志标记的结束。细化后的伪代码如下所示：

```
Refinement in Pseudocode
main:   set text_state to true
        read character
        while not end-of-file
        if text_state is true
           if character = '<'
              set text_state to false
           else
              print character to file
        else
           if character = '>'
              set text_state to true
        read next character
```

伪代码中的步骤已经足够详细，可以转换成C++语句：

```
/*------------------------------------------------------------------*/
/* Program Chapter5_6                                               */
/* This program reads an html file, and writes the text             */
/* without the tags to a new file.                                  */

#include<iostream> //Required for cin, cout, cerr.
#include<fstream> //Required for ifstream, ofstream.
#include<string> //Required for string.
using namespace std;

int main()
{
  // Declare objects.
  char character;
  bool text_state=TRUE;
  string infile, outfile;
  ifstream html;
  ofstream htmltext;

  // Prompt user for name of input file.
  cout << "enter the name of the input file";
  cin >> infile;

  // Prompt user for name of output file.
  cout << "enter the name of the output file";
  cin >> outfile;

  // Open files.
```

```
    html.open(infile.c_str());
    if(!html.fail())
    {
       cout << "Error opening input file\n";
       exit(1);
    }
    htmltext.open(outfile.c_str());

    // Read first character from html file.
    html.get(character);

    while(!html.eof())
    {
    // Check state.
       if(text_state)
       {
            if(character == '<')          // Beginning of a tag.
                 text_state=FALSE;        // Change State.
            else
                 htmltext << character;   // Still text, write to
                                             the file.
       }
       else
       {
    // Command state, no output required.
            if(character == '>')          // End of tag.
                 text_state = TRUE;       // Change States.
       }

    // Read next character from html file.
       html.get(character);
    }
    html.close();
    htmltext.close();
    return 0;
}
```

5. 测试

使用习例中的 HTML 文件，我们得到了下面的程序输出：

```
HomePage
C. Ingber
Accurate Solutions in Applied Physics, LLC
Albuquerque, New Mexico
```

这与范例中的输出匹配。

修改

1. 修改程序 chapter5_6，将其中的 if 语句用 switch 语句替换。
2. 根据程序 chapter5_6 的伪代码，画出对应的流程图。

5.5 错误检查

在前一节中，我们使用函数 fail() 来测试输入流的状态，以判断在打开文件时是否出错。当我们从文件或者标准输入读取数据时也可能会出现错误。如果输入数据不是输入语句所希望的，那么输入流将被设置为错误状态（error state）。

下面是从标准输入读取两个值的语句：

```
int iVar1(0), iVar2(0);
cin >> iVar1 >> iVar2;
cout << iVar1 << " " << iVar2 << endl;
```

因为变量 iVar1 和 iVar2 被声明为 int 类型，因此从键盘输入的必须是由空白分隔的两个整数。在 C++ 中，空白被定义为空格、制表符、换行、换页以及回车。如果遇到任何其他类型的数据，如小数点、逗号，或者任何除空白外的非数字字符，cin 都会被置为错误状态。

假设从键盘输入的数据如下：

```
10,20
```

当在行尾点击回车键时，数据将存储在标准输入缓冲区中。在点击回车键后，标准输入缓冲区的内容和与之关联的指针如图 5.2 所示。

注意，由按下回车键所生成的换行符在输入缓冲区中作为一个单独字符存储。对于从输入缓冲区中读取数据的过程追踪如下所示。

语句跟踪：

语句	内存快照
`int iVar1(0), iVar2(0);`	integer iVar1 0　integer iVar2 0
`cin >> iVar1 >> iVar2;`	integer iVar1 0　integer iVar2 0
`cout << iVar1 << " " << iVar2 << endl;`	

标准输出

```
10 0
```

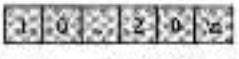


图 5.2　标准输入缓冲区

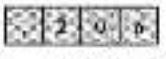


图 5.3　标准输入缓冲区

可以看到，字符 1 和 0 从输入缓冲区中读取出后被转换为整数值 10，并赋给 iVar1。因为逗号不能被解释为整型数据，因此逗号被留在了输入缓冲区中。因为整型值 10 被成功读出并赋给了 iVar1，因此在第一次释放后不会发生错误。当操作符 ">>" 试图为整型变量 iVar2 读取一个值时，会遇到逗号。由于逗号不是一个数字字符，也不是空白，所以逗号将留在输入缓冲区中。因此没有数值被读到，也没有数据被赋给 iVar2。

因为对 iVar2 的赋值失败，所以 iVar2 的值不会改变，cin 则处于一个错误状态。接下来 cout 语句执行时将会输出 10 0 到屏幕上。修改后的标准输入缓冲区如图 5.3 所示。

当输入流处于错误状态中时，程序将继续执行，但是应用到这个流上的操作都将是空操作，这意味着输入缓冲区不会被改变。

为了读取输入缓冲区中的值 20，我们必须首先将 cin 设为非错误状态。一旦 cin 处于正常状态，逗号就可以从输入流中读取出来，然后值 20 也可以被读到并赋给 iVar2。函数 clear() 可以用来将 cin 设为正常状态。在后面的章节中，我们将介绍包括函数 clear() 在内的一些用于处理流错误的函数，它们都在流类架构中被定义。

流的状态

每个流都有一个与之关联的状态（state）。流的状态在类 ios_base 中被定义为一组变量。这些变量通常被称作状态标志（state flag）。回忆一下前面我们说过所有的流类都继承自 ios_base。因此每个流都有一组与之相关的状态标志。状态标志指示着一个布尔状态，其值为 0 或 1。这些标志可以用来设置并测试潜在的错误。

流的状态受到事件的影响，当有操作应用到流上时，这些事件就会发生，如表 5.1 所示。注意，在状态标志的名字中包含了词 bit。前面说过一个 bit 就是一个二进制数位，可以表示值 0 或 1。因此，这里的名字指出了标志的布尔属性。

从表 5.1 可以看到，当流刚被创建时，goodbit 被初始化为真，只要 eofbit、failbit、badbit 保持为假，goodbit 将一直为真。当遇到文件末尾时，eofbit 被置为真。当打开文件失败或者遇到文件末尾或者在输入流中遇到非预期的数据时，failbit 被置为真。这些位的状态决定了相关流的状态。

表 5.1 流状态标志

事件	badbit	failbit	eofbit	goodbit
流初始化	0	0	0	1
文件打开失败	0	1	0	0
遇到非预期的数据	0	1	0	0
遇到文件结尾	0	1	1	0

如果流崩溃，数据丢失，那么 badbit 将被置为真。在某些用户自定义类型和类模板的输入过程中，当输入数据与要求的格式不匹配时，输入程序将会把 badbit 置为真。

使用流类架构中定义的函数，可以对流的状态进行测试和直接修改。这些函数被设计用来帮助处理错误，在所有依赖于正确输入而得到正确结果的程序中都应当使用它们。表 5.2 给出了流类层次中定义的部分函数列表。

表 5.2 流类函数

函数	描述
bool bad()	如果设置了 badbit，返回真
bool eof()	如果设置了 eofbit，返回真
bool fail()	如果设置了 failbit，返回真
bool good()	如果设置了 goodbit，返回真
void clear (iostate flag = goodbit)	设置状态标志
iostate rdstate()	返回状态标志的值

程序 chapter5_7 说明了这些处理输入错误的函数的用法。

```
/*------------------------------------------------------------*/
/* Program Chapter5_7                                         */
/* This program illustrates the use of stream                 */
/* flags and stream functions for detecting                   */
/* and handling input errors.                                 */

#include<iostream> //Required for cin, cout, cerr.
#include<fstream> //Required for ifstream, ofstream.
#include<string> //Required of string
#include<iomanip> //Required for setw()
using namespace std;

int main()
{
  //Declare variables.
  ifstream fin;
  ofstream fout;
  int iVar1, count = 0;
  string filename;
  char junk;
```

```
//Request name of input file.
cout << "Enter the name of input file";
cin >> filename;

//Open file and check for failure.
fin.open(filename.c_str());
while(fin.fail())
{
  --count;

//Open failed. Attempt to recover.
//rdstate() returns the iostate of a stream
//Print the state of the fin stream
cout << "could not open" << filename
     <<". The state of the fin stream is"
     << fin.rdstate() << endl;
cerr << setw(10) << "badbit:" << setw(10)
     << fin.bad() << endl;
cerr << setw(10) << "failbit:" << setw(10)
     << fin.fail() << endl;
cerr << setw(10) << "eofbit:" << setw(10)
     << fin.eof() << endl;
cerr << setw(10) << "goodbit:" << setw(10)
     << fin.good() << endl;
cerr << "**********************" << endl;

fin.clear(); //reset fin to good state

//Print the state of fin after clearing.
cout << "fin state reset to"
     << fin.rdstate() << endl;
cout << setw(10) << "badbit:" << setw(10)
     << fin.bad() << endl;
cout << setw(10) << "failbit:" << setw(10)
     << fin.fail() << endl;
cout << setw(10) << "eofbit:" << setw(10)
     << fin.eof() << endl;
cout << setw(10) << "goodbit:" << setw(10)
     << fin.good() << endl;
cout << "**********************" << endl;
if(count >= 2)
{
  cerr << "Failed to open an input file.";
  exit(1);
}
cout << "enter the name of a file";
cin >> filename;
fin.open(filename.c_str());
}
//File has been successfully opened.
//Get state of fin.
cout << "File" << filename
     << "is open. State of fin is"
     << fin.rdstate() << endl;

//Open file for output.
fout.open("output.dat");

//Print table of values to output file.
//Print heading.
fout << "Count iVar2" << endl;
fout << "--------------------" << endl;
```

```
//Read and print data from file
count = 0;//Reset count to zero
fin >> iVar;
while(!fin.eof())
{
  //Test state of fin.
  if(!fin) //If fin is bad
  {
    cerr << "bad data encountered.\n";
    cerr << "The state of fin is:"
         << fin.rdstate()<<endl;
    cerr << setw(10) << "badbit:"
         << setw(10) << fin.bad() << endl;
    cerr << setw(10) << "failbit:"
         << setw(10) << fin.fail() << endl;
    cerr << setw(10) << "eofbit:"
         << setw(10) << fin.eof() << endl;
    cerr << setw(10) << "goodbit:"
         << setw(10) << fin.good() << endl;
    cerr << "********************" << endl;
    //Reset fin to good state
    fin.clear();
    //Remove bad character from input stream.
    fin.get(junk);
    cerr << "The bad character is:" << junk << endl;
    continue; //Force next iteration of while loop.
  }
  ++count;
  cout << setw(5) << count
       << setw(10) << iVar << endl;
  fin >> iVar;
}
//Print current state of fin to standard output.
cout << "Outside of while, state of fin is:"
     << fin.rdstate() << endl;
cout << setw(10) << "badbit:" <<
     setw(10) << fin.bad() << endl;
cout << setw(10) << "failbit:"
     << setw(10) << fin.fail() << endl;
cout << setw(10) << "eofbit:"
     << setw(10) << fin.eof() << endl;
cout << setw(10) << "goodbit:"
     << setw(10) << fin.good() << endl;
cout << "********************" << endl;
fin.close();
fout.close();
return 0;
}
```

该程序的一次示例运行结果如下所示。我们首先输入一个不存在的文件的名字。当提示输入一个新的文件名时，我们输入文件名 test.dat。文件 test.dat 中包括下列数据：

```
20 30
5 6
1 3
8 5
 7 4
```

示例运行：

```
enter the name of input file badfilename
could not open badfilename. The state of the fin stream is 4
    badbit:         0
   failbit:         1
    eofbit:         0
   goodbit:         0
**********************
fin state reset to 0
    badbit:         0
   failbit:         0
    eofbit:         0
   goodbit:         1
**********************
enter the name of a file test.dat
File test.dat is open. State of fin is 0
Bad data encountered.
The state of fin is: 4
    badbit:         0
   failbit:         1
    eofbit:         0
   goodbit:         0
**********************
The bad character is:,
Outside of while. state of fin is: 6
    badbit:         0
   failbit:         1
    eofbit:         1
   goodbit:         0
**********************
```

在程序运行后，输出文件 output.dat 包含下面的数据：

```
Count       iVar2
---------------
    1          20
    2          30
    3           5
    4           6
    5           1
    6           4
    7           3
    8           8
    9           9
   10           7
   11           6
```

修改

1. 程序 chapter5_7 允许用户在第一次输入文件名失败后再尝试 5 次。修改程序，使得用户只有 1 次额外的尝试机会，需要使用一个布尔控制变量，而不是整型变量 count。
2. 修改程序 chapter5_7，使其能测试 fin 状态的代码段失效。编译并运行修改后的程序。标准输出上将显示什么内容？输出文件的内容是什么？解释运行结果。注意：让代码段失效，可以通过在代码段前后加上注释标记，使编译器将它们视作注释行。

练习

假设下面一行数据是从键盘输入的：

```
1,2,3
```

在下面每组语句执行后，给出相应的内存快照以及流状态标志的值。

```
1. int i(0), j(0);
   cin >> i >> j;
```

```
2. double x(0), y(0);
   cin >> x >> y;
```

```
3. char ch1, ch2;
   cin >> ch1 >> ch2;
```

```
4. char ch;
   double x, y;
   cin >> x >> ch >> y;
```

*5.6 数值方法：线性建模

线性建模是指这样的一个过程：确定一个线性等式，使之最适合描述一组数据点，而适合程度则是由数据点到直线的距离的平方所决定的，距离平方之和越小越适合。这一过程也称作线性回归（linear regression）。为了说明这一过程，我们以2.5节中从一种新引擎的气缸头部收集到的温度数据为例。

时间	温度
0	0
1	20
2	60
3	68
4	77
5	110

如果画出这些数据点，就会发现它们看上去都靠近一条直线。事实上，通过在图上画出这些点，然后计算出斜率和y截距，我们可以得出关于这条直线的较好估计。图5.4中标绘出了这些点（x轴为时间，y轴为温度），相应的直线表达式为

$$y=20x$$

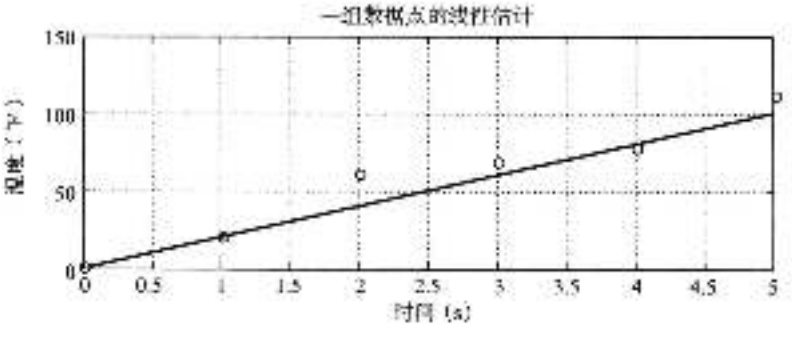


图5.4 使用线性估计对一组数据点建模

为了测量关于数据的线性估计的拟合程度，我们首先确定每个点到线性估计的垂直方向上的距离，图5.5中画出了这些距离。前两个点都落在了直线上，因此d_1和d_2的值都为0。d_3的值等于60−40即20。其余的距离也通过类似的方法计算出来。如果我们计算出距离的总和，由于一些为正，一些为负，这些值将会相互抵消，最后算出的总和比其实际应有的值要小。为了避免这个问题，我们应当将绝对值或者平方数相加；在线性回归中使用的是平方数。因此，衡量线性估计的适合程度就是计算点到线之间距离的平方的和。这个值容易计算得到，为573。

如果我们通过这些点画出另一条线，则可以计算出对应于这条新线的距离的平方和。比较这两条线，最适合的就是距离平方和较小的那一条，这一较小距离也称作最小二乘距离（least-square distance）。为了找出最小的距离平方和，我们从一个一般的线性等式开始：

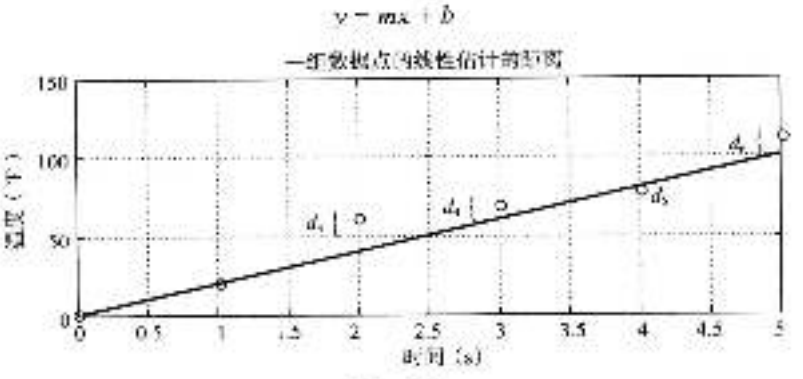


图 5.5

然后写出计算给定点到这个一般性等式之间的距离的算式。使用微积分的方法，我们可以计算出关于 m 和 b 的表达式，然后令表达式的值等于 0。通过这种方式我们可以确定出 m 和 b 的值，它们代表着具有最小的距离平方和的直线。在给出关于 m 和 b 的等式之前，我们要定义求和记号（summation notation）。

在本节开头给出的数据点可以用 (x_1, y_1)、(x_2, y_2)、…、(x_n, y_n) 来表示。符号∑表示求和，因此对于 x 坐标的求和可以写成下面的形式：

$$\sum_{k=1}^{6} x_k$$

这个求和表达式读作“x_k 的和，k 的取值为 1 ~ 6”。对于例子中的数据点，这个求和表达式的值为（0+1+2+3+4+5），即 15。使用例子中的数据点，其他的和计算如下：

$$\sum_{k=1}^{6} y_k = 0 - 20 + 60 + 68 - 77 + 110 = 335$$

$$\sum_{k=1}^{6} y_k^2 = 0^2 - 20^2 + 60^2 + 68^2 - 77^2 + 110^2 = 26\,653$$

$$\sum_{k=1}^{6} x_k y_k = 0 \times 0 - 1 \times 20 + 2 \times 60 + 3 \times 68 - 4 \times 77 + 5 \times 110 = 1202$$

现在我们回到寻找对于某组数据点最适合的直线表示的问题上来。使用前面的步骤，基于微积分的计算结果，我们可以计算出对于具有 n 个数据点的最佳线性拟合的斜率和截距的表达式。以最小二乘的形式表示如下：

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2} \tag{5.1}$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2} \tag{5.2}$$

对于示例的数据集，m 的最优值为 20.83，b 的最优值为 3.76。数据点和最佳拟合线性方程如图 5.6 所示。对于最佳拟合而言，距离的平方和为 356.82，而图 5.5 中的直线则为 573。

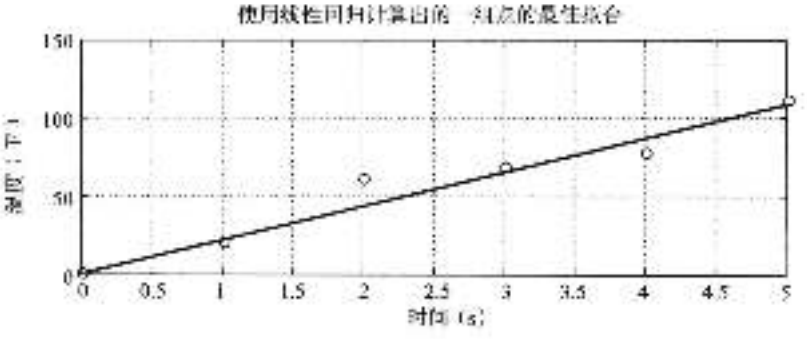


图 5.6 最小二乘线性回归模型

对于一组接近于线性的数据点进行线性回归的优势在于，我们可以对于某些不知道的数据进行估计或预测。例如，在气缸头温度的例子中，假如我们想估计 3.3 秒时的汽缸头温度。通过使用线性回归计算出的公式，我们可以计算出温度估计值为

$$\begin{aligned} y &= mx + b \\ &= (20.83)(3.3) + 3.76 \\ &= 72.5 \end{aligned}$$

使用这个公式模型，我们可以计算出使用线性插值无法计算出的估计值。例如，使用线性模型，我们可以计算出 8 秒时的温度估计值，但是使用线性插值我们无法计算出 8 秒时的温度估计值，因为我们没有一个时刻值大于 8 秒的数据点（这属于外推法，而不是线性插值）。

需要注意的是，线性模型并非对于所有的数据集都能给出一个好的拟合。因此，在使用它来预测一个新的数据点时，需要首先确定线性模型是否适合于给定的数据。第 6 章将讨论如何计算线性模型对一组数据的拟合程度的技术。

下一节我们将设计一个解决方案，用以确定与卫星收集的传感器数据的最佳拟合，然后我们将使用该模型估计其他传感器值。

*5.7 解决应用问题：臭氧测量

卫星传感器可以用来测量许多不同的信息。这些信息可以帮助我们更多地了解大气层。大气层是由围绕着地球的许多层组成的[12]。从地球的表面开始，我们所知道的层依次是对流层、平流层、散逸层、热电离层和外逸层，如图 5.7 所示。大气层中的每一层都可以通过其温度特性来表征。对流层处于大气的最内层，其高度在两极地地面约 5 千米，在赤道距地面约 18 千米，随着高度的增长，温度会有稳定的下降，大部分的云层都在这里形成。平流

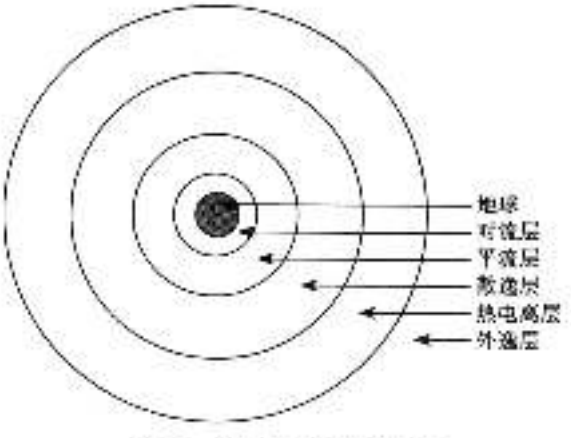


图 5.7 地球周围的大气层次

层的特征是在不同的高度上其温度相对一致。它从对流层开始向外扩展到距离地面约 50 千米（约合 31 英里）。飘入平流层的污染物在落入对流层被稀释之前，可以在平流层停留很多年。散逸层的范围从距地面 50 千米到距地面约 85 千米（约合 53 英里）之间。在这一层中，空气混合十分容易。在散逸层之上是热电离层，它在距地面 85 千米～140 千米（约合 87 英里）。在这一区域，热量来源于原子氧对太阳能的吸收。电离层是热电离层中带电粒子相对密集的地带。有些类型的通信就是通过电离层反射无线电波来完成的。最后，外逸层是大气层里最高的区域。在外逸层里，空气密度很低，以至于空气分子都会向外移动（就像是从大气层中逃离一样），而不是与其他分子碰撞。

1978 年在 NIMBUS 7 宇宙飞船上进行了一个卫星实验，用以收集有关中层大气的组成及结构的相关数据[13]。相关的设备和传感器在 1978 年 10 月 25 日至 1979 年 5 月 28 日之间收集数据，每天向地球返回超过 7000 组数据。这些数据用以确定在平流层和散逸层中温度、臭氧、水蒸气、硝酸和二氧化氮的分布情况。

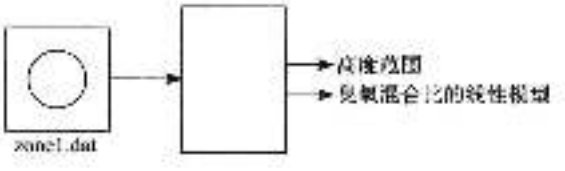
我们要考虑的问题是通过所收集的数据来测量在百万分之一体积比（parts per million volume）中臭氧的混合比。在小的区域中，这些数据是接近线性的，因此我们可以使用线性模型来估计在某些缺乏数据的高度上的臭氧比例。写一个程序从名为 zone1.dat 的数据文件中读取数据，该文件中包含了某区域内以千米为单位的高度值和对应的臭氧混合比（以 ppmv 为单位）。该数据文件中只包含合法数据，没有特殊的开头行和结尾行。使用前面小节中讨论的最小二乘方法来确定和打印模型。此外，打印出起始和终止高度，用以表明模型可以准确模拟的区域范围。

1. 问题描述

使用最小二乘法确定一个线性模型，用以估计给定高度的臭氧混合比。

2. 输入/输出描述

下面的 I/O 示意图给出了程序的输入和输出，输入是数据文件 zone1.dat，输出是高度范围和线性模型。



3. 用例

假设数据由下面四个数据点组成：

高度（km）	臭氧混合比（ppmv）
20	3
24	4
26	5
28	6

现在我们需要计算公式（5.1）和公式（5.2），为了方便起见，在这里我们再次将它们给出：

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2} \tag{5.1}$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2} \tag{5.2}$$

为了使用用例中的数据计算这些公式，我们需要先计算出下面一组和：

$$\sum_{k=1}^{4} x_k = 20 + 24 + 26 + 28 = 98$$

$$\sum_{k=1}^{4} y_k = 3 + 4 + 5 + 6 = 18$$

$$\sum_{k=1}^{4} x_k y_k = 20.3 + 24.4 + 26.5 + 28.6 = 454$$

$$\sum_{k=1}^{4} x_k^2 = (20)^2 + (24)^2 + (26)^2 + (28)^2 = 2436$$

使用这些和，我们可以计算出 m 和 b 的值：

$$m = 0.37$$
$$b = 4.6$$

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤：

分解提纲

1）读取数据文件中的值，计算出对应的和和范围；

2）计算出斜率和截距；

3）打印出高度范围和线性模型。

分解提纲的第一步包含了一个从数据文件读取数据的循环，同时还要计算出线性模型所需的对应值的和。在读取文件时我们还要确定数据点的数目。循环退出的条件是判断是否到达了文件末尾，因为在数据文件的头部和尾部都不存在特殊的信息。因为对于到达文件末尾的测试必须要在输入语句之后立即进行，所以我们将使用 while 循环而不是 do-while 循环。for 循环在这里也不合适，因为我们还不知道文件中包含多少数据点。因为我们要追踪高度的范围，所以还要保存第一个和最后一个高度值。分解提纲的步骤 2 和步骤 3 中是一组包括了计算和打印的顺序步骤。因此，细化后的伪代码如下：

```
Refinement to Pseudocode
Begin:   Set count to zero
         Set sumx, sumy, sumxy, sumx2 to zero
```

```
        while not at end-of-file
                read x, y
                increment count by 1
                if count = 1
                        set first to x
                add x to sumx
                add y to sumy
                add x2 to sumx2
                add xy to sumxy
        set last to x
        compute slope and y intercept
        print first, last, slope, y intercept
```

现在伪代码已经足够详细，可以转换成 C++ 语句：

```
/*------------------------------------------------------------------*/
/*  Program chapter1_B                                              */
/*                                                                  */
/*  This program computes a linear model for a set                  */
/*  of altitude and ozone mixing ratio values                       */
#include <iostream>  //Required for cin, cout
#include <fstream>  //Required for ifstream
#include <string>  //Required for string
using namespace std;

int main()
{
   //  Declare and initialize objects.
   int count(0);
   double x, y, first, last, sumx(0), sumy(0), sumx2(0),
          sumxy(0), denominator, m, b;
   string filename;
   ifstream zone1;
   cout << "Enter name of input file:";
   cin >> filename;

   //  Open input file.
   zone1.open(filename.c_str());
   if(zone1.fail())
   {
     cerr << "Error opening input file\n";
     exit(1);
   }

   //  While not at the end of the file,
   //  read and accumulate information.
   zone1 >> x >> y;
   while ( !zone1.eof() )
   {
     ++count;
     if (count == 1)
       first = x;
     sumx += x;
     sumy += y;
     sumx2 += x*x;
     sumxy += x*y;
     zone1 >> x >> y;
   }
   last = x;
```

```
    // Compute slope and y-intercept:
    denominator = sumx*sumx - count*sumx2;
    m = (sumx*sumy - count*sumxy)/denominator;
    b = (sumx*sumxy - sumx2*sumy)/denominator;

    // Set format flags:
    cout.setf(ios::fixed);
    cout.precision(2);

    // Print summary information.
    cout << "Range of altitudes in km: \n";
    cout << first << " to " << last << endl << endl;
    cout << "Linear model: \n";
    cout << "ozone-mix-ratio = " << m << " altitude + "
         << b << endl;

    // Close file and exit program.
    zone1.close();
    return 0;
}
// ------------------------------------------------------------------------ //
```

5. 测试

将用例中的数据作为数据文件 zone1.dat 的内容，我们将得到下面的程序输出：

```
Range of altitudes in km:
20.00 to 28.00

Linear model:
ozone-mix-ratio = 0.37 altitude + -4.60
```

这与用例中计算出的值相匹配。这里的高度范围也表明问题中用到的测量值是在平流层之中。

修改

这些问题与本节所开发的程序有关。在某些问题中你可能需要用到下面的关系式：

1km=0.621mi

1. 向程序中添加语句，使程序允许输入一个以千米为单位的高度值，然后使用模型估算出对应的臭氧混合比。
2. 修改问题 1 中的程序，使程序中检查你输入的高度是否位于模型能有效应用的范围之类。
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值。（程序中应当将英里转换成千米。）
4. 修改最初的程序，使程序还能打印出一个以英里为单位的线性模型。假设数据文件中包含的高度仍然是以千米为单位。

本章小结

本章我们介绍了从数据文件中读取程序中需要的信息的相关语句，还介绍了使用程序生成一个数据文件的相关语句。数据文件通常用于解决工程问题；因此在本书的前面我们就提到过在后续许多的问题解决方案中我们都将用到数据文件。我们还介绍了在输入数据时如何处理可能发生的错误。最后，我们介绍了为一组数据点生成线性模型的方法，使用最小二乘法确定最佳拟合解的公式。

关键术语

data file（数据文件）
data filter（数据过滤器）
end-of-file indicator（文件结束指示符）
error condition（错误条件）
input file stream（输入文件流）
least square（最小二乘）
linear modeling（线性建模）
linear regression（线性回归）
output file stream（输出文件流）
sentinel signal（标志信号）
standard input buffer（标准输入缓冲区）
stream class inheritances（流类继承）
stream flags（流标志）
stream state（流状态）
summation notation（求和记号）
trailer signal（跟踪信号）

C++ 语句总结

文件流对象声明

```
ifstream sensor1;
ofstream balloon;
```

文件打开函数

```
//filename is a string object
sensor1.open(filename.c_str());
balloon.open(filename.c_str());
```

文件失败函数

```
if (sensor1.fail())
```

文件输入

```
sensor1 >> x >> motion;
```

文件输出

```
balloon << setw(8) << time << setw(8) << height << setw(8) << velocity;
```

输出到标准错误

```
cerr << "Error encountered on input";
```

文件关闭函数

```
sensor1.close();
stream clear function
sensor1.clear()
standard error output stream
```

注意事项

1. 提示用户输入文件名，以使用户可以在运行时确定需要使用哪个文件。

调试要点

1. 当调试一个从数据文件里读数据的程序时，每当读取一个数据就立即将其打印出来，以检查在读取信息时是否出错。
2. 当调试一个从数据文件里读数据的程序时，确保你的程序检查了文件是否打开成功。
3. 为了避免在输入数据时出现无限循环，在每次循环之前都检查一下 istream 对象的错误状态。

4. 为了避免操作系统对大小写敏感的问题，文件名都使用小写字母。

习题

判断题

1. cin，cout 和 cerr 都是 C++ 关键字。
2. 当程序尝试打开一个文件作为输入时，如果文件没有找到，那么程序将终止。
3. 在 while 循环中的语句块至少会被执行一次。
4. for 循环中的语句块至少执行一次。
5. 当从标准输入中读取一个整数值时如果遇到了逗号，程序将终止。

多选题

6. 当遇到文件结尾时，下面（　　）流状态标志被置为真。

   (a) badbit　　(b) failbit

   (c) eofbit　　(d) goodbit

7. 当输入流处于错误状态时，程序将（　　）。

   (a) 终止

   (b) 继续正常执行

   (c) 继续执行，但是应用到流的输入操作都为空操作

   (d) 发送一个警告消息给 cerr，然后继续正常执行

内存快照问题

假设在键盘上输入了下面一行数据：

```
15.2 a, b
```

在下面每组语句执行完之后，给出对应的内存快照和流状态标志值。

8.
```
int i(0), j(0);
double x, y;
char ch1, ch2;
cin >> x >> y >> ch1 >> ch2;
```

9.
```
int i(0), j(0);
double x, y;
char ch1, ch2;
cin >> x >> i >> j >> ch1 >> ch2;
```

10.
```
int i(0), j(0);
double x, y;
char ch1, ch2;
cin >> i >> x >> ch1 >> ch2;
```

11.
```
int i(0), j(0);
double x, y;
char ch1, ch2;
cin >> i >> x >> ch1 >> ch2;
```

数据过滤器。调用数据过滤器的程序通常用于从数据文件中读取信息，并分析内容。在很多情况下，数据过滤器程序用于去掉数据文件中的数据错误，这些数据错误会导致其他读取数据文件的程序出现问题。下面一组习题用来完成对数据文件中的信息进行错误检查和数据分析。生成用来调试程序所有特殊的数据文件。

12. 写一个程序，从一个只包含整数值（因此文件中只应该包含数字、正负号和空白）的数据文件中读取信息。程序应当打印出文件中任何非法的字符。在结束的时候还要打印出所有非法字符的计数值。

13. 写一个程序分析一个只包含整数值和空白的数据文件。程序应该打印出文件的行数和整数值的个数（不是数字字符的个数）。

14. 写一个程序读取一个只包含有整数的文件，但其中的部分整数被逗号隔开了，如 145,020。程序应当将信息复制到一个新文件中，并将原有信息中的逗号都去掉。不要改变文件中每行的数值数目。
15. 写一个程序，读取一个包含有整数和浮点数的文件。数值间用逗号隔开，中间可以存在或不存在额外的空白。生成一个数据文件，其中的整数和浮点数均只由空格隔开；删除每个值之间的其他空白，只插入单个空格。不要改变文件中每行的数值数目。
16. 写一个程序，读取一个包含会计软件包生成的数据的文件。当文件只包含数值信息时，数值中可能包含逗号和美元符号，如 $3,200。负值包含在括号之中，如（200.56）。写一个程序生成一个新文件。文件中所包含的数值不能包含逗号和美元符号，负值要在开头加上负号表示而不是用括号标识。不要改变文件中每行上的数值数目。
17. 一个对两个文件进行逐字符比较以确定它们是否完全相同的程序是十分有用的。写一个程序比较两个文件。该程序应当打印出一条消息来说明两个文件是否完全相同或者是存在差异。如果文件不同，程序需要打印出那些不相同的行的行号。
18. 很有年龄人们对设计密码都很感兴趣。一个简单的编码方案就是将文本文件中的每个字符都用另一个字符替代，而替代的规则就是对所原字符偏移一个固定位置后用对应位置的字符替代。例如，如果每个字符都用在它所在位置右侧偏移两个位置的字符来替代，那么字母 a 将会被字母 c 替换，字母 b 被字母 d 替换，以此类推。写一个程序读取一个文件中的文本，生成一个包含按照前述方案编码过的文本文件。只改变字母顺序。
19. 写一个程序解码问题 18 中的方案。使用问题 18 中生成的文件对程序进行测试。

探空火箭轨迹。探空火箭用于探测不同层次的大气，以收集信息用于监控大气中的臭氧水平。除了载有收集上层大气数据的科学装置以外，火箭上还带有遥感系统，用来向发射站的接收机传输数据。除了科学数据外，火箭本身的性能测量数据也被传输回地面以供工程师作后续分析。这些性能数据包括高度、速率和加速度数据。假设这些信息存储在一个文件中，文件的每行都包含 4 个值：时间、高度、速率和加速度。假设单位分别是 s、m、m/s 和 m/s^2。

20. 假设文件 rocket1.dat 中包含一个指出后续实际数据行数的初始行。写一个程序读取这些数据，并确定火箭在哪个时刻开始落回地球。（提示：找出高度开始下降的那个时刻。）
21. 火箭的阶段数是由速率增长到某个峰值然后开始下降的次数决定的。写一个程序读取这些数据，并确定火箭的阶段数。使用数据文件 rocket2.dat，它包含一个尾行——其中所有的 4 个数值均为 −99。
22. 修改问题 21 中的程序，使它可以打印出每个阶段点火时刻对应的时间。假设点火时间对应速率开始增长的时刻。
23. 在火箭点火后的每个阶段，其加速度开始增加，然后降至 $-9.8m/s^2$，这是由于地球引力导致的下降加速度。找出火箭加速度只受重力影响的飞行时段，在这一时段加速度会在理论值的 65% 范围内浮动。使用数据文件 rocket3.dat，文件中不含特殊的首行和尾行。

缝合线包装（suture packaging）。缝合线是用于在受伤或手术后将活体组织缝合起来的线或纤维。缝合线的包装在送到医院之前必须被认真封好，以免污染物进入其中。用来封装包的物品称作封装模具（sealing die）。通常封装模具使用电加热器加热。为了让封装的过程成功，封装模具需要在特定的温度下制作，并且在特定的压力下在特定的时间内将包封装起来。封装模具封装包的时间称为停留时间（dwell time）。假设对于一个可接受的封装过程的可容忍的参数范围如下：

```
Temperature:        150-170 degrees F
Pressure:           60-70 psi
Dwell time:         2-3.5 s
```

24. 数据文件 suture.dat 中包含了一周内被拒绝的一批缝合线的信息。数据文件中的每行包含了一个被拒绝批次的批号，温度，压力和保压时间。质量控制工程师将分析这些信息，他们需要一份计算出因温度被拒绝、因压力被拒绝、因保压时间被拒绝的批次各自所占百分比的报告。对于一个给

定的批次，它可能同时由于多种原因被拒绝，因此它被计入所有实际存在的原因之中。写一个程序计算并打印出这三种百分比。

25. 修改问题24中的程序，使之可以打印出每种被拒绝分类中所包含的批次数，以及总的被拒绝批次数。(注意，一个被拒绝的批次在总数中只出现一次，但是在不同的拒绝分类中可以出现多次。)

26. 写一个程序来读取数据文件 suture.dat，并确保其中的信息只与应被拒绝的批次有关。如果有任何不应出现在数据文件中的批次，打印出与该批次信息有关的消息。

木材再生。在木材管理中有一个问题，就是如何确定一个地区需保留下来不能砍伐的木材量，以保证经过一段确定的时间后树木再生的木材量能够重新恢复。假设植被的再生每年以某个确定的比率进行，这取决于气候和土壤条件。再生公式表示了保留木材量与再生率之间的函数关系。例如，如果在收获之后留下100英亩未砍伐，再生率为0.05，那么在第一年年终将有 100 + 0.05 * 100 即 105 英亩植被。在第二年年终，将有 105 + 0.05 * 105 即 110.25 英亩植被。

27. 假设总共有14 000英亩，2500英亩未砍伐，再生率为0.02。打印出一个表格，显示每年年终的植被数量，总计20年。

28. 修改在问题27中开发的程序，以允许用户输入表格所要打印的总年数。

29. 修改在问题27中开发的程序，允许用户输入一个英亩值，程序应当计算出要再生达到该数值所需要的年数。

天气模式。在第1章中，我们讨论过国家气象局所收集的信息类型。图1.5中包含了一份可用的气象信息报告。在指导书CD包含的一组数据文件中，带有有关 Stapleton 国际机场在1991年1月到12月的气象信息相关内容。每个文件中都包含了一个月的数据；文件中的每行包含32个域的信息，其顺序如图1.5所示。这些数据都已被编辑过，以保证它们都是纯数值的。如果某个域的信息包含了T，那么在数据文件中对应的值将包含0.001。有几种可能的天气类型，而因为在一天之内可能出现多种天气，所以九个域都用于存储信息。例如，如果天气类型1发生，那么九个域中的第一个域将置为1，否则为0。如果天气类型2发生，那么九个域中的第二个域将置为1，否则为0。阵风的风向按照下列规则转换成整数表示：

```
N  1
NE  2
E  3
SE  4
S  5
SW  6
W  7
NW  8
```

数据文件中的每行数据里的数值都使用空白分隔，数据文件则依次命名为 jan91.dat、feb91.dat 等。

30. 写一个程序，确定1991年1月下面温度分类的天数：

(a) 低于0	(b) 0 ~ 32
(c) 33 ~ 50	(d) 51 ~ 60
(e) 61 ~ 70	(f) 超过70

注意，温度的变化在一天之内可能会跨越几个分类。

31. 修改问题30中的程序，使之打印出百分比而不是天数。

32. 修改问题30中的程序，使之对1991年5月到8月的数据进行分析。

33. 写一个程序，计算出1991年11月每天的平均温度。

34. 写一个程序，确定1991年12月温差最大的一天。打印出日期、最高温度和最低温度，以及温差。

关键路径分析。关键路径分析用于确定工程计划。这些信息在工程开始之前对于阶段的计划很重要，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划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。在下一个事件开始之前，前一个事件必须完成，而在同一事

件中的多个任务则可以同时进行。一个事件的完成时间取决于事件中完成时间最长的任务。类似地，工程完成的总时间是每个事件完成时间的总和。假设某个重点建筑工程的关键路径信息存储在一个数据文件中。数据文件的每行包含事件号、任务号以及完成任务需要的天数。数据文件中，事件2的任务数据排在事件1的任务数据之后，以此类推。因此，一个典型的数据集如下所示：

事件号	任务号	天数
1	11	9
1	27	5
1	34	4
2	15	9
2	18	4
3	26	7
4	15	2
5	26	7
6	27	7
9	16	4

35. 写一个程序，读取关键路径信息，打印出工程的完成时间表，表中列举出每个事件号、事件中任务的最大完成天数，以及工程完成的总天数。

36. 写一个程序，读取关键路径信息，打印出一份报告，其中列出事件号以及事件中完成时间超过5天的任务数。

37. 写一个程序，读取关键路径信息，打印出一份报告，其中列举出每个事件号和事件中的任务数量。

第 6 章

Engineering Problem Solving with C++, 3e

使用函数进行模块化编程

工程挑战：仿真

仿真已经逐渐被视作第三种科学范式，前两种分别是实验和理论。在某些情况下，由于实验实现难、成本或者开展实验所具有的危险性，实验可能无法进行，为了获取那些影响工程设计的关键参数，仿真是唯一可用的方法。在仿真中，物理现象的数学模型被转换成计算机软件。仿真过程可以使用不同的数据集和输入参数。从仿真运行过程中获取的信息可以减少风险，提高设计决策的速度和质量。

教学目标

在本章中，我们所讨论的问题解决方案中包括：

- 来自标准 C++ 库中的函数
- 自定义函数
- 生成随机数的函数
- 仿真技术
- 计算多项式实根的方法
- 数值积分方法

6.1 模块化

在前面的章节中，我们所设计的问题解决方案都只包含了一个名为 main() 的函数。当问题解决方案开始执行时，操作系统将从定义 main() 的语句块开始执行。因此，每个 C++ 程序中都必须包含且只能包含一个名为 main() 的函数。如果问题解决方案中没有包含 main() 函数或者含有一个以上的 main() 函数，那么将会出现编译错误，可执行文件将无法生成。

除了 main() 函数外，问题解决方案中还可以包含其他函数。这些函数，或者模块（module），是在 main() 之外定义的独立语句块，它们通常完成一个操作或者计算一个值。例如，在 cmath 中定义的函数 sqrt() 用作计算一个值的平方根，它可以被 main() 或者任何需要完成平方根计算的函数调用，这需要在程序中包含头文件 <cmath>。

为了在较长且更复杂的解决方案中保持简单和易读，我们在开发解决方案时将使用多个函数，而不再只写一个很长的 main() 函数。通过将一个解决方案划分为多个模块，可以使每个模块更简单且更易于理解。因此这与第 3 章和第 4 章所讨论的结构化编程的基本原则是一脉相承的。

正如我们在第 3 章中第一次给出分解提纲时所讨论的，开发一个问题解决方案的过程常常是一个分治的过程。分解提纲是一组解决问题的顺序的执行步骤。因此它为我们选择潜在的函数提供了一个好的开始。事实上，分解提纲中的每个步骤通常都对应着 main() 函数中所引用的一个或多个函数。

将一个问题解决方案分解成一组模块有很多优势。因为一个模块有一个特定的目标，它可以独立于问题解决方案的其他部分进行编写和测试。一个独立的模块比完整的解决方案要小，所以测试起来更容易。此外，一旦一个模块通过了认真的测试，它就可以用于新的问题解决方案之中，而不需要再次测试。例如，假如一个模块用于找出一组值的平均值。一旦这个模块编写好并经过测试，那么它就可以用在其他需要计算平均值的程序中。在大型软件系统的开发中，可重用性是非常重要的问题，因为它可以节省开发时间。事实上，经常被使用的模块库（如标准 C++ 库）通常在计算机系统中都是可用的。

模块的使用（称作模块化（modularity））通常会减少整个程序的长度，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，在需要的时候就可以使用一条单独的语句来引用。

如果将工程分为多个模块，由于各个模块可以独立于其他模块开发和测试，多个程序员就可以同时工作于同一个工程。这加速了开发进度，因为工程任务中的一部分内容现在可以并行进行。

使用那些为了特定任务而编写的模块也符合抽象（abstraction）的思想。模块中包含了这些任务的细节，程序员在使用这些模块时不需要关心其中的细节。我们在开发问题解决方案时所用到的 I/O 举图就是一个抽象的例子：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。类似地，我们可以将模块视作一个“黑盒”，它具有特定的输入，可以计算特定的信息，我们可以使用这些模块来帮助开发解决方案。因此，我们可以在更高的抽象层次上进行操作以解决问题。例如，标准 C++ 库包含了计算对数的函数。我们可以引用这些函数，而无须关注它们的实现细节，如函数是使用无穷级数的方法还是查表计算。通过抽象的思想，我们在提高软件质量的同时还可以减少软件开发时间。

作为总结，我们在下面列出了模块化的一些优势：

- 模块可以独立地编写和测试，因此在大型工程中的模块开发可以并行进行。
- 模块是解决方案的一小部分，因此测试起来更容易。
- 模块在编写和测试之后，可以在问题解决方案中使用多次。
- 一旦模块经过了认真的测试，在新的问题解决方案中使用时不需要重新测试。
- 使用模块通常可以缩短程序的长度，使程序更可读。
- 使用模块体现了抽象的思想，这允许程序员隐藏模块的细节；也允许我们使用模块时无须关心实现细节。

本章接下来的内容中我们还将指出模块的其他好处。

结构图（structure chart），或者叫模块图（module chart），展示了程序的模块结构。main() 函数调用其他函数，其他函数也可以互相调用。图 6.1 中包含了本章及后续两章中解决应用问题小节中一些程序的结构图。注意，在结构图中并未指出分解提纲中的步骤顺序。这些结构图说明了程序任务分解成模块的方式以及模块之间的调用关系。因此，分解提纲和结构图提供了对于问题解决方案有用而不同的视图。此外，注意结构图中并未包含模块对标准 C++ 库的调用关系，因为标准 C++ 库被使用得太频繁，且因为它们是 C++ 环境中不可或缺的一部分。

当我们开始开发更加复杂问题的解决方案时，程序将变得更长。因此，在这里我们给出三个关于长程序的调试建议。首先，有时使用不同的编译器来运行同一个程序是有帮助的，因为不同的编译器会给出不同的错误消息；事实上，某些编译器会给出扩展的错误消息，而

其他一些则只给出很少的错误信息。其次，在调试长程序时的另一个有用步骤是在某些代码段周围添加注释（/* 和 */），这可以让你将注意力集中在程序的其他部分。当然，你要注意你没有将可能影响你希望测试的那部分语句注释掉。最后，对于复杂的函数应由开发者自己测试。这通常由称作驱动（driver）的专门程序来完成。驱动的目的是在你和你所测试的函数之间提供一个简单的接口。有代表性的就是程序的 main() 函数要求你输入传递给函数的参数，它则会打印出函数返回值。在接下来的几小节中，驱动程序的益处将会变得更加明显。

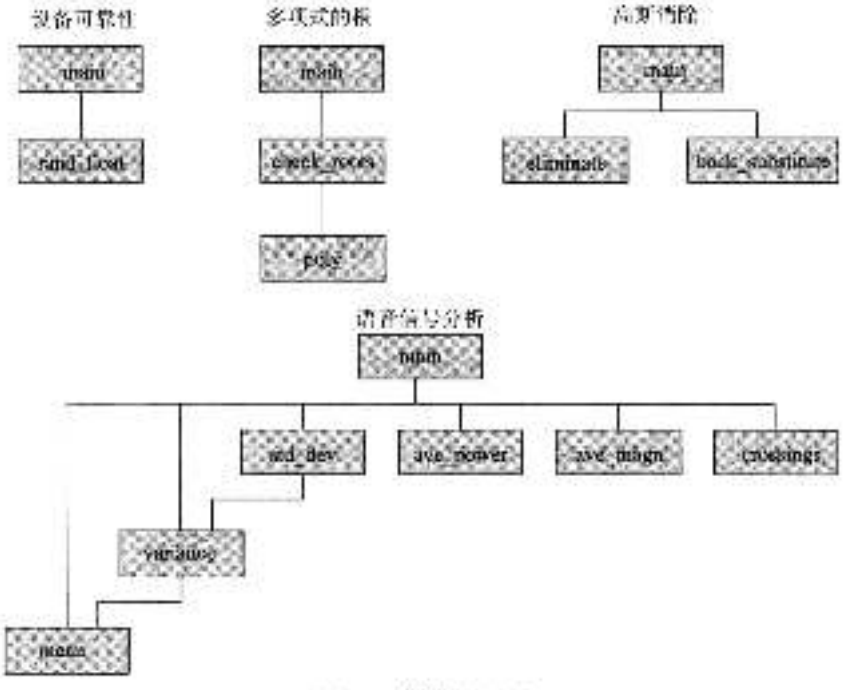


图 6.1 结构图示例

6.2 自定义函数

程序的执行总是从 main 函数开始。在程序遇到其他函数名时，其他函数将被调用。这些函数必须在包含 main() 函数的文件中定义，或者在其他可用的文件或者库文件中定义。（如果函数包含于一个系统库文件中，如 sqrt 函数，通常被称作库函数（library function），其他函数通常被称作自定义函数（programmer-written function 或 programmer-defined function）。）在执行完一个函数中的语句之后，程序将继续执行调用该函数之后的语句。

考虑一个将摄氏度转换成华氏度的简单任务。我们可以写一个自定义函数来完成计算，然后从任何需要这种转换的程序解决方案中调用这个函数。我们的转换函数需要一个以摄氏度为单位的温度作为输入参数，它将返回一个等价的华氏度温度值。为了说明这一过程，我们将编写一个程序解决方案从输入文件中读取摄氏温度值，并将华氏温度值写到一个输出文件中。程序解决方案将包含两个自定义函数，函数 main() 完成输入和输出，函数 celsiusToFahr() 完成转换过程。图 6.2 中给出了转换程序的结构图和流程图，我们的解决方案和程序追踪在图 6.2 后给出。

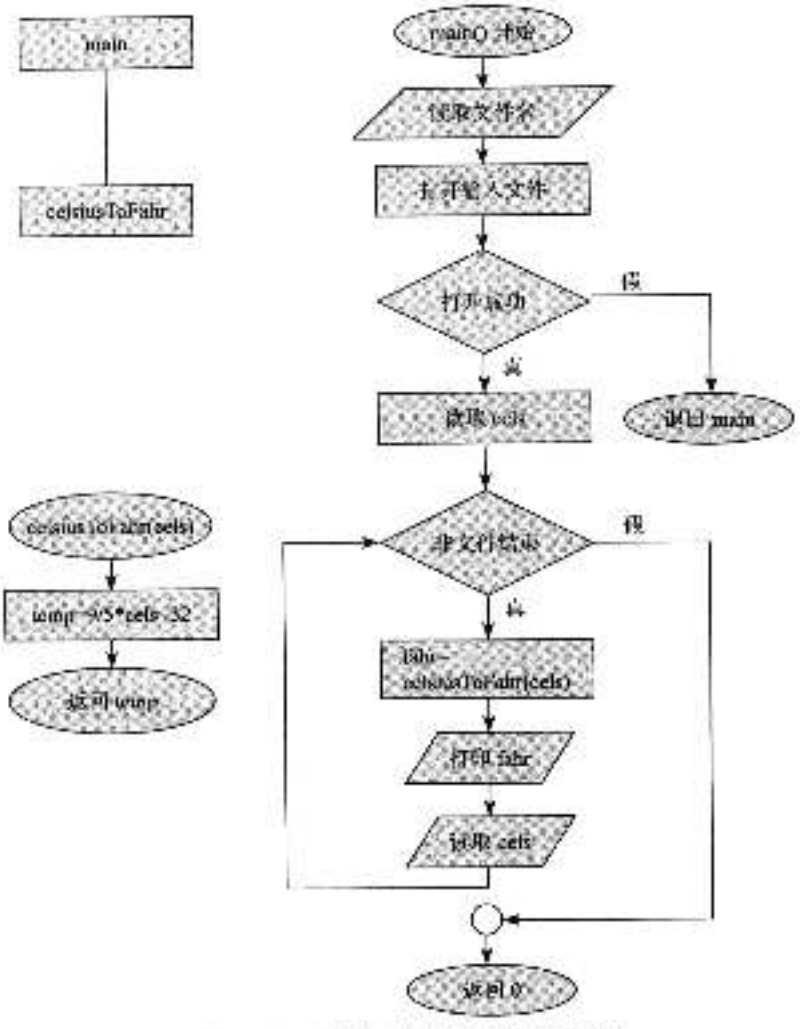
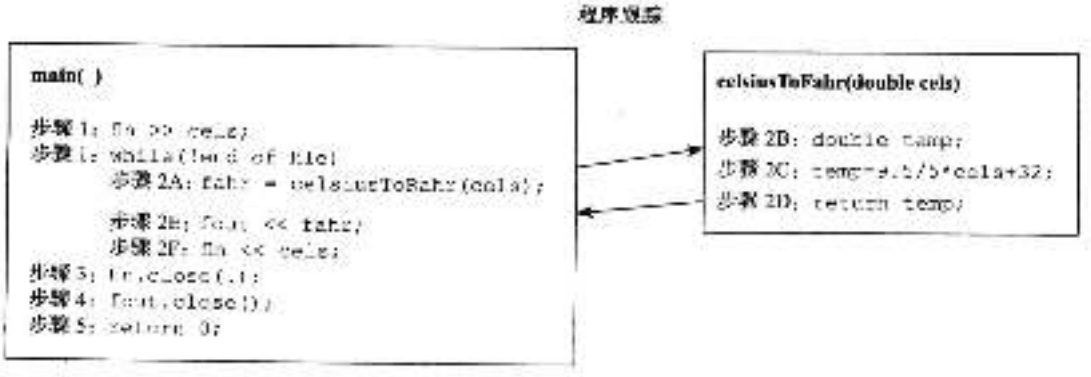


图 6.2 温度转换程序的结构图和流程图

```
#include<iostream> //Required for cin, cout
#include<fstream> //Required for ifstream, ofstream
#include<string> //Required for string, c_str()
using namespace std;

//Function prototype.
double celsiusToFahr(double celsius); //Programmer defined function.

/* ------------------------------------------------------------------*/
/* Program chapter6_1                                                */
/* This program reads temperatures in degrees Celsius                */
/* from an input file, calls a conversion function                   */
/* and writes converted temperatures to an output file.              */
int main()
{
  //Declare variables.
  ifstream fin;
  ofstream fout;
  string filename;
  double cels, fahr;
```

```
    //Open files.
    cout << "Enter name of input file\n ";
    cin >> filename;
    fin.open(filename.c_str());
    if(fin.fail())
    {
      cerr << "Could not open the file " << filename << endl;
      exit(1);
    }
    fout.open( (filename + "ToFahr").c_str() );
    fin >> cels;

    //while not end of file
    while(!fin.eof())
    {
      //Convert temperature and write to file.
      fahr = celsiusToFahr(cels); //Function call.
      fout << fahr << endl;
      fin >> cels;
    }
    fin.close();
    fout.close();
    return 0;
  }
  /*                                                                    */
  /* This function performs a conversion from                           */
  /* degrees Celsius to degrees Fahrenheit.                             */
  /* Precondition: celsius holds a temperature in degrees Celsius       */
  /* Postcondition: returns degrees Fahrenheit                          */

double celsiusToFahr(double celsius) //Function header.
{
  //Declare local variables.
  double temp;

  //Convert from degrees celsius to degrees Fahrenheit.
  temp = (9.5/5.0)*celsius + 32.0;

  return temp;
}
```

注意，函数 celsiusToFahr() 中没有包含输入语句和输出语句，只有用于计算温度转换和返回计算值的语句。输入和输出在 main() 中完成。现在我们给出程序 chapter6_1 中从“fin >> cels;”语句开始的部分程序跟踪内容。



程序跟踪

main( )

步骤 1: fin >> cels;
步骤 1: while(!end of file)
　　步骤 2A: fahr = celsiusToFahr(cels);
　　步骤 2B: fout << fahr;
　　步骤 2F: fin << cels;
步骤 3: fin.close();
步骤 4: fout.close();
步骤 5: return 0;

celsiusToFahr(double cels)

步骤 2B: double temp;
步骤 2C: temp=9.5/5*cels+32;
步骤 2D: return temp;

6.2.1 函数定义

当我们认真观察到现在为止所定义的函数时，可以发现一个函数定义是由一个函数头（function header）以及其后的语句块组成的。函数头定义了函数的返回值类型；在前一小节的示例中，对于 main() 是 int，对于 celsiusToFahr() 是 double。如果函数没有返回值，那么返回类型为 void。在返回类型之后是函数名和参数列表（parameter list）。注意，参数列表可以为空，如 main() 函数的定义。函数定义的一般形式如下：

```
return_type function_name(parameter list)
{
   declarations;
   statements;
}
```

参数列表代表着传递给函数的信息；如果没有输入参数，则可以忽略参数列表。但是，在函数名之后必须跟有一对圆括号，就像我们定义 int main() 一样。函数使用的其他对象在函数的语句块中定义，后者称作函数体（function body）。

函数名的选择应有助于说明函数的目的，还应在函数中包含进一步说明函数功能和步骤的注释。我们还使用带有破折号的注释行将自定义函数与 main() 函数和其他自定义函数隔开。

所有具有返回值的函数中都必须包含一条 return 语句。return 语句的一般形式如下：

```
return expression;
```

该语句指明了要返回给调用自定义函数的语句的值。return 语句中返回的表达式类型应该与函数定义中指明的返回类型匹配，以避免发生潜在的错误。如果需要，可以使用类型转换操作符（在第 2 章中讨论）显式指明返回表达式的类型。void 函数不需要返回值，因此其函数头一般如下：

```
void function_name(parameter declarations)
```

在一个 void 函数中 return 语句是可选的，它也不需要包含一个表达式。一般形式为：

```
return;
```

函数定义：函数定义由函数头和其后的语句块组成。

语法

```
返回类型 函数名([参数列表]) // 函数头
{
// 语句块
}
```

示例

```
int main()
{
  cout << "hello world" << endl;
  return 0;
}
void drawBlock(ostream& out, int size)
{
  int width, height;
```

```
    for(height=0; height<size; ++height)
    {
      for(width=0; width<size; ++width)
      {
        out << "-";
      }
      out << endl;
    }
  }
```

函数可以在 main 函数之前或之后定义。记住，一个右大括号标志着定义函数体的语句块的结束。但是，在另一个函数定义开始之前，另一个函数必须完成定义；函数定义不能嵌套。在我们的程序中，首先包含了 main 函数，其他函数按照它们在程序中被调用的顺序依次包含。

现在我们给出使用定义函数的第二个程序示例。图 6.3 中标绘出的 sinc(x) 函数在许多工程应用中都被用到。函数 sinc(x) 的最常用定义如下：

$$f(x) = \mathrm{sinc}(x) = \frac{\sin(x)}{x}$$

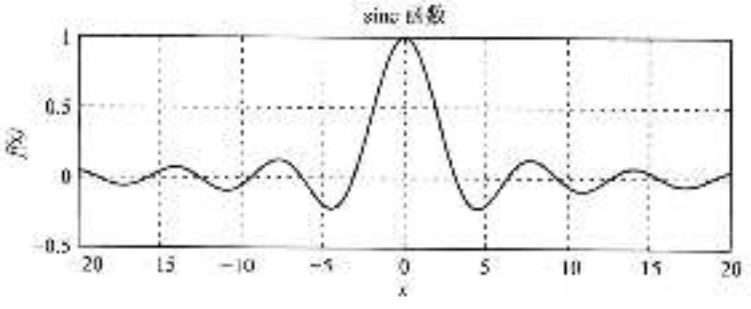


图 6.3 [-20, 20] 上的 sinc 函数

这个函数的值很容易计算，但 sinc(0) 除外，因为它是一个 0/0 的不确定形式。在这种情况下，微积分中的洛必达法则可以证明 sinc(0) = 1。

假如我们希望开发一个程序，允许用户输入一个 $a \sim b$ 的区间限制。程序应该计算并打印出区间 $a \sim b$ 上的 21 个 sinc(x) 值，这里的 x 取值从 $a \sim b$（含 a 和 b）均匀增加。因此，x 的第一个值为 a。下一个 x 的值应当加上一个增量，以此类推，直到第 21 个值，即 b。因此，x 的增量为

$$x\_increment = \frac{interval\_width}{20} = \frac{b-a}{20}$$

选择 a 和 b 的值以及 x 的增量，确保 a 为第 1 个值，b 为第 21 个值。

因为 sinc(x) 不是标准 C++ 库中提供的数学函数的一部分，所以我们采用两种方法来完成解决方案。在一种方案中，我们将 sinc(x) 的计算语句放在 main() 函数中；在另一种方案中，我们使用一个自定义函数来计算 sinc(x)，然后在每次需要计算时调用该自定义函数。下面给出了这两种方案，以便于我们进行比较。

解决方案 1：

```
/*-------------------------------------------------------*/
/*  Program chapter6_2                                   */
```

```
/*                                                          */
/*  This program prints 21 values of the sinc               */
/*  function in the interval [a,b] using                    */
/*  computations within the main function.                  */

#include<iostream> //Required for cin, cout.
#include<cmath> //Required for sin()
using namespace std;

int main()
{
   //  Declare objects.
   double a, b, x_incr, new_x, sinc_x;


   //  Get interval endpoints from the user.
   cout << "Enter end points a and b (a<b): \n";
   cin >> a >> b;
   x_incr = (b - a)/20;


   //  Set formats.
   cout.setf(ios::fixed);
   cout.precision(6);

   //  Compute and print table of sinc(x) values.
   cout << "x and sinc(x) \n";
   for (int k=0; k<=20; ++k)
   {
      new_x = a + k*x_incr;
      if (fabs(new_x) < 0.000)
      {
         sinc_x = 1.0;
      }

      else
      {
         sinc_x = sin(new_x)/new_x;
      }
      cout << new_x << " " << sinc_x << endl;
   }


   //  Exit program.
   return 0;
}
/*-------------------------------------------------------------*/
```

解决方案 2：

```
/*-------------------------------------------------------------*/
/*  Program chapter6_3                                         */
/*                                                             */
/*  This program prints 21 values of the sinc                  */
/*  function in the interval [a,b] using a                     */
/*  programmer defined function.                               */
/*                                                             */

#include<iostream> //Required for cin, cout
#include<cmath> //Required for sin()
```

```
using namespace std;

//Function Prototype
//Programmer defined function.
double sinc(double x);

int main()
{
   //  Declare objects
   double a, b, x, incr, new_x;

   //  Get interval endpoints from the user.
   cout << "Enter endpoints a and b (a<b): \n";
   cin >> a >> b;
   x_incr = (b - a)/20;

   //  Set formats
   cout.setf(ios::fixed);
   cout.precision(6);

   //  Compute and print table of sinc(x) values.
   cout << "x and sinc(x) \n";
   for (int k=0; k<=20; k++)
   {
      new_x = a + k*x_incr;
      cout << new_x << " " << sinc(new_x) << endl;
   }

   //  Exit program.
   return 0;
}
/*-------------------------------------------------------------*/
/*  This function evaluates the sinc function.                 */

double sinc(double x)
{
   if (fabs(x) < 0.0001)
   {
      return 1.0;
   }
   else
   {
      return sin(x)/x;
   }
}
/*-------------------------------------------------------------*/
```

下面的输出给出了每个程序中可能发生的示例交互过程：

```
Enter endpoints a and b (a<b):
-5 5
x and sinc(x)
-5.000000  0.191785
-4.500000 -0.217229
-4.000000 -0.189201
-3.500000  0.100224
-3.000000 -0.047040
-2.500000 0.239989
```

```
-2.000000 0.454649
-1.500000 0.664997
-1.000000 0.841471
-0.500000 0.958851
0.000000 1.000000
0.500000 0.958851
1.000000 0.841471
1.500000 0.664997
2.000000 0.454649
2.500000 0.239389
3.000000 0.047040
3.500000 -0.100229
4.000000 -0.189201
4.500000 -0.217229
5.000000 -0.191785
```

图 6.4 中包含了 4 组不同 $[a, b]$ 区间对应的 21 个值的图示。由于程序只计算 21 个值，所以图示的分辨率会受到间隔大小的影响：增量的间隔越小，分辨率越好。方案 2 的 main 函数比方案 1 中的 main 函数更易读，因为前者比后者更短。

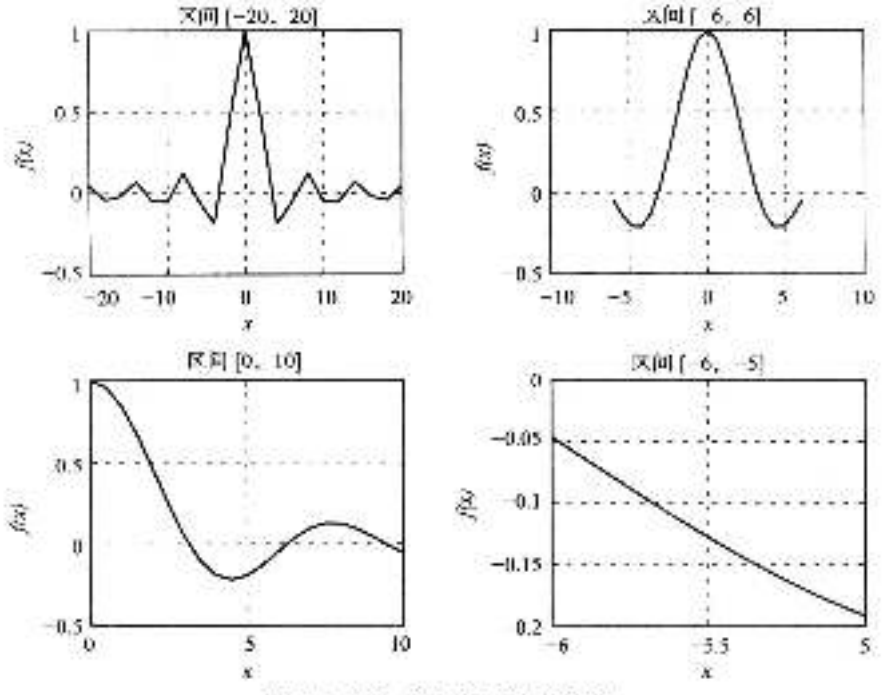


图 6.4 4 种不同区间的程序输出

在方案 2 中的 main 函数和程序 chapter6_1 中的 main 函数之前都各自包含了一个函数的声明语句。在下面的小节中我们将讨论函数原型，并对函数调用语句和函数定义之间的关系进行更深入的说明。

6.2.2 函数原型

程序 chapter6_3 的 main 函数定义之前包含了下面的语句：

```
double sinc(double x);
```

这种语句称为函数原型（function prototype）。这个原型声明 sinc() 要求一个 double 类型的参数，并返回一个 double 类型的值。编译器将检查所有关于 sinc() 的调用是否与函数原型兼容。注意，编译器并不需要使用标识符 x 来检查对 sinc() 的调用。事实上，下面的函数原型语句对于程序 chapter6_3 而言也是合法的。

```
double sinc(double);
```

这两种函数原型为编译器提供的信息都是相同的。我们较为常用带有标识符的函数原型，因为标识符可以帮助说明参数的定义以及顺序。

函数原型语句：函数原型为编译器提供了检测函数被调用时存在的潜在错误信息。

语法

```
返回类型 函数名（参数列表）;
```

示例

原型：

```
double sinc(double x);
```

合法引用：

```
double y(.5);
cout << sinc(y);
cout << sinc(1.5);
cout << sinc(10);
y = sinc(0.0);
```

非法引用：	错误信息：
`sinc("1.5");`	Invalid string argument,expeced double
`sinc('0');`	Invalid char argument,expected double

原型：

```
void clearScreen(ostream&);
```

合法引用：

```
clearScreen(cout);
```

非法引用：	错误信息：
`cout << clearScreen(cout);`	std::ooperator << error,clearScreen() is a void function

为了让函数原型在定义它的文件中对所有函数可用，函数原型应当被包含在程序文件的开始或位于任何语句块之外。换言之，在程序文件的开始处包含函数原型或使之位于所有语句块之外，将使函数原型具有文件作用域（file scope）或全局作用域（global scope）。在后面的章节中我们将对作用域进行更详细的讨论。像 cmath 这样的头文件中含有库文件中包含的函数的函数原型；否则，我们将需要在自己的程序中包含诸如 log() 和 sqrt() 等函数的函数原型。

当程序调用了大量的自定义函数，如果要包含所有的函数原型语句会显得很臃肿。在这种情况下，可以将函数原型和相关的符号常量定义在一个自定义头文件（custom header file）中。头文件的文件名应该以 .h 为后缀。通过使用 include 语句就可以引用自定义头文件，其中文件名使用双引号限定。在这一章中，我们将开发一组用于计算一组值的统计数据的两

数。如果对应的函数原型都包含在名为 stat_lib.h 的头文件中，那么使用下面的语句就可以将所有的函数原型包含在程序中了：

```
#include "stat_lib.h"
```

自定义头文件通常都与随那些供程序员共享的实例一起出现。注意，因为自定义头文件是由程序员定义的，所以它们的文件名是用引号界定的，而不是“<>”字符。如果自定义头文件与可执行文件不在同一个目录下，还必须在文件名前加上完整的路径。

6.3 参数传递

函数头定义了调用函数时所需要的参数，这些参数称为形参（formal parameter）。任何调用函数的语句都必须为对应的参数提供相应的值，这些提供的值称作实参（actual parameter）或函数参数（function argument）。例如，在本节前面开发的 sinc 函数中，其函数头为：

```
double sinc (double x);
```

假如 main 函数中使用下面的语句对该函数进行调用：

```
cout << newX << " " << sinc(newX) << endl;
```

那么这里的形参是 x，函数参数是 newX。当 sinc 函数在 cout 语句中被调用时，函数参数的值被赋给形参 x，然后函数头之后的语句块被执行，sinc() 的返回值被输出到标准输出缓冲区中，然后和 newX 的值一起被打印在屏幕上。

需要指出的是，形参的值并没有被返回给函数参数。我们使用内存快照来展示从函数参数到形参的值的变化过程。假设在 sinc 函数被调用时 newX 的值为 5.0：

main()	sinc(double x)
函数参数	形参
double newX 5.0	double x 5.0

在函数参数的值被赋值到形参中之后，sinc 函数被执行。当调试一个函数时，一个比较好的做法就是使用 cout 语句将函数调用时的函数参数和在函数开始时的形参的内存快照打印出来。形参只定义在定义函数体的语句块中，在语句块之外不能被引用。因此形参是局部（local）变量。

6.3.1 值传递

在 C++ 程序设计中，参数传递的方式采用的是如 sinc() 函数中一样的值传递（pass by value）。当发生函数调用时，参数值被传递给函数并赋给对应的形参。一般来说，C++ 函数不会改变函数参数的值，但当函数参数是数组（在第 8 章讨论）或形参被声明为引用传递（pass by reference）时会有例外。后一种情况将在下一节讨论。

因为函数 sinc 中的形参采用的是值传递，所以传递的有效参数可以是表达式或者其他的函数调用，如下面的语句所示：

```
cout << sinc(x-2.5) << endl;  //argument is expression x-2.5

double y;
cin >> y;
cout << sin(y) << endl;
```

```
x = x*x + sinc(2.0*x); //argument is expression 2.0*x
x = sinc(fabs(y));  //argument is value returned by fabs(y).
```

如果函数有形参，那么形参和函数参数在数目、类型和顺序上都必须匹配。如果有不匹配的情况，编译器将会通过函数原型检查出来。如果函数参数的类型与对应的形参不相同，那么函数参数的值可能被转换成对应的类型，这种转换称作参数的强制转换（coercion of argument），且可能会导致错误。强制转换在第2章中进行过讨论，它是将一种类型的值存储到一个不同类型的对象中。将值转换成更高的类型（如从 float 到 double）一般都可以正确工作；而将值转换成更低的类型（如从 double 到 int）常常会导致错误。为了说明参数的强制转换，考虑下面返回两个值中最大值的函数：

```
/*-------------------------------------------------------*/
/*  This function returns the maximum of two             */
/*  integer values.                                      */

int theMax(int a, int b)
{
   if (a > b)
   {
      return a;
   }
   else
   {
      return b;
   }
}
/*-------------------------------------------------------*/
```

假设一个对该函数的调用为 theMax(xSum, ySum)，其中 xSum 和 ySum 都是整数，且值分别为 3 和 8。下面的内存快照展示了在调用 theMax(xSum, ySum) 时函数参数到形参的转换过程：

函数参数		形参	
int xSum	3	int a 3	a 接收 xSum 的值
int ySum	8	int b 8	b 接收 ySum 的值

上述函数调用的返回值为 8。

现在假设对函数 theMax() 的调用使用的是 double 类型的对象 t1 和 t2。如果 t1 和 t2 的值分别为 2.8 和 4.6，那么当调用 theMax(t1, t2) 时将会发生下面的参数转换：

函数参数		形参	
double t1	2.8	int a 2	a 接收值 2
double t2	4.6	int b 4	b 接收值 4

函数调用 theMax(t1, t2) 的返回值为 4。很显然，函数返回的值是不正确的。但是，问题不在函数里，而是由于调用时函数参数类型不正确。

如果函数参数的顺序不正确，也会导致错误。编译器可能检测不到这些错误，也很难通过代码的检查来发现它们；因此，在形参的顺序和函数参数匹配时一定要十分小心。

练习

考虑下面的函数：

```
/*-------------------------------------------------------*/
/*  This function counts positive parameters.            */

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

假如函数被下面的语句调用：

```
x = 2;
total = positive(x, sqrt(x), x-30);
```

1. 给出形参和函数参数的内存快照。
2. total 的新值是多少？

6.3.2 引用传递

当形参是一个值传递的参数时，对应的函数参数在函数体中不会被修改。当函数的目的是修改一个或多个函数参数时，对应的形参必须接收要修改的参数的地址（address），而不是参数的值。这种参数传递方式称作引用传递（pass by reference）。

为了说明什么时候需要使用参数的引用传递方式，我们开发了一个交换两个整型变量值的函数。交换两个值需要三条赋值语句和一个临时的整型变量。执行交换的语句块和相应的内存快照如下：

```
{
   int hold, a(5), b(10);
   hold = a;
   a = b;
   b = hold;
}
```

	整型值 a	整型值 b	整型值 hold
	5	10	?
	5	10	5
	10	10	5
	10	5	5

在类似排序程序这样需要频繁交换值的问题解决方案中，编写一个函数来完成这项工作会使程序变得简单。考虑下面的函数，该函数试图通过值传递的方式来交换两个整型参数的值。

```
/*-------------------------------------------------------------*/
/* Incorrect function that attempts to swap two integer value  */
/* using pass by-value parameters.                             */
```

```
void swapIntegers(int a, int b)
{
   //Define temporary variable:
   int hold=a;

   //Swap the values in a and b
   a = b;
   b = hold;

   //exit void function
   return;
}
```

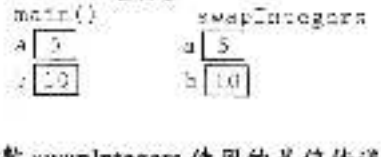
我们将使用下面的驱动程序来测试这个函数：

```
/*************************************************************/
/* Driver program to test swapIntegers(int a, int b)         */

#include<iostream>//Required for cin, cout.
using namespace std;

//Function prototype
void swapIntegers(int, int);

int main()
{
   //Declare variables.
   int a, b;

   //Get values for a and b.
   cout << "Enter two integer values: ";
   cin >> a >> b;

  //Print values of a and b before the swap
  cout << "Before swap:\n a is " << a << " b is " << b << endl;

  //Call swap function and print values
  swapIntegers(a, b);
  cout << "After call to swapIntegers\n a is " << a << " b is "
       << b << endl;

  //Exit main
  return 0;
}
```

驱动程序的一次示例运行的输出结果如下：

```
Enter two integer values: 5 10
 Before swap:
 a is 5 b is 10
After call to swapIntegers
 a is 5 b is 10
```

值传递



可以看到函数参数并没有被修改。因为函数 swapIntegers 使用的是值传递方式，传递给形参的是函数参数的值而不是地址。因此形参的值正确交换了，但是函数参数的值并没有发生变化。

看过错误的解决方案之后，我们知道要修改对应函数参数的值需要对应的形参使用引用传递方式。为了指定是引用传递，需要在函数头和函数原型中的每个形参类型后加上 & 符

号。这是我们需要做的唯一修改。&符号告诉编译器将函数参数的地址传递给对应的形参，而不是传递值。正确的解决方案和程序跟踪在下面给出。

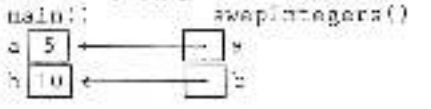
```
/*****************************************************************/
/* Correct function to swap two integer values                   */
/* using pass by reference parameters.                           */

void swapIntegers(int& a, int& b)
{
  //Define temporary variable
  int hold = a;

  //Swap the values in a and b
  a = b;
  b = hold;

  //exit void function
  return;
}
```

我们将使用下面的驱动程序来测试该函数：

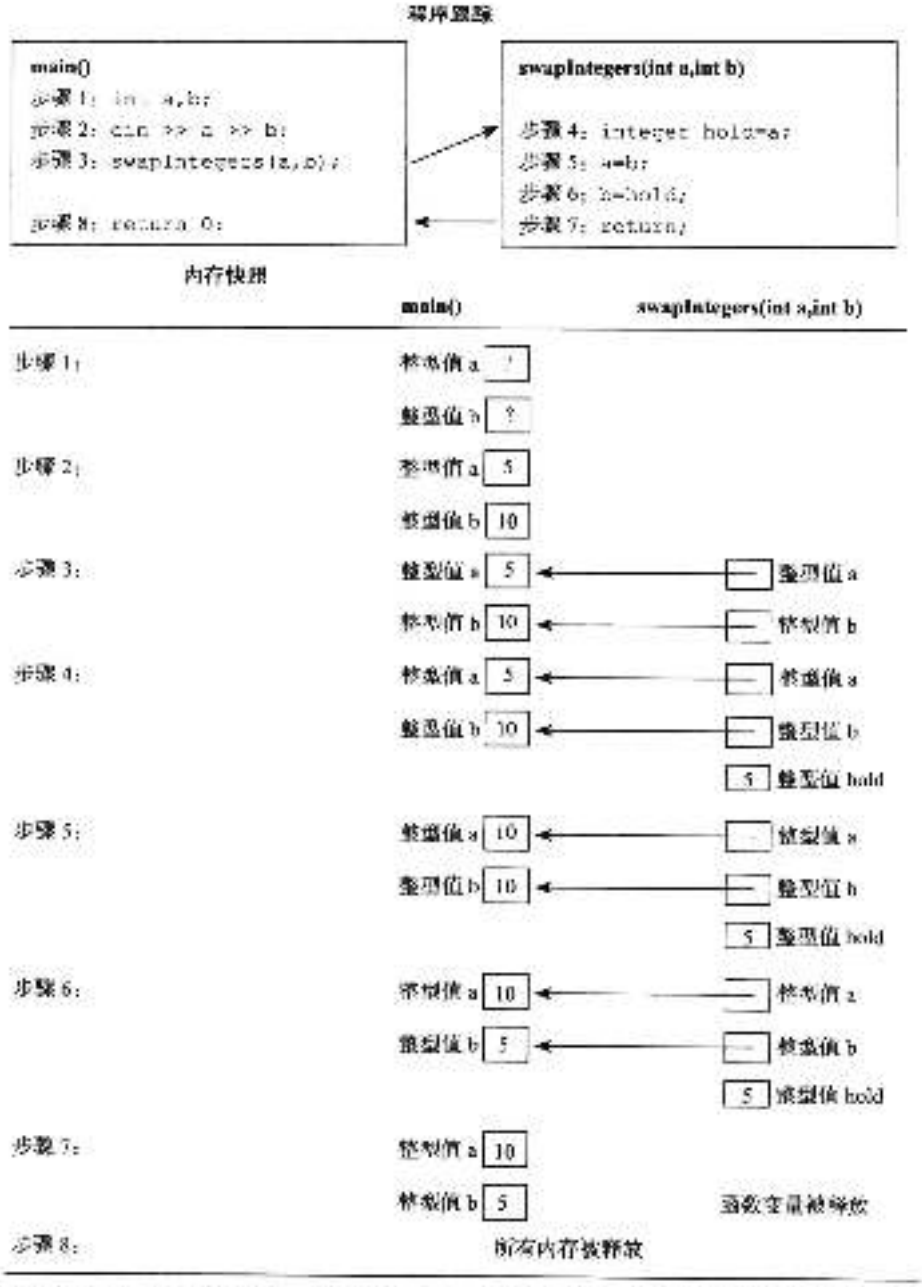
```
#include<iostream> //Required for cin, cout
using namespace std;

//Function prototype
void swapIntegers(int&, int&);
int main()
{
  //Declare variables
  int a, b;

  //Get values for a and b
  cout << "Enter two integer values: ";
  cin >> a >> b;

  //Print values of a and b before the swap
  cout << "Before swap:\n a is " << a << " b is " << b << endl;

  //Call swap function and print values
  swapIntegers(a,b);
  cout << "After call to swapIntegers\n a is " << a
       << " b is " << b << endl;

  //Exit main
  return 0;
}
```

我们的驱动程序的一次示例的输出结果如下：

```
Enter two integer values: 5 10
Before swap:
  a is 5 b is 10
After call to swapIntegers
  a is 10 b is 5
```



为了进一步说明引用传递，我们跟踪了程序的执行过程，其中忽略了 cout 语句。其中的直线箭头用来表示形参引用，或者指向使用引用传递时的函数参数。



函数参数和形参在顺序、类型和数目上都必须匹配，但是名字可以不同。在上面的例子中，我们的函数参数和形参采用了相同的标识符。因为标识符是在不同的代码块中定义的，所以这是允许的。但是，要注意到 main 函数中定义的 a 和 b 与函数头中定义的 a 和 b 分配的是不同的内存地址。对应于引用传递的函数参数必须是变量，因为它们的值是可被修改的。常量和表达式不能作为参数进行引用传递。

修改

1. 修改前一节中所开发的驱动程序，在其中加入下面的函数调用：

```
swapIntegers(10, 5);
```

当你编译并运行程序时会出现什么结果？解释原因。

2. 修改前一节中开发的驱动程序，在其中加入下面的函数调用：

```
swapIntegers(a, a + b);
```

当你编译并运行程序时会出现什么结果？解释原因。

练习

在练习 1 ～ 4 中，与执用函数 swapIntegers 有关。对于不合法的调用，解释为什么调用不合法。对于合法的调用，给出调用前后的内存快照。

1.
```
int x=1, y=5;
...
swapIntegers(x,y);
```

2.
```
swapIntegers(10,5);
```

3.
```
int x=1, y=2;
...
swapIntegers(x, y+5);
```

4.
```
double x=1.5, y=3.2;
...
swapIntegers(x,y);
```

5. 下面程序的输出是什么？

```
#include<iostream>
using namespace std;

void fun_ch3(int first, int& second);

int main()
{
    int n1(3), n2(10);
    fun_ch3(n1,n2);
    cout << n1 << endl << n2 << endl;
    return 0;
}
void fun_ch3(int first, int& second)
{
    first++;
    second += 2;
    return;
}
```

6.3.3 存储类型和作用域

到现在为止所给出的示例程序中，我们所声明的对象都位于 main 函数和自定义函数中，并且我们将自定义函数的原型放在 main 函数之前。在 main 函数之前定义一个对象也是允许的。这将会影响对象的作用域（scope）。这里的作用域是指在程序中可以合法引用该对象的范围。作用域有时也被定义为对象在程序中可见或可访问的范围。确定函数或对象的作用域是很重要的，因为对象的作用域直接关系到它的存储类型（storage class）。我们会讨论四种存储类型：自动型、外部型、静态型和寄存器型。

首先，我们将说明局部（local）作用域和全局（global）作用域（或文件作用域）之间的差别。局部对象定义在某个函数中，因此其包括了形参和其他任何在函数中声明的对象。局部对象只能在定义它的函数中被访问。当函数执行时，局部对象有一个相应的值，但当函数执行完之后，对象的值就不再保留。全局对象被定义在 main 函数和其他自定义函数之外。全局对象的定义在所有函数之外，所以它可以被程序文件中的任何函数所访问。自动型存储类型（automatic storage class）用来表示局部对象，这是它的缺省存储类型，也可以在类型限定符前加上关键字 auto 表示其存储类型为自动型。外部存储类型（external storage class）用来表示全局对象。

考虑包含下面语句的程序：

```
#include <iostream>
using namespace std;

int count(0);
...
int main()
{
   int x, y, z;
   ...
}
int calc(int a, int b)
{
   int x;
   count += x;
   ...
}
void check(int sum)
{
   count += sum;
   ...
}
```

对象 count 是全局的，因此可以被函数 calc() 和 check() 所引用。对象 x、y 和 z 是局部对象，只能在 main 函数中被引用；类似地，对象 a、b 和 x 是局部对象，只能在函数 calc() 中被引用，sum 是局部变量，只能被在 check() 中被引用。注意，程序中有两个名为 x 的局部对象，它们是作用域不同的两个不同对象。

分配给全局对象的内存在程序的整个持续时间内都被保持。虽然全局对象可以在函数中被引用，但我们不鼓励使用全局对象。一般而言，在给函数传递信息时优先使用参数，因为参数在函数原型中是明确的，而全局对象在函数原型中是不可见的。应该避免使用一个非常量的全局对象。

函数名也具有外部存储类型，因此它可以被其他函数调用。函数原型和在任何函数外被包含的 #include 指令也都是全局的，因此它们对于程序中的其他函数都是可用的；这也解释了为什么我们不需要在每个调用了数学类函数的函数中包括头文件 cmath。

静态存储类（static storage class）用于指出局部对象的内存在整个程序执行期间都应当被保持。因此，如果通过在对象类型前加上关键字 static 而指定函数中的一个局部对象为静态存储类型，那么该对象在程序退出定义它的函数之前会一直存在。一个 static 对象可以用来计算函数被调用的次数，因为 count 的值在一个函数调用另一个函数时会一直保持。下面的程序说明了 static 存储类型的用法：

```
#include<iostream>
using namespace std;

void ch6_static();

int main()
{
    ch6_static();
    ch6_static();
    ch6_static();
}

void ch6_static()
```

```
{
    int x(0);
    static int count(2);
    x++;
    count++;
    cout << x << ',' << count;
    return;
}
```

在函数 ch6_static() 第一次被调用时，对象 x 和 count 被定义并初始化。由于 count 是 static 类型的，并且其值将一直保留，所以在后续对函数的调用中对它的初始化都将被忽略。程序的输出如下：

```
1,1
1,2
1,3
```

关键字 register 用在一个应存放在寄存器或高速内存中的对象类型限定符之前。因为访问寄存器要比访问内存快，所以这种存储类型用于被频繁访问的值。因为可用的高速内存有限，所以不能保证请求一定得到满足，而且对象最终可能还是被分配在普通内存中。

6.4 解决应用问题：计算重心

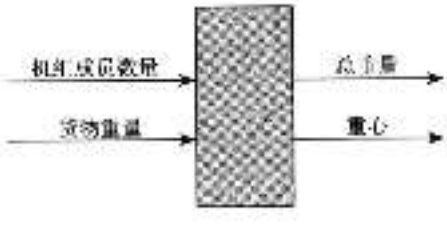
当驾驶飞机时，在起飞前确定飞机的重量和重心是很重要的。如果飞机超载，将很难甚至不能起飞（achieve lift）；如果重心在设计的限制之外，那么飞机可能很难控制。本节，我们将完成对飞机总重量和重心的计算。

1. 问题描述

基于机组成员的数量（最多两名）和货物的重量（最大 5000 磅），确定飞机的总重量和重心。为了计算重心，程序应当将每个重量与其到机头的距离相乘。把这些结果（称作矩）加在一起，将得到的和除以总重量即可得到重心。飞机空载时自重已知为 9021 磅，其空载时的重心距离机头 305 英寸。因此，其空载时的矩为 2 751 405 英寸－磅。飞机最多可装载 540 加仑的燃料。为了简化问题，我们假定飞机起飞时油箱是加满的，且每加仑燃料重 6.7 磅，则燃料的矩为 1 169 167.3 英寸－磅。

2. 输入/输出描述

下面的 I/O 图表明程序的输入为机组成员数量（1 或 2）以及货物的重量。输出是关于飞机总重量和重心的报告。所有的数据对象都将使用内建数据类型 int 和 double。



3. 用例

假设飞机上有两名机组成员，货物总重量为 100 磅。假定每个人的平均重量为 160 磅，我们可以按照下面的方式计算得到机组成员的矩：

成员数量 * 每人的平均重量 * 成员到机头的距离

货物的矩按照下面方式计算：

货物重量 * 货舱到机头的距离

飞行手册上给出机组成员座位到机头的距离为120英寸，货舱到机头的距离为345英寸。因此成员的矩为：

$$2*160*120 = 38\,400 \text{ 英寸} - \text{磅}$$

货物的矩为：

$$100*345 = 34\,500 \text{ 英寸} - \text{磅}$$

飞机的总重量为：

成员重量 + 货物重量 + 燃料重量 + 飞机空载时自重，即 320+100+3618+9021=13059 磅

重心力矩的和除以总重量：

$$(38\,400+34\,500+2\,751\,405+1\,169\,167.3) \text{ 英寸} - \text{磅} / 13\,059 \text{ 磅}$$
$$=1\,438\,172.3 \text{ 英寸} - \text{磅} / 13\,059 \text{ 磅} = 305.802 \text{ 英寸}$$

4. 算法设计

我们首先给出分解提纲，它将解决方案分为了一系列顺序的步骤：

分解提纲

1）读取机组成员数量和货物重量；

2）计算总重量；

3）计算重心。

步骤1涉及提示用户输入必要的信息。因为成员数量和货物重量有限制，所以需要对输入数据进行错误检查。该步骤适合使用一个函数完成。步骤3需要每个矩的值，我们将写出用以计算要求的矩的函数，同时对于问题中假定的值使用全局常量来表示。细化后的伪代码如下：

```
Refinement in Pseudocode
main:  Get_Data(crew, cargo)
       calculate total_weight
       calculate center_of_gravity
       print center of gravity, total_weight
```

伪代码中的步骤已经足够详细，可以转换成C++语句：

```
/*                                                                    */
/* Program chapter5_4                                                 */
/* This program calculates the total weight and                       */
/* center of gravity of an aircraft.                                  */
#include<iostream>  //Required for cin, cout
using namespace std;

//Program Assumptions

const double PERSON_WT(160.0);           //Average weight/person
const double FUEL_MOMENT(1169167.2);     //Fuel moment for full tank
const double EMPTY_WT(9021.0);           //Standard empty weight
const double EMPTY_MOMENT(2751405.0);    //Standard empty moment
const double FUEL_WT(3618.0);            //Full fuel weight
const double CARGO_DIST(345.0);
const double CREW_DIST(120.0);
```

```
//function prototypes

double CargoMoment(double);
double CrewMoment(int);
void GetData(int&, double&);

int main()
{

  //Declare objects.
  int crew; //number of crew on board (1 or 2)
  double cargo;   //weight of baggage, pounds
  double total_weight, center_of_gravity;

  //Set format flags.
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(1);

  GetData(crew, cargo);

  total_weight = EMPTY_WT + crew*PERSON_WT + cargo
               + FUEL_WT;

  center_of_gravity = (CargoMoment(cargo) + CrewMoment(crew)
                    + FUEL_MOMENT + EMPTY_MOMENT)/total_weight;

  cout << endl << "The total weight is " << total_weight
       << " pounds. \n"
       << "The center of gravity is " << center_of_gravity
       << " inches from the nose of the plane.\n";
  return(0);
}//end main
/*-------------------------------------------------------------*/
double CargoMoment(double weight)
{
   return(CARGO_DIST*weight);
}//end CargoMoment
/*-------------------------------------------------------------*/
double CrewMoment(int crew)
{
   return(CREW_DIST*crew*PERSON_WT);
}//end CrewMoment
/*-------------------------------------------------------------*/
void GetData(int& crew, double& cargo)
{
  cout << "enter number of crew members (Maximum of 2) ";
  cin >> crew;
  while(crew <= 0 || crew > 2)
  {
    cout << endl << crew
         << " is an invalid entry\n"
         << " re enter number of crew, 0 < crew <= 2  ";
    cin >> crew;
  }//end while
  cout << crew << " crew members, thank you.\n\n";
  cout << "enter weight of cargo (Maximum of 5000 lbs) ";
  cin >> cargo;
  while(cargo < 0 || cargo > 5000)
  {
```

```
    cout << endl << cargo
         << " is an invalid entry"
         << " re-enter cargo weight. 0 < cargo <= 5000\n  ";
    cin >> cargo;
  }//end while
  cout << cargo << " pounds of cargo loaded.  Thank you.\n\n";
  return;
}//end getdata
/*------------------------------------------------------------------/
```

5. 测试

如果我们使用用例中的数据，则与程序的交互过程如下（计算出的总重量和重心与人工计算结果匹配）：

```
enter number of crew members (Maximum of 2) 2
2 crew members, thank you.
enter weight of cargo (Maximum of 5000 lbs) 100
100.0 pounds of cargo loaded.  Thank you.

The total weight is 13059.0 pounds.
The center of gravity is 332.3 inches from the nose of the plane.
```

修改

1. 修改 GetData() 函数，完成错误检查并更正错误的数据，如当输入的成员数目为浮点或者字符数据时，测试你的函数。
2. 修改程序，增加一个距离机头 522 英寸的货仓，其中的货物最大重量为 1000 磅。

6.5 随机数

随机数（random number）序列不是由方程定义的，而是通过某些特征定义的。这些特征包括最大、最小值、平均值，以及数值发生的概率。随机数序列可以通过实验生成，如抛硬币、投色子，或者选择有标号的球。随机数序列也可以使用计算机生成。

许多工程问题的解决方案中都需要使用随机数。某些情况下，随机数用于设计一个复杂问题的仿真。为了分析结果，仿真可以反复运行，每次运行都是实验的一次重复。我们还使用随机数来近似噪声序列。例如，我们听广播时的静音状态就是一个噪声序列，如果我们测试一个使用输入数据文件的程序，该输入数据文件表示一个广播信号，则我们可能希望生成噪声并将它加到演讲信号或音乐信号中，以提供更真实的信号。

工程应用中常常需要分布在特定值范围内的随机数。例如，我们可能希望生成 1 ~ 500 之间的随机数，又或者我们希望生成 −5 ~ 5 之间的随机浮点数。现在我们将讨论生成在两个特定值之间的随机数。随机数生成的概率是相等的，这意味着如果随机数在 1 ~ 5 之间的整数中生成，那么集合 1、2、3、4、5 中的每个整数出现的机会是相等的。另一种说法就是在一次运行中每个整数出现的概率都是 0.2。一个在给定集合中的值也被称为均匀随机数或均匀分布随机数。

6.5.1 整数序列

在标准 C++ 库中包含了一个函数 rand 用来生成 0 ~ RAND_MAX 的随机整数，其中

RAND_MAX 是一个系统相关的整数值，定义在 cstdlib 中。（RAND_MAX 的一个常见值为 32 767。）函数 rand 没有输入参数，可以通过表达式 rand() 进行调用。因此，要生成并打印两个随机数的序列，可以使用下面的语句：

```
cout << "random numbers: " << rand() << " " << rand() << endl;
```

包含该语句的程序每次执行时，打印出的两个数值都是相同的。因为 rand 函数是按照一个特定序列生成整数的。（因为这个序列最后会开始重复，所以有时它调用一个伪随机序列而不是随机序列）。但是，如果我们在相同的程序中生成更多的随机数，它们的值是不同的。因此，下面的一组语句生成 4 个随机数：

```
cout << "random numbers: " << rand() << " " << rand() << endl;
cout << "random numbers: " << rand() << " " << rand() << endl;
```

在程序中，函数 rand() 每次被调用时，将生成一个新的值；但是，每次程序运行时，都会生成相同的数值序列。

为了使程序在每次运行时都生成一个不同的数值序列，我们可以给随机数生成器提供一个新的随机数种子（random number seed）。函数 srand()（来自 cstdlib）为随机数生成器指明种子；对于每个种子，rand() 都会生成一个新的随机数序列。函数 srand() 的参数是一个无符号整数，它在计算中用于初始化序列，但种子的值并不是序列的第一个值。如果在调用 rand() 函数之前没有使用 srand() 函数，那么计算机将假定种子的值为 1。因此，如果你指定种子的值为 1，得到的序列将与不指定种子的值而得到的序列相同。

在下一个程序中，用户将被要求输入一个种子值，然后程序会生成 10 个随机数。如果每次执行程序时，用户都输入相同的种子，那么每次得到的 10 个随机数的集合都相同；如果每次输入的种子不同，那么每次生成的 10 个随机数的集合也不同。函数 rand() 和 srand() 的原型语句包含在 cstdlib 中。

```
/*  ------------------------------------------------------------  */
/*  Program Chapter6_5                                            */
/*                                                                */
/*  This program generates and prints ten                         */
/*  random integers between 1 and RAND_MAX.                       */

#include <iostream> //Required for cin, cout
#include <cstdlib> //Required for srand(), rand().
using namespace std;

int main()
{
   //  Declare objects.
   unsigned int seed;

   //  Get seed value from the user.
   cout << "Enter a positive integer seed value: \n";
   cin >> seed;
   srand(seed);  //Seed random number generator.

   //  Generate and print ten random numbers.
   cout << "Random Numbers: \n";
   for (int k=1; k<=10; k++)
   {
      cout << rand() << " ";  //Print a random number
   }
```

```
    cout << endl;
    // Exit program.
    return 0;
}
/*--------------------------------------------------------------------*/
```

在 Linux 系统上使用 g++ 编译、一个示例输出如下所示：

```
Enter a positive integer seed value:
129
Random Numbers:
1289593395 1692901613 420005879 748133630 726553275 289139931
837365195 1568889022 45168426 1888841001
```

在你的计算机系统上使用两个程序进行实验；使用相同的种子生成相同的数字，并使用不同的种子生成不同的数字。

因为 rand() 和 srand() 的原型语句包含在 cstdlib 中，所以我们不需要单独在程序中包含它们的原型。但是分析这些原型语句是有益的。因为 rand() 函数返回一个整数而没有输入，它的原型语句为

```
int rand();
```

因为 srand() 函数没有返回值，同时具有一个无符号函数作为输入，所以它的原型语句为

```
void srand(unsigned int);
```

使用 rand() 函数生成一个给定范围内的随机整数是很容易的。例如，假定我们想生成 0 ~ 7 之间的随机整数。下面的语句首先生成一个位于 0 ~ RAND_MAX 之间的随机数，然后使用取模操作符来计算随机数对于整数 8 的模：

```
x = rand()%8;
```

取模操作的结果是使用 rand() 除以 8 后得到的余数，所以 x 的值是在 0 ~ 7 之间的整数。

假如我们想生成一个 -25 ~ 25 之间的随机整数。可能的整数的数目为 51，可以通过下面的语句得到一个在该范围内的随机数

```
y = rand()%51 - 25;
```

该语句首先生成一个 0 ~ 50 之间的数，然后将该数减去 25，就得到一个在 -25 ~ 25 之间的数值。

现在我们可以编写一个生成两个给定整数 a ~ b 之间的整数的函数。函数首先计算 n，它是 a 和 b 之间（包含 a、b 在内）的某个数，其值等于 b - a + 1。然后函数对 rand() 函数进行取模操作，生成一个位于 0 ~ n - 1 之间的新整数。最后，将下限 a 加到新生成的值上，得到一个位于 a ~ b 之间的数。所有的步骤都可以放在一个表达式中，并通过 return 语句返回：

```
/*--------------------------------------------------------------------*/
/*  This function generates a random integer                          */
/*  between specified limits a and b (a<b).                           */

int rand_int(int a, int b)
{
   return rand()%(b-a+1) + a;
}
/*--------------------------------------------------------------------*/
```

为了说明该函数的用法，下一个程序生成并且打印出了用户指定区间内的10个整数。用户还可以输入种子来初始化序列：

```
/*-------------------------------------------------------------*/
/*  Program Chapter_5                                          */
/*                                                             */
/*  This program generates and prints ten random               */
/*  integers between user specified limits.                    */

#include <cstdlib> //Required for srand(), rand().
#include <iostream> //Required for cin, cout.
using namespace std;

//  Function prototype.
int rand_int(int a, int b);

int main()
{
   //  Declare objects.
   unsigned int seed;
   int a, b;

   //  Get seed value and interval limits.
   cout << "Enter a positive integer seed value: \n";
   cin >> seed;

   //Seed the random number generator.
   srand(seed);
   cout << "Enter integer limits a and b (a<b): \n";
   cin >> a >> b;

   //  Generate and print ten random numbers.
   cout << "Random Numbers: \n";
   for (int k=1; k<=10; ++k)
   {
      cout << rand_int(a,b) << ' ';
   }
   cout << endl;

   //  Exit program.
   return 0;
}

/*-------------------------------------------------------------*/
/*  This function generates a random integer                   */
/*  between specified limits a and b (a<b).                    */

int rand_int(int a, int b)
{
   return rand()%(b-a+1) + a;
}
/*-------------------------------------------------------------*/
```

程序运行生成的一组数据如下：

```
Enter a positive integer seed value:
1
Enter integer limits a and b (a<b):
-5 5
Random Numbers:
3 1 4 -4 0 4 3 0 -3 3
```

修改

使用本节所开发的程序生成下列每个范围内的若干组随机数集合，并使用不同种子值。

1. 0 ~ 500
2. −10 ~ 200
3. 50 ~ −10
4. −5 ~ 5

6.5.2 浮点序列

在许多工程问题中，我们需要生成在给定区间 [a, b] 上的随机浮点值。将一个 0 ~ RAND_MAX 之间的整数转化成一个 a ~ b 之间的浮点值有三步。首先用 rand() 函数所生成的数除以 RAND_MAX，得到一个 0 ~ 1 之间的浮点数。然后将这个 0 ~ 1 之间的浮点数乘以 (b−a)，即乘以区间长度，得到一个位于 0 ~ (b−a) 之间的数。最后将这个 0 ~ (b−a) 之间的数加上 a，即可得到一个 a ~ b 之间的数。这三步可以放在一个表达式中完成，并通过 return 语句返回。

```
/*-----------------------------------------------------*/
/*  This function generates a random                   */
/*  double value between a and b.                      */

double rand_float(double a, double b)

 return ((double)rand()/RAND_MAX)*(b-a) + a;
}
/*-----------------------------------------------------*/
```

注意在将整数 rand() 转换成 double 时要使用强制转换操作符，以保证除法的结果是一个 double 类型的值。

本节前面给出的程序进行简单修改后就可以生成并打印浮点值。经过修改后的程序运行得到的一组示例输出值如下：

```
Enter a positive integer seed value:
82
Enter limits a and b (a<b):
 0 5
Random Numbers:
3.64335 -1.51118 2.9092 2.21546 -4.32425 -1.22527
0.709869 -3.41159 -4.86308 -0.958863
```

修改

对前面生成整数的程序进行修改，使之变成一个可以生成用户指定范围内的 10 个浮点值的程序。然后生成下列每个范围内的若干组随机数集合，并使用不同种子值：

1. 0.0 ~ 1.0
2. 0.1 ~ 1.0
3. −5.0 ~ −4.5
4. −5.1 ~ 5.1

6.6 解决应用问题：仪器可靠性

如果某个设备的使用环境十分危险或者是不易接触，则必须对该设备进行可靠性分析。

通过对统计数据和概率进行研究，可以得到用于分析仪器可靠性的公式，这里的可靠性是指仪器正常工作时间的比例。因此，如果一个组件的可靠性为 0.8，这表示它在 80% 的时间里都能正常工作。如果已知单个组件的可靠性，也可以计算出组件组合起来的可靠性。考虑图 6.5 中的图示，对于流水线设计中从点 a 到点 b 流动的信息，所有的三个组件都必须正常工作。在并行设计中，只要三个组件中有一个正常工作，信息就可以从点 a 流向点 b。

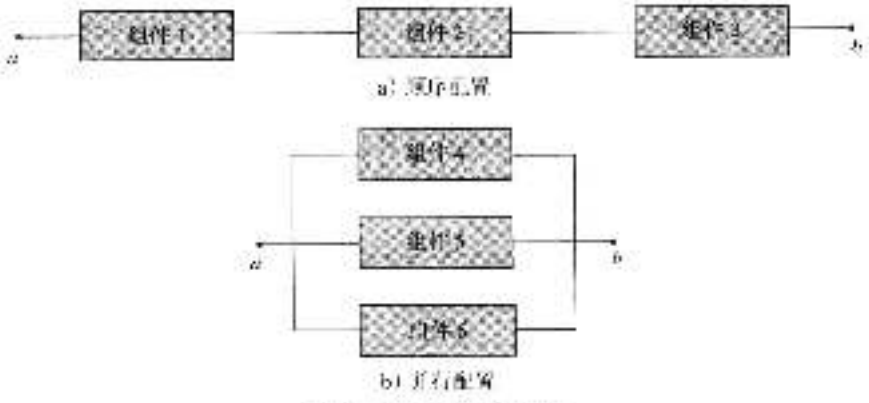


图 6.5 顺序和并行配置

如果我们已知单个组件的可靠性，那么通过两种方式可以确定组件的特定组合形式的可靠性：解析可靠性可以通过使用统计和概率知识的定理推论计算出来。使用计算机仿真可以得到对这种可靠性的估计值。

考虑图 6.5a 中的顺序配置方式。如果三个组件的可靠性都为 r，那么这个顺序配置方式的可靠性就是 r^3。因此，如果每个组件的可靠性为 0.8（这意味着组件在 80% 的时间里都能正常工作），那么顺序配置方式的解析可靠性为 0.8^3 或 0.512。这表示顺序配置方式在 51.2% 的时间里都能正常工作。

考虑图 6.5b 中的并行配置方式。只要有一个组件不失效，配置就可以正常工作。描述一个并行配置方式的可靠性公式如下：

$$R = 1 - \prod_{i=0}^{n}(1 - r_i)$$

其中，R 是配置的可靠性，n 是组件的数量，r_i 是第 i 个组件的可靠性。

如果所有的组件都有相同的可靠性，那么图 6.5b 中并行配置的可靠性公式可以简化为 $3r - 3r^2 + r^3$。因此，如果每个组件的可靠性为 0.8，那么并行配置的解析可靠性为 $3(0.8) - 3(0.8)^2 + (0.8)^3$ 或者 0.992，并行配置方式在 99.2% 的时间里都应当工作正常。

你的直觉可能告诉你并行配置更可靠，因为只要有一个组件工作正常，这个配置就可以工作，而顺序配置方式则要求三个组件都正常工作才可以正常工作。

我们也可以使用计算机仿真产生的随机数对两种设计方式的可靠性进行估算。首先，我们需要仿真单个组件的性能。如果一个组件的可靠性为 0.8，那么它在 80% 的时间里都可以正常工作。为了仿真这一性能，我们可以生成一个 0 ~ 1 之间的随机值，如果值在 0 ~ 0.8 之间，我们就认为组件工作正常。否则我们就认为它失效了。（我们也可以使用一个 0 ~ 0.2 之间的值表示失效，0.2 ~ 1.0 之间的值表示组件工作正常。）为了仿真三个组件的顺序设计方式，我们可以生成 3 个 0 ~ 1 之间的随机浮点数。如果 3 个数值都不超过 0.8，那么这次试验中的设计方式正常工作；如果有一个或以上的值超过 0.8，那么这次试验中的设计方式未正常工作。如果我们运行上千次这样的试验，则可以计算出在所有设计方式正常工作时的时间比例。这种仿真估计是对于使用解析法计算可靠性的一种近似。

为了估计并行设计方式的可靠性，该设计中单个组件可靠性为 0.8，我们可以再生成 3 个在 0 ~ 1 之间的随机浮点数。如果 3 个数值都不超过 0.8，那么这次试验中的设计方式

正常工作；如果有一个或以上的值超过0.8，那么这次试验中的设计方式未正常工作。为了通过仿真估计可靠性，我们用设计方式正常工作的试验次数除以总的试验次数。

如前面讨论中指出的，我们可以使用计算机仿真对解析计算的结果进行验证。因为随着试验次数的增加，仿真可靠性将不断接近解析计算可靠性。在以解析的方式计算仪器可靠性的过程中也会有很难进行解析计算的情况。在这些情况下，计算机仿真则可以对可靠性给出好的估计。

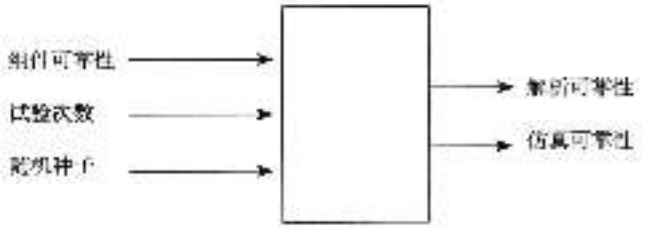
开发一个程序，使用仿真结果比较图6.5中顺序和并行配置方式的解析可靠性。允许用户输入单个组件的可靠性和使用仿真的试验次数。

1. 问题描述

针对使用三个组件的顺序配置方式和使用三个组件的并行配置方式，比较它们的解析可靠性和仿真可靠性。假定所有组件的可靠性相同。

2. 输入/输出描述

下面的I/O示意图给出了程序的输入和输出。输入是组件可靠性、试验次数和用于初始化随机数序列的随机数种子。输出由顺序方式和并行方式的解析可靠性和仿真可靠性组成。



3. 用例

在用例中，我们使用的组件可靠性为0.8，并进行3次试验。因为每次试验需要3个随机数，假如前9个生成的随机数如下。（这些是以6 666为种子，使用rand float函数生成的0～1之间的数值。）

第一组3个随机数为

```
0.939775     0.0422243     0.929037
```

第二组3个随机数为

```
0.817733     0.211689     0.9909
```

第三组3个随机数为

```
0.0377037     0.103508     0.407272
```

通过每组3个随机数，我们可以判断顺序配置方式是否正常工作以及并行配置方式是否正常工作。对于第一组和第二组中的3个随机数，有两个值超过0.8，所以只有并行配置方式正常工作。在第三组随机数的条件下，两种配置方式都工作正常。因此，解析计算的结果（本节前面已计算出来）和三次试验的仿真结果如下：

```
Analytical Reliability:
Series: 0.512 Parallel:  0.992
```

```
Simulation for 3 Trials
Series: 0.333333 Parallel: 1
```

当我们增加试验次数时，仿真结果将接近解析结果。如果我们改变随机数种子，那么仿真结果也会变化，即使是在只有三次试验的情况下。

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤：

分解提纲

1）读取组件可靠性、试验次数、随机数种子；

2）计算解析可靠性；

3）计算仿真可靠性；

4）打印出可靠性的比较情况。

步骤 1 中提示用户输入必要的信息，并获取相关信息。步骤 2 使用本节前面给出的公式计算出解析可靠性。因为计算过程是明确的，所以我们在 main 函数中进行计算。步骤 3 包括了一个生成随机数的循环，用以确定每次试验中的配置方式是否能正常工作。函数 rand_float() 用于在循环中计算随机数。步骤 4 中，我们打印出结果的比较情况。图 6.1 中给出了解决方案的结构图。细化后的伪代码如下：

```
main:    read component reliability, number of trials,
           and random number seed
         compute analytical reliabilities
         set series_success to zero
         set parallel_success to zero
         set k to 1
         while k <= number of trials
             generate three random numbers between 0 and 1
             if each number <= component reliability,
                 increment series_success by 1
             if any number <= component reliability,
                 increment parallel_success by 1
             increment k by 1
         print analytical reliabilities
         print simulation reliabilities
```

伪代码中的步骤已经足够详细，可以转换成 C++ 语句。在程序中我们也包括了 rand_float() 函数：

```
/*-------------------------------------------------*/
/*  Program Chapter6_7                             */
/*                                                 */
/* This program estimates the reliability          */
/* of a series and a parallel configuration        */
/* using a computer simulation.                    */

#include <iostream> //Required for cin, cout.
#include <cstdlib> //Required for srand(), rand().
#include <cmath>  //Required for pow().
using namespace std;

//  Function prototypes
double rand_float(double a, double b);

int main()
```

```
{
   // Declare objects.
   unsigned int seed;
   int n;
   double component_reliability, a_series, a_parallel,
          series_success(0), parallel_success(0),
          num1, num2, num3;
   // Get information for the simulation.

   cout << "Enter individual component reliability: \n";
   cin >> component_reliability;
   cout << "Enter number of trials: \n";
   cin >> n;
   cout << "Enter unsigned integer seed: \n";
   cin >> seed;
   srand(seed);
   cout << endl;

   // Compute analytical reliabilities.
   a_series = pow(component_reliability,3);
   a_parallel = 3*component_reliability
                - 3*pow(component_reliability,2)
                + pow(component_reliability,2);

   // Determine simulation reliability estimates.
   for (int 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++;
     }
   }

   // Print results.
   cout << "Analytical Reliability \n";
   cout << "Series: " << a_series << "  "
        << "Parallel: " << a_parallel << endl;
   cout << "Simulation Reliability " << n << " trials \n";
   cout << "Series: " << (double)series_success/n << " Parallel: "
        << (double)parallel_success/n << endl;

   // Exit program.
   return 0;
}

/*-------------------------------------------------------*/
/* This function generates a random                      */
```

```
/*  double value between a and b.                                    */
double rand_float(double a, double b)
{
  return ((double)rand()/RAND_MAX)*(b-a) + a;
}
/*-------------------------------------------------------------------*/
```

5. 测试

如果我们使用用例中的数据，则与程序的交互过程如下，程序的输出与我们手工计算的数据匹配：

```
Enter individual component reliability:
0.8
Enter number of trials:
5
Enter unsigned integer seed:
6666

Analytical Reliability
Series: 0.512  Parallel: 0.992
Simulation Reliability, 5 trials
Series: 0.333333  Parallel: 1
```

下面的两次仿真结果表明随着试验次数的增加，仿真结果接近于解析结果：

```
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: 0.54  Parallel: 0.97
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: 0.514  Parallel: 0.995
```

修改

这些问题与本节开发的用来比较解析结果和仿真结果的程序有关。

1. 假定组件可靠性为 0.85，使用程序计算试验次数为 100、1000 和 10 000 时的仿真结果。
2. 假定组件可靠性为 0.75。使用程序计算出试验次数为 1000 时的仿真结果，在试验中使用 5 个不同的随机数种子，并与解析结果进行比较。
3. 如果要求顺序方式的可靠性为 0.7，那么要求组件可靠性为多少？（提示：使用解析可靠性公式）。使用程序验证你的答案。
4. 如果要求并行方式的可靠性为 0.9，那么要求组件可靠性为多少？在这种情况下使用解析可靠性公

式不是那么简单。如果你的计算器不能计算多项式的根，则使用程序进行实验，直到接近所要求的可靠性为止。

5. 当每个组件的 r_c 值相同时，完成从公式 $R=1-\prod_{c=0}^{3}(1-r_c)$ 到 $3r-3r^2+r^3$ 的简化过程。

6.7 定义类方法

在第2章和第3章中，我们设计了一个名为Point的部分自定义的数据类型结构，其类声明如下：

```
/*-------------------------------------------------------------*/
/* Point class Chapter3.7                                      */
/* Filename: Point.h                                           */
class Point
{
   //Type declaration statements
   //Data members:
   private:
   double xCoord, yCoord; //Class attributes

   public:
   //Declaration statements for class methods
   //Constructors for Point class
   Point(); //default constructor
   Point(double x, double y); //parameterized constructor

  //Overloaded operators
  double operator  (const Point& p2) const;
  bool operator ==(const Point& p2) const;
};
/*-------------------------------------------------------------*/
```

在Point类中声明的构造函数和重载函数是Point类的成员函数（或称成员方法）。回想第2章中定义的可对类对象进行操作的方法。现在我们的Point类的定义中包括的操作十分有限。应用程序可以使用两种构造方法中的任何一个对Point类型的对象进行声明和初始化，但是应用程序不能访问Point对象的属性。因此，一旦一个Point对象被初始化，它的值就不能被修改，其中的私有属性的值也不能被应用程序访问。本节中，我们将为我们自定义的数据类型Point定义一组访问方法和修改方法（mutator method），以提供更好的公共接口。

6.7.1 公共接口

一个设计良好的数据类型公共接口需要提供一组完整但最小的公共方法集合，并通过封装（encapsulation）隐藏数据类型的实现。封装要求对于类属性的直接访问被限制在类自定义成员函数和友元函数（friend function）之内。关键字friend将在第10章讨论。封装和一个良好的公共接口可以保证数据类型的可维护性和扩展性。我们在Point类中使用private关键字限制对属性的访问。因此，在当前的公共接口中，应用程序不能访问或修改Point对象的值。考虑下面的测试程序：

```
#include <iostream>
#include " Point.h"
using namespace std;
int main()
{
  Point p1;
```

```
    cout << p1.xCoord << endl;
    return 0;
}
```

该程序尝试打印 Point 对象 p1 的私有数据成员 xCoord。在语法上 p1.xCoord 是正确的，但是这时编译器将生成一条错误信息，因为 xCoord 是一个私有属性。当我们使用 g++ 编译器在一台 MacBook Pro 上对程序进行编译时，将生成下面的信息：

```
Point.h: In function 'int main()':
Point.h:9: error: 'double Point::xCoord' is private
main.cpp:7: error: within this context
```

一个良好的公共接口要求一组完备的 public 方法，以支持对类的私有属性的访问。因为我们的 Point 类有两个 private 属性，我们将在我们的公共接口中增加四个方法：两个访问方法和两个修改方法。扩展后的类声明在如下所示。

```
/*-------------------------------------------------------------*/
/* Point class Chapter6_1                                      */
/* Filename: Point.h                                           */
class Point
{
  //Type declaration statements
  //Data members
  private:
  double xCoord, yCoord; //Class attributes

  public:
  //Declaration statements for class methods
  //Constructors for Point class
  Point(); //default constructor
  Point(double x, double y); //parameterized constructor

  //Overloaded operators
  double operator -(const Point& rhs) const;
  bool operator == (const Point& rhs) const;

  //Accessor Methods
  double getX() const {return xCoord;}
  double getY() const {return yCoord;}

  //Mutator Methods
  void setX(double newX);
  void setY(double newY);

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

6.7.2 访问方法

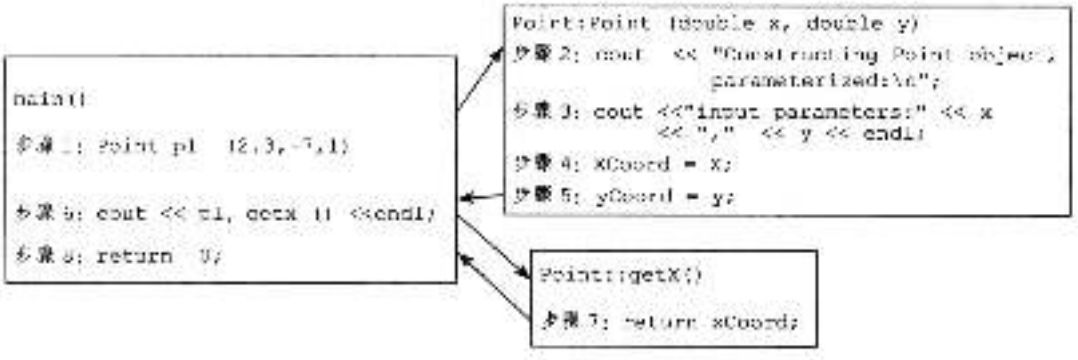
访问方法是具有返回值且参数列表为空的成员函数。一个访问方法的唯一目的就是返回一个属性的值。因为访问方法是小而简单的方法，它们可以在类声明中定义。现在的约定是将这些方法命名为 get Attribute()。注意，我们在访问方法的定义中使用了 const 限定符。访问方法对属性的权限应限于只读，使用 const 限定符可以保证这一点。任何在具有 const 限定符的方法中对一个私有属性进行修改的尝试都会导致编译错误，如在我们的测试程序中所说明的那样。为了说明访问方法的用法，我们将修改测试程序，在其中使用访问方法 getX()

打印出一个点的 x 坐标。

```
#include <iostream>
#include " Point.h"
using namespace std;
int main()
{
  Point p1(2.3, -7.1);
  cout << p1.getX() << endl;
  return 0;
}
```

因为访问方法 getX() 是 public 的，所以测试程序在编译时不会出错。在执行时将会把值 2.3 打印到标准输出上，如下所示。为了说明，图 6.6 中给出了程序跟踪和内存快照。

```
Ingbers-MacBook-Pro:Programs jaingber$ g++ main.cpp Point.cpp
Ingbers-MacBook-Pro:Programs jaingber$ ./a.out
Constructing Point object, parameterized:
Input parameters: 2.3,-7.1
2.3
```



内存快照：main()

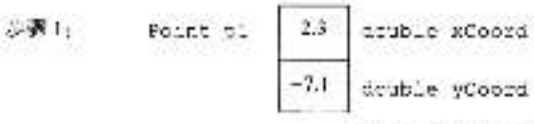


图 6.6 程序跟踪和内存快照

注意，g++ 编译器将可执行代码写入了一个名为 a.out 的文件中。命令 ./a.out 要求操作系统在当前工作目录 (./) 中找到文件 a.out 并执行。

6.7.3 修改方法

修改方法具有一个以上输入参数，返回类型为 void。修改方法的唯一目的就是修改调用对象的值。输入参数允许应用程序指定一个新值赋给某个属性。现在的约定是将这些方法命名为 set Attribute()。注意我们在修改方法的原型中没有如访问方法的原型中一样使用 const 限定符。我们在类实现中定义了修改方法。扩展后的类实现如下。

```
/*-------------------------------------------------------*/
/* Point Class Chapter6_2                                */
/* filename: Point.cpp                                   */
```

```
#include " Point.h"      //Required for Point
#include <iostream>    //Required for cout
#include <cmath>       //Required for sqrt() and pow()
using namespace std;

//Parameterized constructor
Point::Point(double x, double y)
{
  //input parameters x,y
  cout << " Constructing Point object, parameterized: \n" ;
  cout << " Input parameters: "  << x << " ,"  << y << endl;
  xCoord = x;
  yCoord = y;
}

//Default constructor
Point::Point()
{
   cout << " Constructing Point object, default: \n" ;
   cout << " initialising to zero"  << endl;
   xCoord   0.0;
   yCoord = 0.0;
}
//Distance Formula
double Point::operator  (const Point& rhs) const
{
  double t1, t2, d;
  t1 = rhs.xCoord - xCoord; //(x2-x1)
  t2 = rhs.yCoord - yCoord; //(y2-y1)
  d = sqrt( pow(t1,2) + pow(t2,2) );
  return d;
}
bool Point::operator ==(const Point& rhs) const
{
  if(rhs.xCoord == xCoord &&
     rhs.yCoord == yCoord )
  {
    return true;
  }
  else
  {
    return false;
  }

}
void Point::setX(double xVal)
{
   xCoord = xVal;
}
void Point::setY(double yVal)
{
   yCoord = yVal;
}
```

类方法对调用对象所有的数据成员都具有访问权。在修改方法 setX() 的实现中我们看到输入参数 xVal 的值被赋给了私有属性 xCoord。类似地，在修改方法 setY() 中，输入参数 yVal 的值被赋给私有属性 yCoord。

方法定义：方法是类的成员函数。方法的定义由函数头及其后的语句块组成。函数头必须包括类的名称，其后为域解析操作符和方法名。

语法

```
返回类型 类名::方法名(参数列表)[限定符]
{
   //语句块
}
```

示例

```
double Point::getX() const
{
   return xCoord;
}
void Point::setX(double newX)
{
   xCoord = newX;
}
```

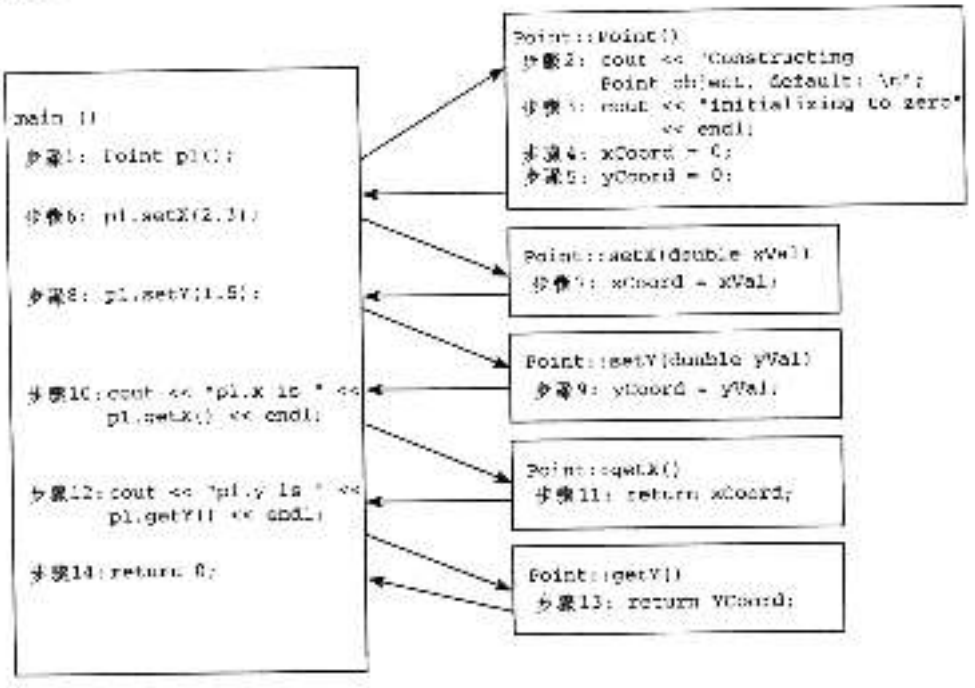
在我们关于平面上某一点的定义中，并未限制可以赋给 x 和 y 坐标的合法数值集合。任何实数值都是合法的，所以在修改方法中不需要对输入参数进行验证。在定义一个新的数据类型时，通常会对可以赋给其一个或多个数据成员的值进行某些限制。如果我们定义一种新的数据结构来表示一个圆，这里我们将圆用平面上的一个点（圆心）和半径来表示，那么给半径赋一个负值是不合理的。为了避免创建一个无意义的圆，在每个修改方法中，赋一个新值之前都应当进行测试。如果输入数据不在允许的值的范围内，则在方法中需要对这样的错误进行处理。因此，一个修改方法允许应用程序对对象的值或者状态进行修改，但是要保持对对象状态的控制。这对由一个非法数据导致的错误发生时减少错误的传播是十分有用的。在第 10 章中将讨论在一个修改方法中进行数据验证。

现在我们编写一个小的驱动程序对 Point 类中的修改方法进行测试。驱动程序生成的输出如下所示。图 6.7 中给出了程序跟踪和内存快照。

```
/*-------------------------------------------------------*/
/* Driver program to test Point class                    */
/* Filename main.cpp                                     */
#include "Point.h"
#include <iostream>
using namespace std;

int main()
{
  Point p1;
  p1.setX(2.3);
  p1.setY(1.5);
  cout << " p1.x is " << p1.getX() << endl;
  cout << " p1.y is " << p1.getY() << endl;
  return 0;
}
```

```
Ingbers-MacBook-Pro:Programs jaingber$ g++ main.cpp Point.cpp
Ingbers-MacBook-Pro:Programs jaingber$ ./a.out
Constructing Point object, default:
```

```
initializing to zero
p1.x is 2.3
p1.y is 1.5
```



内存快照：main()

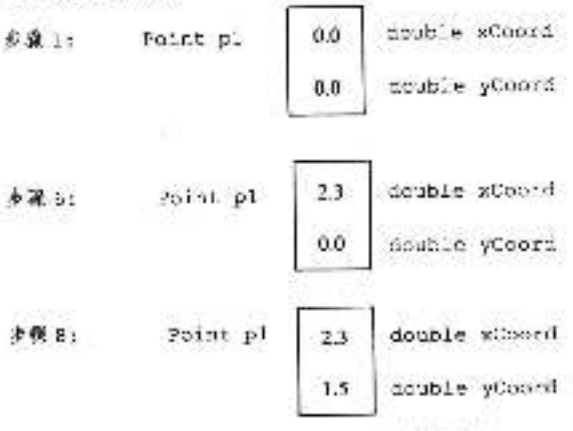


图 6.7 驱动程序的程序跟踪和内存快照

练习

下面的问题与本节设计的 Point 类有关。

考虑下面所给出的程序：

```
#include "Point.h"
#include <iostream>
using namespace std;
int main()
{
  Point p1, p2; //line 1
```

```
    p1.xCoord = 5; //line 2
    cout <<p1.getY(); //line 3
    cout << p1 << p2 << endl; //line 4
    return 0;
}
```

1. 这个程序可以通过编译吗？
2. 行1是合法的语句吗？解释原因。
3. 行2是合法的语句吗？解释原因。
4. 行3是合法的语句吗？解释原因。
5. 行4是合法的语句吗？解释原因。
6. 为Point类增加一个修改方法，用于为应用程序提供修改x和y坐标的接口。原型为：

```
void setXY(double xVal, double yVal);
```

6.8 解决应用问题：复合材料设计

高级复合材料的设计是在许多科学和工程领域中都很重要的问题。其中一个应用的例子就是封装材料，封装材料在工程中用于电子组装中关键部件的隔热与防震。对于封装材料而言，必须足够坚韧，以确保电子组装能抵抗外部的渗透和来自内部的震动。在本次仿真中，我们对一种溶融材料进行建模，该材料中包含了悬浮在两个圆柱空间之间的氧化铝颗粒。刚开始时，氧化铝颗粒是均匀分布的。当内侧的圆柱开始旋转时，氧化铝颗粒将远离内侧的圆柱而向外侧圆柱移动。在氧化铝颗粒密度高的地方韧性更好，在密度低的地方震动阻尼（vibrational dampening）更好。对于这次练习，我们更关注于确定位于关键半径之外的颗粒比例，如图6.8所示。在仿真过程中，我们将创建一个名为compositeMaterialsSim1.dat的文件作为输入文件。

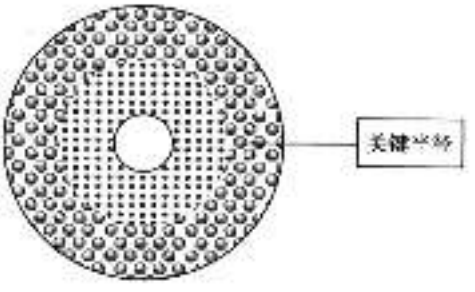


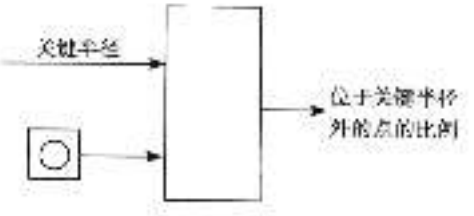
图6.8 悬浮在两个圆柱间的颗粒

1. 问题描述

数据文件compositeMaterialsSim1.dat中包含了在一次仿真中所记录的颗粒的x和y坐标。写一个C++程序，确定位于用户指定的关键半径之外的颗粒比例。

2. 输入/输出描述

程序的输入是名为compositeMaterialsSim1.dat的数据文件和关键半径。数据文件的第一行保存了外侧圆柱的半径，其后则是内层圆柱的半径。剩下的每行记录了一个颗粒的x和y坐标。关键半径是可变的，它由用户从键盘输入。

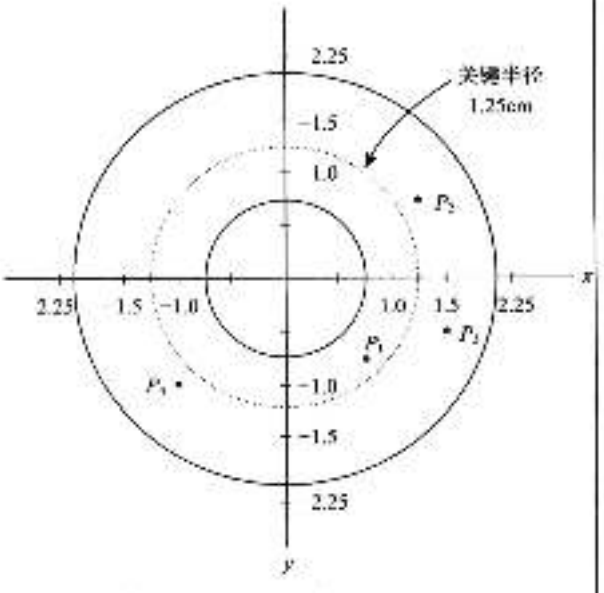


3. 用例

我们将使用一个只有4个顶点的用例，如右下图所示：

用例使用的数据文件内容如下：

```
 2.0     0.75
 0.75   -0.75
 1.25    0.75
 1.5    -0.5
-1.0    -1.0
```



我们使用的关键半径为1.25。

从图6.8中可以看到，两个圆柱的中心都在原点。使用距离公式可以计算出从原点到数据文件中每个点的距离，据此我们得到下面的结果：

从p1到原点的距离：1.060 66

从p2到原点的距离：1.457 74

从p3到原点的距离：1.581 14

从p4到原点的距离：1.414 21

4个点中有三个位于关键半径之外。因此在关键半径之外的点的比例为75%（3/4*100）。

4. 算法设计

分解提纲

1）初始化用于计算比例的计数器；

2）用户输入关键半径，从输入文件读取数据；

3）计算从原点到每个点的距离；

4）比较距离，并修改计数器；

5）打印位于关键半径之外的点的比例。

步骤2需要读取数据。关键半径只读取一次，圆柱半径从输入文件的第一行读取。为了从输入文件读取剩下点的数据，我们将使用数据终止循环来完成，因为在文件中并没有指定点的数目。步骤3和4将在循环内完成。我们将编写一个具有返回值的函数来完成步骤3。在所有的数据都从输入文件读出后，步骤5将在循环外完成。

```
Refinement in Pseudocode
      main:  Initialize counters for percentage
             Read critical radius
             Read radius of outer cylinder and inner cylinder
             Read x and y
             while not end of file
                increment point counter
                calculate distance from origin
                if distance is greater than origin
                      increment outside counter
             Print percentage of points outside critical radius
```

将代码中的步骤已经足够详细，可以转换成C++语句。对于这个应用，我们将提供

两种解决方案。第一种方案使用自定义的 Point 类，第二种使用 C++ 内建的数据类型。

解决方案 1：

```
/*-------------------------------------------------------------*/
/*  Program chapter6_8                                         */
/*                                                             */
/*  This program calculates the percentage points that lie     */
/*  outside of a critical radius.                              */

#include <iostream>  //Required for cin, cout
#include <fstream>   //Required for file input
#include "Point.h"   //Programmer-defined data type
using namespace std;

int main()
{
  //Declare objects
  const Point ORIGIN(0.0);
  Point p;
  int pointCount(0), outside(0);
  double x,y,criticalRad;
  double dist, radiusOuter, radiusInner;

  //open input file
  ifstream fin("compositeMaterialsSim1.dat");
  if(fin.fail())
  {
    cout << "Could not open data file compositeMaterialsSim1.dat"
         << endl;
    exit(1);
  }

  //Input critical radius from user
  cout << "Enter critical radius ";
  cin  >> criticalRad;

  //Input radius of outer and inner cylinders
  fin  >> radiusOuter >> radiusInner;

  //While not end of file, input point data
  fin >> x >> y;
  while(!fin.eof())
  {
    ++pointCount;  //increment point count
    p.setX(x);
    p.setY(y);
    dist = p - ORIGIN;
    if(dist > criticalRad)
    {
       ++outside;  //increment outside counter
    }
    fin >> x >> y;
  }

  //Print results
  //Pre-Multiply by 100.0 to force floating point arithmetic
  cout << (100.0*outside/pointCount)
       << "% lie outside the critical radius" << endl;
  return 0;
}
```

解决方案 2:

```
/*-------------------------------------------------------------*/
/*  Program chapter6_9                                         */
/*                                                             */
/*  This program calculates the percentage points that lie     */
/*  outside of a critical radius.                              */

#include <iostream>  //Required for cin, cout
#include <fstream>   //Required for file input
using namespace std;

//Function Prototypes
double distance(double x1, double y1, double x2, double y2);

int main()
{
  //Declare objects
  int pointCount(0), outside(0);
  double x,y,criticalRad;
  const double xORIGIN(0), yORIGIN(0);
  double dist, radiusOuter, radiusInner;

  //open input file
  ifstream fin("compositeMaterialsSim1.dat");
  if(fin.fail())
  {
    cout << "Could not open data file compositeMaterialsSim1.dat"
         << endl;
    exit(1);
  }

  //Input critical radius from user
  cout << "Enter critical radius ";
  cin >> criticalRad;

  //Input radius of outer and inner cylinders
  fin >> radiusOuter >> radiusInner;

  //While not end of file, input point data
  fin >> x >> y;
  while(!fin.eof())
  {
    ++pointCount;  //increment point count
    dist = distance(x,y,xORIGIN, yORIGIN);
    if(dist > criticalRad)
    {
       ++outside;  //increment outside counter
    }
    fin >> x >> y;
  }
  //Print results
  //Pre-Multiply by 100.0 to force floating point arithmetic
  cout << (100.0*outside/pointCount)
       << "% lie outside the critical radius" << endl;
  return 0;
}
#include <cmath>  //Required for sqrt and pow
double distance(double x1, double y1, double x2, double y2)
```

```
{
  double d1, d2, d;
  d1 = x2-x1;
  d2 = y2-y1;
  d = sqrt( pow(d1,2) + pow(d2,2) );
  return d;
}
```

5. 测试

使用用例中的输入文件，方案1的输出如下：

```
Ingbers-MacBook-Pro:Programs jaingber$ g++ chapter6_8.cpp
    Point.cpp
Ingbers-MacBook-Pro:Programs jaingber$ ./a.out
Constructing Point object, parameterized:
input parameters: 0.0
Constructing Point object, default:
initializing to zero
Enter critical radius 1.25
75% lie outside the critical radius
```

使用相同的输入文件，方案2的输出如下：

```
Ingbers-MacBook-Pro:Programs jaingber$ g++ chapter6_9.cpp
Ingbers-MacBook-Pro:Programs jaingber$ ./a.out
Enter critical radius 1.25
75% lie outside the critical radius
```

修改

1. 编写一个带返回值的函数，其函数原型如下：

```
bool validate(double rad1, double rad2, double critic
alRadius);
```

如果 rad1 < critical radius（关键半径）< rad2，则函数应返回 true，否则返回 false。在程序 chapter6_8 中使用这个函数验证用户所输入的关键半径是否位于从数据文件读取的外侧半径和内侧半径之间。允许用户有二次输入关键半径的机会。如果三次输入都非法，程序应当终止并在退出前显示相关消息。

2. 为 Point 类添加一个新方法。新方法的原型如下：

```
void Point::input(fstream& fin);
```

语句 "p.input (fin);" 将从文件输入流 fin 中输入两个浮点值，并将它们赋给调用对象的 x 和 y 坐标。在程序 chapter6_8 中使用该方法替换两条 fin >> x >> y 语句。注意，当你使用 p.input(fin) 时，不需要调用任何 set 方法。

*6.9 数值方法：多项式的根

多项式是一个关于对象的函数，它可以表示成下面的一般形式：

$$f(x) = a_0x^N + a_1x^{N-1} + a_2x^{N-2} + \cdots + a_{N-2}x^2 + a_{N-1}x + a_N \qquad (6.1)$$

其中对象是 x，系数由 a_0，a_1，…，a_N 表示。一个多项式的次数等于其中最大的非零指数。因此，一个二次多项式的一般形式为

$$g(x) = a_0x^3+a_1x^2-a_2x+a_3$$

一个具体的三次多项式的例子为

$$h(x) = x^3-2x^2+0.5x-6.5$$

注意，对于等式中的每项而言，系数下标与对象指数的和等于使用公式（6.1）中标记的多项式次数。

6.9.1 多项式的根

许多工程问题解决方案中都涉及求解形如下面方程的根

$$y = f(x)$$

这里的根是当 y 等于 0 时 x 的值。需要求解方程的根的应用实例包括机械手控制系统的设计，汽车悬架和减震器的设计，电机响应分析，以及数字滤波器稳定性的分析等。

如果函数 $f(x)$ 是一个 N 次多项式，那么 $f(x)$ 有 N 个根。这 N 个根可能包含实根或复根，后面的例子将说明这一点。如果我们限定多项式的系数（a_0、a_1、…、a_N）都是实数，那么复根将总是以共轭对的形式出现。（回忆一下，一个负数可以被表示为 $\alpha+i\beta$，其中 $i=\sqrt{-1}$。那么 $\alpha+i\beta$ 的复数共轭为 $\alpha-i\beta$。）

如果一个多项式可以分解成线性项，那么通过令每一项为 0，可以很容易地看出多项式的根。考虑下面的等式：

$$f(x) = x^2+x-6 = (x-2)(x+3)$$

如果 $f(x)$ 等于 0，可以得到

$$(x-2)(x+3) = 0$$

当 $f(x)$ 等于 0 时，方程的根或者 x 的值为 $x = 2$ 和 $x = -3$。这些根也对应于多项式与 x 轴的交点，如图 6.9 所示。

如果一个二次方程（多项式的次数为 2）不容易分解，我们可以使用二次公式来确定方程的根。二次方程的一般形式为

$$y = ax^2-bx+c$$

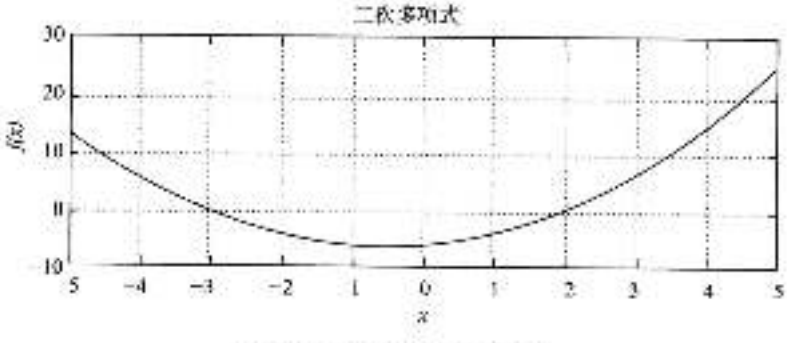


图 6.9 有两个实根的多项式

那么其根可以按照下面公式计算得到

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

和

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

因此，如果二次方程为

$$f(x)=x^2+3x+3$$

那么其根为

$$x_1=\frac{-3+\sqrt{-3}}{2}=-1.5+0.87\sqrt{-1}$$

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为3，所以有3个根。如果我们假定系数都是实数，那么只有下面4种可能性：

- 三个不同的实根（相异根）
- 三个相同的实根（重根）
- 一个相异根和两个重根
- 一个实根和一对共轭复根

下面的函数示例说明了其中的每种情况：

$$f_1(x)=(x-3)(x+1)(x-1)=x^3-3x^2-x+3$$
$$f_2(x)=(x-2)^3=x^3-6x^2+12x-8$$
$$f_3(x)=(x+4)(x-2)^2=x^3-12x^2-16$$
$$f_4(x)=(x+2)[x-(2+i)][x-(2-i)]=x^3-2x^2-3x+10$$

图6.10中画出了每个函数的图像。再一次看到这里的实根对应于函数与x轴的交点。

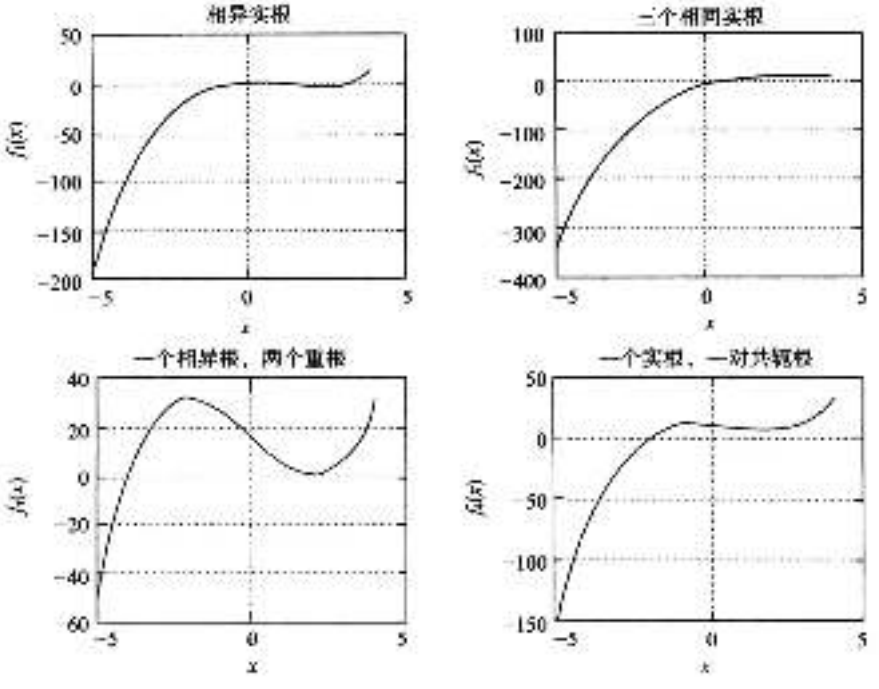


图6.10 三次多项式

确定一个一次或二次多项式的根相对简单，但确定三次及以上次数的多项式的根比较困难。有许多数值方法用于确定多项式的根。这些方法，如增量搜索、二分法、试位法等，通过搜索函数符号改变的区间来找到实根，因为函数符号的改变意味着函数与 x 轴相交了。也有其他的方法用于寻找复根，如牛顿－拉普森方法。

6.9.2 增量搜索方法

增量搜索（increment-search）方法通常用于确定函数在给定区间 [a, b] 内的实根。该方法对符合函数值一端为负、另一端为正这一条件的子区间 $[a_k, b_k]$ 进行搜索。在这样的子区间中，我们可以确信其中至少有一个根。

增量搜索方法有许多不同的变化形式。我们讨论的这种方法在初始时选择一个步长，用来将原始区间划分成一组子区间，如图 6.11 所示。对于每个子区间，我们计算两端的函数值。如果两端的函数乘积为负，那么在该子区间内存在一个根。（函数值的乘积为负表明其中一个函数值为负，另一个为正。因此，函数在这个区间必定与 x 轴相交）。这时，我们可以估计根在该段区间的中点上，如图 6.12a 所示。也有可能子区间的一个端点就是根或者离根很近，如图 6.12b 所示。记住，一个浮点值可以很接近 0，所以测试某个端点是否是根，应当将对应的函数值与一个很小的数进行比较，而不是与 0 进行比较。

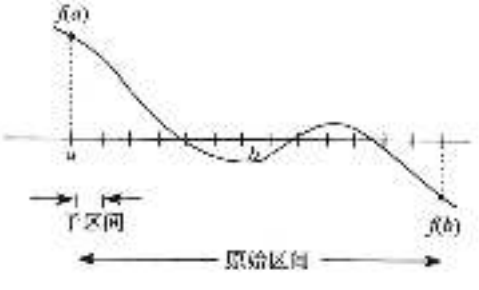


图 6.11 增量搜索

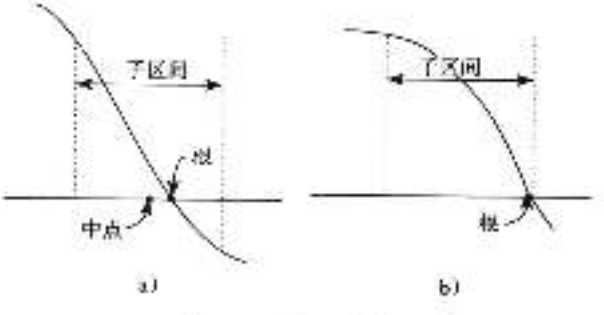


图 6.12 子区间分析

需要认识到的重要一点就是，增量搜索方法也有失败的情况。例如，假定在一个子区间内有两个根。在这种情况下，由于两端点的函数值可能符号相同，它们的乘积为正，算法就会跳过这个区间进入下一个区间。另一个例子是在一个子区间内有三个根。在这种情况下，两端点的函数值符号不同，估计函数的根在区间的中点。然后我们继续搜索下一个子区间，因此漏掉了前一个区间上的两个根。这些例子用于说明增量搜索方法是有一些缺陷的。但是一般而言，它可以工作得很好。如果需要具有更好性能特征的其他根搜索方法，可以参考 [16]。

*6.10 解决应用问题：系统稳定性

术语“系统”（system）通常用来代表具有特定输入、输出或特定行为的装置或设备，比如与管线连接的冷却装置，用于制造设备的机械手和高速子弹头列车等。稳定的系统（stable system）的简单定义如下：如果一个合理的输入会导致合理的输出，那么系统就是稳定的。例如，考虑机械手的控制系统。系统的合法输入需要指出机械手移动的一个合理方向。如果一个合法的输入导致机械手不稳定或者向一个不合理的方向移动，那么系统是不稳定的。系统设计的稳定性分析关系到系统的动态属性。关于分析类别或函数类别的讨论超出

了本书的范围，但是分析中的一部分需要确定多项式的根。通常，分析中对于实根和复根都需要，但是求复根的方法涉及多项式的导数，相关的方法数学性更强。因此，我们缩小了问题的范围，只求解给定搜索区间上多项式的实根。同时，我们还假定多项式为三次多项式，但是开发的解决方案可以很容易地扩展到处理更高次的多项式。

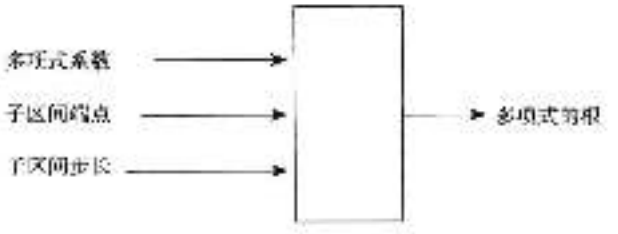
开发一个程序以确定一个三次多项式的实根。允许用户输入多项式的系数、要搜索的区间，以及在搜索中使用的子区间的步长。

1. 问题描述

确定一个三次多项式的实根。

2. 输入/输出描述

I/O 图表明输入信息包括了多项式系数、区间端点和子区间的步长。输出为指定区间上的根。



3. 用例

在用例中，我们使用等式

$$y=2x-4$$

该函数可以被描述为一个三次多项式，其系数依次为 $a_0=0$，$a_1=0$，$a_2=2$，$a_3=-4$。如果令多项式等于 0，可以很容易地看出根为 2。为了验证暴露搜索方法，我们首先采用一个使根落在子区间端点上的步长，然后采用不使根落在子区间其中一个端点上的步长。如果根落在一个端点上，我们可以很容易地找到它，因为此时多项式的值将十分接近于 0。如果根落在子区间之内，那么端点上函数值的乘积将为负，我们就可以估计根在区间的中点上。

首先，考虑步长为 0.5 时的区间 [1, 3]。产生的子区间和对应的信息如下所示：

子区间 1：[1.0, 1.5]

$f(1.0)*f(1.5)=(-2)*(-1)=2$

在该区间上没有根。

子区间 2：[1.5, 2.0]

当我们计算两个端点值时，可以算出根 $x=2.0$。

子区间 3：[2.0, 2.5]

当计算端点值时，我们再次算出根 $x=2.0$。注意，我们在计算时需要小心，不要在程序中将同一个根识别出两次。

子区间 4：[2.5, 3.0]

$f(2.5)*f(3.0)=(1)*(2)=2$

在该区间上没有根。

现在，考虑步长为 0.3 时的情况，产生的子区间和对应的信息如下：

子区间 1：[1.0, 1.3]

$f(1.0)*f(1.3)=(-2)*(-1.4)=2.8$

在该区间上没有根。

子区间 2：[1.3, 1.6]

$f(1.3)*f(1.6)=(-1.4)*(-0.8)=1.12$

在该区间上没有根。

子区间 3：[1.6, 1.9]

$f(1.6)*f(1.9)=(-0.8)*(-0.2)=1.6$

在该区间上没有根。

子区间 4：[1.9, 2.2]

$f(1.9)*f(2.2)=(-0.2)*(0.4)=-0.08$

在该区间上的根被估计在子中点上。

子区间 5：[2.2, 2.5]

$f(2.2)*f(2.5)=(0.4)*(1.0)=0.4$

在该区间上没有根。

子区间 6：[2.5, 2.8]

$f(2.5)*f(2.8)=(1.0)*(1.6)=1.6$

在该区间上没有根。

子区间 7：[2.8, 3.1]

注意，右端点超过了区间的范围。在程序中，我们将进行修正以保证区间在原有的右端点上结束。

$f(2.8)*f(3.0)=(1.6)*(2.0)=3.2$

在该区间上没有根。

4. 算法设计

首先设计分解提纲，它将解决方案分解成一系列的顺序步骤。

分解提纲

1）读取多项式系数，区间范围，步长；

2）使用子区间确定根。

步骤 1 需要提示用户输入必需的信息，并读取信息。步骤 2 需要一个循环来计算子区间端点并确定是否有根落在端点上或在子区间内。当根被确定后，打印出相应的消息。在步骤 2 中有大量的操作，所以我们需要考虑使用函数，以避免主函数过长。因为我们需要在程序中的几个地方计算三次多项式，所以可以考虑编写一个函数来完成这项工作。在每个子区间内，我们需要对根进行搜索；这个搜索过程同样可以考虑用一个函数完成。解决方案的结构图在图 6.1 中给出。细化后的伪代码如下所示：

```
Refinement in Pseudocode
main:  read coefficients, interval endpoints a and b,
          and step size
       compute the number of subintervals, n
       set x to 0
```

```
        while k<= n-1
             compute left subinterval endpoint
             compute right subinterval endpoint
             check roots (left, right, coefficients)
             increment k by 1
        check_roots (b,b,coefficients)

check_roots (left, right, coefficients):
        set f_left to poly(left,coefficients)
        set f_right to poly(right,coefficients)
        if f_left is near zero
             print root at left endpoint
        else
             if f_left * f_right < 0
                  print root at midpoint of subinterval
        return
poly(x,a0,a1,a2,a3):
        return a3x^3 + a2x^2 + a1x + a0
```

注意，伪代码中的 check_root() 函数用于检查左区间端点是否为根，但是没有检查右端点。这是为了避免将同一个根标识两次：一个区间的右端点同时也是下一个子区间的左端点。因为我们只检查左端点，所以我们需要检查区间的最后一个右端点，因为它不是任何子区间的左端点。

伪代码中的步骤已经足够详细，可以转换成 C++ 语句：

```
/*-------------------------------------------------------*/
/*  Program chapter6_10                                  */
/*                                                       */
/*  This program estimates the real roots of a           */
/*  polynomial function using incremental search.        */

#include<iostream>  //Required for cin, cout
#include<cmath>  //Required for call()
using namespace std;

// Function Prototypes
void check_roots(double left, double right, double a0,
                 double a1, double a2, double a3);
double poly(double x, double a0, double a1,
            double a2, double a3);

int main()
{
   //  Declare objects and function prototypes.
   int n;
   double a0, a1, a2, a3, a, b, step, left, right;

   //  Get user input.
   cout << "Enter coefficients a0, a1, a2, a3: \n";
   cin >> a0 >> a1 >> a2 >> a3;
   cout << "Enter interval limits a, b (a<b): \n";
   cin >> a >> b;
   cout << "Enter step size: \n";
   cin >> step;

   //Check subintervals for roots.
   n = call((b - a)/step);
   for (int k=0; k< n-1; k++)

     left = a + k*step;
```

```
      if (k == n-1)
      {
         right = b;
      }
      else
      {
         right = left + step;
      }
      check_roots(left,right,a0,a1,a2,a3);
   }
   check_roots(b,b,a0,a1,a2,a3);

   //  Exit program.
   return 0;
}
/*-------------------------------------------------------------*/
/*  This function checks a subinterval for a root.             */

void check_roots(double left, double right, double a0,
                 double a1, double a2, double a3)
{
   //  Declare objects and function prototypes.
   double f_left, f_right;

   //  Evaluate subinterval endpoints and
   //  test for roots.
   f_left = poly(left,a0,a1,a2,a3);
   f_right = poly(right,a0,a1,a2,a3);
   if (fabs(f_left) < 0.1e-04)
   {
      cout << "Root detected at " << left << endl;
   }
   else
   {
      if (fabs(f_right) < 0.1e-04)
         ;
      else
      {
         if (f_left*f_right < 0)
         {
            cout << "Root detected at " << (left+right)/2 << endl;
         }
      }
   }
   // Exit function.
   return;
}
/*-------------------------------------------------------------*/
/*  This function evaluates a cubic polynomial.                */

double poly(double x, double a0, double a1, double a2,
            double a3)
{
   return a0*x*x*x + a1*x*x + a2*x + a3;
}
/*-------------------------------------------------------------*/
```

5. 测试

如果我们使用用例中的数据，将得到下面的交互输出（计算出的根与手工计算的结果

```
函数):
   Enter coefficients a3, a2, a1, a0:
   0 3 2 -4
   Enter interval limits a, b (a<b):
   1 3
   Enter step size:
   0.5
   Root detected at: 2.000
   Enter coefficients a3, a2, a1, a0:
   0 3 2 -4
   Enter interval limits a, b (a<b):
   1 3
   Enter step size:
   0.3
   Root detected at 2.050
```

使用二次多项式 $f(x)=x^2+3x-3$ 测试该程序。所选的区间和步长不要让根总是落在子区间的端点上。

修改

1. 如果根不在某个子区间的端点上，那么区间的大小会影响根的估计。使用用例中的多项式，分别使用以下步长运行实验：1.1，0.75，0.5，0.3，0.14，区间范围为 [0.5, 3]。
2. 使用三次多项式 $h(x)=x^3-2x^2+0.5x-6.5$，对程序进行测试，并使根落在输入区间 [a, b] 的端点上。
3. 使用三次多项式 $h(x)=x^3-2x^2+0.5x-6.5$，找到一个步长，使程序在初始区间 [-10, 10] 上丢掉某些根。说明程序为什么会丢掉这些根。是程序有错误吗？
4. 修改程序，使其检查子区间的右端点是否是根，而不再检查左端点。确保程序中包含对区间 [a, b] 第一个端点的检查。
5. 修改程序，让它可以找到一个四次多项式的实根。
6. 使用这个程序解决 6.6 节中的第 4 个问题。

牛顿－拉普森方法

牛顿－拉普森方法（Newton-Raphson method）是一种依赖于等式信息的常用求根方法。由于它使用的是根估计方式，因此它也是一种有名的切线法。牛顿－拉普森方法同时使用函数 $f(x)$ 和它的导数 $f'(x)$ 来计算切线的截距，其中切线是曲线的每个点上最接近曲线本身的直线。切线和 x 轴的交点是根的下一个估计值，同时也是计算切线的下一个点。重复这一过程，直到找到根的位置。因为牛顿－拉普森方法在每一步对根都有更好的估计值，所以它收敛所需要的步骤明显少于增量搜索方法。

使用图 6.13 中的图像，我们发现牛顿－拉普森方法可以用下面的一系列步骤来描述。初始估计值 x_1 是一个适合于函数 $f(x)$ 的根。然后使用 $f'(x_1)$ 来计算曲线在 x_1 点上的截距，然后使用这个截距画出通过 $f(x_1)$ 的切线。点 x_2 处的切线与 x 轴的交点形成了根的一个新的估计值。然后重复前面的过程，使用 $f'(x_2)$ 作为截距，画一条通过 $f(x_2)$ 的切线，这条切线与 x 轴的交点为 x_3。如此重复，直到收敛到根（记为 x_r）为止。

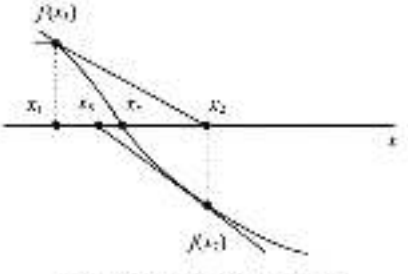


图 6.13 牛顿－拉普森方法

给定当前根的估计值 x_i，算法通过 $f(x_i)$ 处的切线可以计算得到下一个估计值 x_{i+1}，计算公式如下，其中 $f(x_i)$ 处的切线使用 y 轴上的变量除以 x 轴上的变化量得到：

$$f(x_i) = \frac{f(x_i)-0}{x_i - x_{i+1}}$$

解出这个等式中的 x_{i+1}，即计算除了根的下一个估计值：

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

如同前面所说的，牛顿－拉普森方法通常需要的迭代次数更少，因此计算效率比增量搜索方法更高。此外，当初始估计值为复数或者等式中有复数时，它也可以用于找出复根。这种方法还可以推广到多维情况，用于解决非线性等式问题。

这种方法也有一些限制。最重要的限制与初始估计值有关：如果初始估计值不够好，那么算法可能完全漏掉根，找到一个根或者一个根都找不到，取决于等式本身。该方法还可能出现其他问题。在曲线顶点或拐点处令 $f'(x)$ 等于 0 或者接近于 0，可能出现函数值和导数值都为 0 的情况，那么方法在一开始就失败了。但这些问题通常都可以通过一个与根足够接近的初始估计值得以避免。而计算函数的导数有时并不容易甚至无法计算。在这种情况下可以使用那些不需要导数的方法，如割线法。

为了说明牛顿－拉普森方法，我们将写一个程序来确定多项式 $p(x)$ 的实根，其中 $p(x)$ 的形式如下：

$$y = p(x) = a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4$$

当 $p(x)$ 小于 0.001 时，我们就认为已经找到了根。作为一个测试用例，假设多项式等式为

$$y = p(x) = x^2 - 4x - 3$$

通过因式分解，我们可以确定等式的根为 $x = -1$ 和 $x = -3$。牛顿－拉普森方法还需要计算多项式的导数，其形式如下：

$$p'(x) = 2x + 4$$

我们从一个初始估计值开始，并计算函数值和它的导数值。误差测量值为函数值的绝对值。当误差小于 0.001 时，过程终止；否则使用前面给出的关于 x_{i+1} 的等式对根进行下一次估计。

现在我们给出一个实现了牛顿－拉普森方法的程序。因为程序较短且直截了当，所以不需要模块：

```
/*------------------------------------------------------------------*/
/*  Program Chapter6 (1                                             */
/*                                                                  */
/*  This program finds the real roots of a cubic polynomial         */
/*  using the Newton-Raphson method.                                */
/*------------------------------------------------------------------*/

#include<iostream> //Required for cout, cin
#include<cmath>  //Required for pow()
using namespace std;

int main()
{
//  Declare objects
      int iterations(0);
```

```
        double a0, a1, a2, a3, x, p, dp, tol;

//  Get user input.
        cout << "Enter coefficients a0, a1, a2, a3\n";
        cin >> a0 >> a1 >> a2 >> a3;
        cout << "Enter initial guess for root\n";
        cin >> x;
// Evaluate p at initial guess.
        p = a0*pow(x,3) + a1*x*x + a2*x + a3;

//  Determine tolerance.
        tol = fabs(p);
        while(tol > 0.001 && iterations < 100)
        {
                //  Calculate the derivative.
                dp = 3*a0*x*x + 2*a1*x + a2;

                //  Calculate next estimated root.
                x = x - p/dp;
                //  Evaluate p at estimated root.
                p = a0*x*x*x + a1*x*x + a2*x + a3;
                tol = fabs(p);
                iterations++;
        }
        if(tol < 0.001)
        {
                cout << "Root is " << x << endl
                     << iterations << " iterations\n";
        }
        else
                cout << "Did not converge after 100 iterations\n";
        return 0;
}
```

程序的几次示例运行输出如下：

```
Enter coefficients a0, a1, a2, a3
0 1 4 3
Enter initial guess for root
0
Root is -0.999695
 3 iterations
-----------------------------------------------------------------------
Enter coefficients a0, a1, a2, a3
0 1 4 3
Enter initial guess for root
5
Root is -0.999799
5 iterations
-----------------------------------------------------------------------
Enter coefficients a0, a1, a2, a3
0 1 4 3
Enter initial guess for root
 4
Root is -3.002303
 3 iterations
```

修改

1. 运行牛顿－拉普森程序，使用相同的系数和初始估计值 −1。
2. 运行牛顿－拉普森程序，使用相同的系数和初始估计值 −2，说明发生了什么。

3. 修改牛顿－拉普森程序，允许用户输入停止迭代的误差值。
4. 修改牛顿－拉普森程序，找出一个五次多项式的根。

*6.11 数值方法：积分

积分类作为工程师和科学家提供了有关于功能或数据集的重要信息。例如，距离、速率和加速度都能通过积分与相关联起来。速率是加速度的积分，即离是速率的积分。在微积分课程中对积分的主题有详细的讨论，但一些基本的理论可以用面积的概念来进行简单的解释。在一个区间上的函数的积分就是在函数图像下方覆盖的面积。积分可以使用几种不同方法中的任何一种来进行数值上的近似。本章我们将讨论使用梯形法则（trapezoidal rule）来计算数值积分的方法。

使用梯形法则进行积分

为了使用微积分中的解析方法得到某一区间上的函数积分，我们必须知道函数的表达式。在许多工程和科学应用中，我们知道函数中的数据点或测量值，但并不了解函数的表达式。因此，我们需要一种只依靠函数上的点就可以计算函数积分的方法。在其他应用中，我们可能知道函数的表达式，但是使用解析方法来确定积分比较困难甚至不可能。在这种情况下，我们希望有一种方法允许我们计算函数上的点或函数值，然后采用数值方法计算积分。本节中我们给出的方法是一种简单的方式，它通过曲线上给定的点估计曲线下覆盖的面积。方法中使用的是梯形区域的面积，因此被称作使用梯形法则的积分。

函数 $f(x)$ 在 $a \sim b$ 上的积分表示如下：

$$\int_a^b f(x)\mathrm{d}x$$

它代表了函数 $f(x)$ 在 $x = a \sim x = b$ 中函数曲线下的面积，如图 6.14 所示。

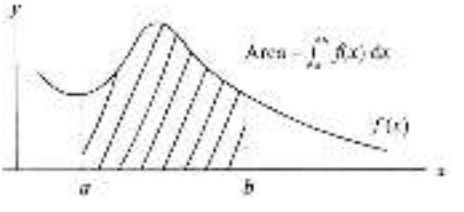


图 6.14 曲线下的面积

如果我们给定了代表曲线的函数，则可以计算出在间隔一定区间的点上的函数值，如图 6.15 所示。注意，因为 $y = f(x)$，我们可以将 $f(x_1)$ 表示为 y_1，$f(x_2)$ 表示为 y_2，等等。

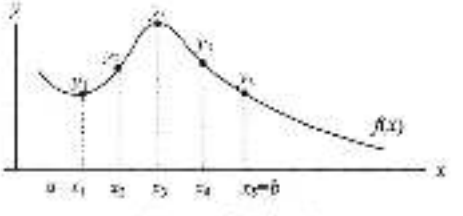


图 6.15 间隔区间

如果将曲线上的点使用直线连接，我们将得到一组梯形，它们的组合近似于曲线下方区域的面积。曲线上的点离得越近，在区间上的梯形就越多，其比在近似积分时就越准确。在图 6.16 中，我们使用曲线上的 5 个点形成了 4 个梯形，这 4 个梯形的面积之和就近似于函数在 $a \sim b$ 上的积分。

梯形的面积是两边之和与底的乘积的 1/2：

$$\text{Area} = 1/2 \times \text{base} * (\text{height}_1 + \text{height}_2)$$

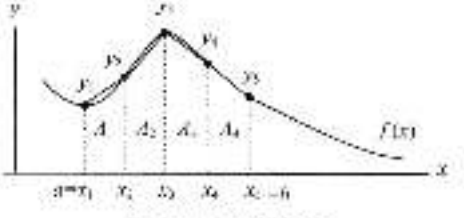
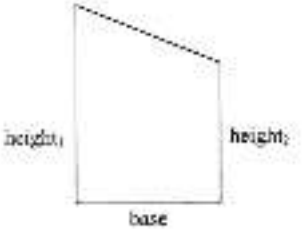


图 6.16 四个梯形



因此，第一个梯形的面积 A_1 使用点 (x_1, y_1) 和 (x_2, y_2) 计算：

$$A_1 = 1/2 * (x_2 - x_1) * (y_1 - y_2)$$

因为本例中曲线上的点在 x 轴上的间隔都相等，所以每个梯形的底都相等。我们可以分别计算出 4 个梯形的面积：

$$A_1 = 1/2 * \text{base} * (y_1 - y_2)$$

$$A_2 = 1/2 * \text{base} * (y_2 - y_3)$$

$$A_3 = 1/2 * \text{base} * (y_3 + y_4)$$

$$A_4 = 1/2 * \text{base} * (y_4 + y_5)$$

因此，在 $a \sim b$ 之间的总面积近似等于 4 个梯形面积：

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{\text{base}}{2}((y_1 + y_2) + (y_2 + y_3) + (y_3 + y_4) + (y_4 + y_5))$$

$$\approx \frac{\text{base}}{2}(y_1 + 2y_2 + 2y_3 + 2y_4 + y_5)$$

一般而言，如果曲线下的区域被划分为 N 个底相等的梯形，那么该区域面积可以用下面的表达式近似：

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{\text{base}}{2}\left(y_1 + 2\sum_{k=2}^{N} y_k + y_{N+1}\right)$$

这个表达式引用了梯形法则。

当使用数值方法计算积分时，我们需要记住曲线上的点可以来自不同的源。如果我们知道曲线的表达式，则可以使用 C++ 程序计算出数据点，利用数据点得到梯形的高；在这种情况下，我们可以按照自己的要求让数据点靠近或者隔开。另一种可能是，数据点是实验收集得到的数据；在这种情况下我们有一组代表梯形底的 x 坐标和一组代表梯形高的 y 坐标。我们仍然可以使用梯形面积来估算积分，但是我们不能选择让数据点靠近或隔开，因为我们没有函数表达式用来计算，而只有已知的数据点可以使用。如果由数据点确定的梯形的底各不相等，那么必须分别将各个梯形的面积相加，而不能使用假定梯形底相等时的公式计算。

现在我们给出一个确定下面表达式积分（在两个给定点之间）估计值的 C++ 程序：

$$y = f(x) = 4e^{-x}$$

```
/*-----------------------------------------------------------------*/
/*  Program chapter6_12                                            */
/*                                                                 */
/*  This program estimates the area under a given curve            */
/*  using trapezoids with equal bases.                             */
```

```
#include<iostream> //Required for cin, cout
#include<cmath> //Required for exp()
using namespace std;

// Function prototypes.
double integrate(double a, double b, int n);
double f(double x);

int main()
{
   // Declare objects
   int num_trapezoids;
   double a, b, area;

   // Get input from user.
   cout << "Enter the interval endpoints, a and b\n";
   cin >> a >> b;
   cout << "Enter the number of trapezoids\n";
   cin >> num_trapezoids;

   // Estimate area under the curve of 4e^-x
   area = integrate(a, b, num_trapezoids);

   // Print result.
   cout << "Using " << num_trapezoids
        << " trapezoids, the estimated area is "
        << area << endl;

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

/*-------------------------------------------------------------*/
double integrate(double a, double b, int n)
{
   // Declare objects.
        double sum(0), x, base, area;

        base = (b-a)/n;
        for(int k=2; k<=n; k++)
        {
                x = a + base*(k-1);
                sum = sum + f(x);
        }
        area = 0.5*base*(f(a) + 2*sum + f(b));
        return area;
}
double f(double x)
{
        return(4*exp(-x));
}
/*-------------------------------------------------------------*/
```

下面是程序运行几次示例的输出：

```
Enter the interval endpoints, a and b
0 1
Enter the number of trapezoids
5
Using 5 trapezoids, the estimated area is 2.53640
-------------------------------------------------------------
Enter the interval endpoints, a and b
0 1
```

```
Enter the number of trapezoids
50
Using 50 trapezoids, the estimated area is 2.528567
Enter the interval endpoints, a and b
0 1
Enter the number of trapezoids
100
Using 100 trapezoids, the estimated area is 2.528303
```

如果我们使用微积分计算给定函数在区间 [0, 1] 上的积分，则得到具有 7 位精度的理论值为 2.528 482。

本章小结

大部分 C++ 程序都倾向于使用库和自定义函数。函数允许我们重用软件并在方案中使用抽象，因此减少了开发时间，同时提高了软件质量。为了说明使用带返回值的自定义函数以解决问题，我们设计了大量例子，其中包括递归函数的例子。为了说明随机数（整数或浮点数）的生成，我们给出了专门的例子，并实现了增量搜索方法和牛顿－拉普森方法用于求多项式的根，同时还介绍了使用梯形法则进行数值积分。

关键术语

abstraction（抽象）
accessor methods（访问方法）
address operator（地址操作符）
automatic storage class（自动存储类型）
center of gravity（重心）
composite materials（复合材料）
computer simulation（计算机仿真）
custom header file（自定义头文件）
encapsulation（封装）
external storage class（外部存储类型）
function argument（函数参数）
formal parameter（形参）
function（函数）
function body（函数体）
function header（函数头）
function prototype（函数原型）
global scope（全局作用域）
incremental search（增量搜索）
library function（库函数）
local scope（局部作用域）
modularity（模块化）
module（模块）
module chart（模块图）
moment（矩）
mutator methods（修改方法）
Newton-Raphson method（牛顿－拉普森方法）
numerical integration（数值积分）
parameter declarations（参数声明）
pass by reference（引用传递）
pass by value（值传递）
public interface（公共接口）
programmer-defined function（自定义函数）
radom number（随机数）
random number seed（随机数种子）
register class（寄存器类型）
reliability（可靠性）
re-usability（可重用性）
root（根）
scope（作用域）
simulation（仿真）
stable system（稳定的系统）
static storage class（静态存储类型）
storage class（存储类型）
structure chart（结构图）
system（系统）
trapezoidal rule（梯形法则）

C++ 语句总结

函数定义

```
return_type function_name(parameter types)
{
   declarations;
   statements;
}
```

方法定义

```
return_type class_name::function_name(parameter types)
{
   declarations;
   statements;
}
```

返回语句

```
void function;

return;
```

带返回值的函数

```
return (a + b/2);
```

函数原型

```
double sine(double x);
double sin(double);
void check_roots(double left, double right, double e0,
                 double e1, double e2, double e3);
```

注意事项

1. 一个由多个模块组成的程序比一个长的 main 函数更易读，更易理解。
2. 选择函数名时应当体现出函数的目标。
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分开。
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. 使用参数列表为函数传递信息，而不要使用全局对象。

调试要点

1. 如果难以理解编译器的错误消息，尝试使用另一个编译器编译程序来获得不同的错误消息。
2. 在调试一个长程序时，在某些代码段周围加上注释符号（/* 和 */），以使你能将注意力集中到程序的其他部分。
3. 使用一个驱动程序来单独测试一个复杂的函数。
4. 确保函数返回的值与它的返回类型匹配。如果有必要，使用类型转换操作符将返回值转换成合适的类型。
5. 函数可以在 main 函数之前或之后定义，但不能在其中定义。
6. 总是使用函数原型语句，以避免参数传递时的错误。
7. 在函数被调用之前使用 cout 语句生成函数参数的内存快照，同时在函数开头生成形参的内存快照。
8. 在实现函数参数和形参时，一定要注意类型、顺序和参数数目是否一致。

9. 与系统相关的限制有时可能会使递归解法中的多个问题变成一个问题。

习题

判断题

1. 函数体包含在大括号之内。
2. 参数列表包含了函数使用的所有对象。
3. 在一次通过值传递进行的调用中，函数不能改变函数参数的值。
4. 在一个函数内声明的静态对象，其值从一次调用到下一次调用时仍然保持。
5. 类的所有数据成员，或者属性，都必须有相同的数据类型。
6. 成员函数对调用对象的私有数据成员有访问权。
7. 修改方法可以改变调用对象的状态。

多选题

8. 下面（　　）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)

9. 在函数调用中，分隔函数参数的是（　　）。

(a) 逗号　　(b) 分号
(c) 冒号　　(d) 空格

10. 在标识符的定义语句中可以知道的是（　　）。

(a) 全局的　　(b) 局部的
(c) 静态的　　(d) 作用域

程序分析

在问题 11 ~ 14 中使用了下面的函数：

```
/*------------------------------------------- */
/* This function returns 0 or 1.              */
/*                                            */
int fives (int x)
{
//  Declare objects.
    int result;
//  Compute result to return.
    if (x%5) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
/*------------------------------------------- */
```

11. fives(15) 的值是多少？
12. fives(26) 的值是多少？
13. fives(ceil(sqrt(62.5))) 的值是多少？（提示：不需要用计算器计算这个值。）
14. 函数对于所有整数都可以正常工作吗？如果不是，它的限制有哪些？

内存快照问题

在问题 15 ~ 17 中使用了下面的类定义：

```
//Class Declaration
class UnitVector
{
private:
//data members
  double x, y; //vector anchor point
  double orientation; //vector orientation
public:
//constructor functions
  UnitVector(); //default constructor
  UnitVector(double x_val, // constructor with 3 parameters
             double y_val,
             double or);
};
//Class implementation
UnitVector::UnitVector()

   x = y = 0;
   orientation = 3.1415;
}

UnitVector::UnitVector(double x_val, double y_val, double o)
{
   x = x_val;
   y = y_val;
   orientation = o;
}
```

给出下面每组语句的两种决策。

15. `UnitVector v1, v2;`
16. `UnitVector v1(0,0,0), v2;`
17. `UnitVector v1(2.1, 3.0, 1.6), v2; v2 = v1;`

编程题

18. 写一个返回为空的函数，使用两个嵌套循环和取模操作符（%）以确定并在标准输出上打印出前 n 个素数。素数是一个只能被自身和 1 整除的数。（提示：函数原型为“primeGen(int n);”。）
19. 写一个返回为空的函数，使用两个嵌套循环和取模操作符（%）确定前 n 个素数，并将这些数输出到一个指定的输出文件中。素数是一个只能被自身和 1 整除的数。（提示：函数原型为“primeGen(int n, ostream& file);”，前提条件是 file 已经定义了。）
20. 写一个有返回值的函数，该函数打印出包含在一个数据文件中的合法整数的数目。如果文件中包含任何非整数的数据，则函数在遇到一个非法数据前应当先返回已经读取的整数数目，并输出消息到标准错误上，提示文件中的数据不是全部都可以被读取。（提示：使用函数头“int countInts(ifstream& file);”，前提条件是 file 已经定义了。）
21. 写一个返回为空的函数，打印出一个文件流的状态标志 badbit、failbit、eofbit 和 goodbit。文件流为输入参数。

简单的仿真。在下面的这些问题中，使用本章所开发的函数 randint 和 randfloat 进行简单的仿真。

22. 写一个程序仿真抛硬币实验。允许用户输入抛掷的次数，打印出抛出硬币正面的次数和抛出硬币反面的次数。硬币正面和反面所占的比例应当是多少？
23. 写一个程序仿真抛硬币实验，其中 60% 的时间里抛掷结果是正面朝上。要求用户输入抛掷的次数，打印出正面朝上的次数和反面朝上的次数。
24. 定义一个名为 Coin 的自定义数据结构。Coin 只有一个属性，即它的面值。将面值定义为一个 char 类型，‘H’代表正面，‘T’代表反面。使用 Coin 类为问题 22 和 23 编写解决方案。
25. 编写一个程序仿真投掷一个六面的骰子，骰子每面的点数依次为 1 ~ 6 点。允许用户输入投掷的次

数。分别打印出投掷结果为1点、2点、3点直到6点的次数。投掷中点数的比例分布是怎样的？

26. 写一个程序仿真投掷两个六面的骰子，允许用户输入投掷的次数。在仿真中投掷结果为8点的比例是多少？

27. 写一个程序仿真使用标号1～10的球进行的彩票抽奖。假定抽取的一个球是随机的，允许用户输入仿真的抽奖次数。仿真中包含一个奇数的比例是多少？仿真中三个数字包含数字7的比例是多少？仿真中同时出现1、2、3的比例是多少？

组件可靠性。下面的几个问题与计算若干组件配置方式可靠性的计算机仿真有关。使用本章中开发的randfloat函数。

28. 写一个程序仿真图6.17所示的配置方式的可靠性，其中组件1、2、3的可靠性分别为0.8、0.85、0.95。进行5000次仿真，打印出可靠性的估计值。（系统的解析可靠性为0.794。）

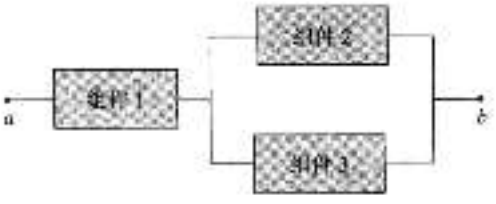


图6.17 配置方式1

29. 写一个程序仿真图6.18所示的配置方式的可靠性，其中组件1、2的可靠性为0.8，组件3、4的可靠性为0.95。使用5000次仿真，打印出可靠性的估计值。（系统的解析可靠性为0.9649。）

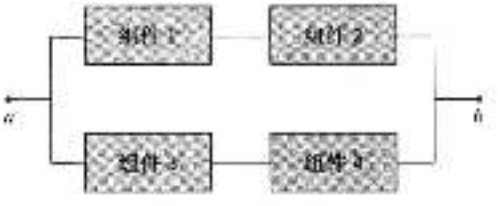


图6.18 配置方式2

30. 写一个程序仿真图6.19所示的配置方式的可靠性，其中所有组件的可靠性均为0.95。使用5000次仿真，打印出可靠性的估计值。（系统的解析可靠性为0.999 76。）

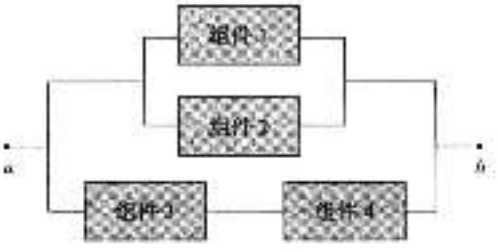


图6.19 配置方式3

飞行模拟器之风速。本问题与飞行模拟器中的风速的计算机仿真有关。假定某个特定区域的风速可以使用一个平均值和在某范围内的阵风值（该阵风值可以加入平均值）进行建模。例如，风速可能是10英里每小时，加上范围从22英里每小时到2英里每小时的噪声（这里指阵风），如图6.20所示。使用本章开发的randfloat函数。

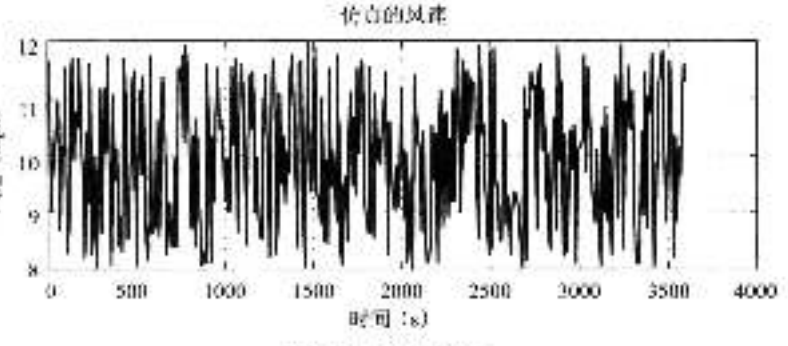


图6.20 风速仿真

31. 写一个程序，生成一个名为wind.dat的数据文件，文件中包含了1小时的仿真风速值。数据文件的每行应当包含以秒为单位的时间和对应的风速。时间应当从0秒开始，时间增量为10秒，数据文件的最后一行应该对应于3600秒。程序应当提示用户输入平均风速和阵风的范围。

32. 在问题 31 中，假定我们希望风速数据中在每个时间窗口内有 0.5% 的可能性遇到一场小型风暴。因此，修改问题 31 中的方案，使其在遇到一场风暴时，每 5 分钟风速增加 10 英里每小时。图 6.21 中给出了一个示例数据文件的图像，其中包含了 3 场风暴。

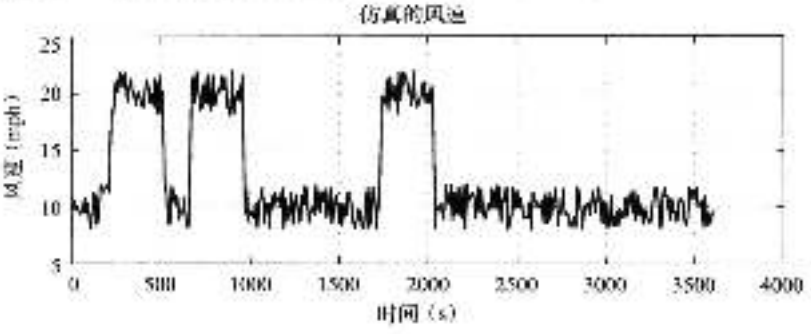


图 6.21 包含三场风暴的风速仿真

33. 在问题 32 中，假定在每次小型风暴中有 1% 的概率出现微爆气流。修改问题 32 中的方案，使其如果遇到一次微爆气流，每隔一分钟风速增加 50 英里每小时。图 6.21 中给出了一个示例数据文件的图像，其中包含了一次微爆气流。
34. 修改问题 32 中的程序，使其允许用户输入遇到风暴的概率。
35. 修改问题 32 中的程序，使其允许用户输入一次风暴持续的时间。
36. 修改问题 35 中的程序，使得风暴持续的时间是一个在 3 ~ 5 分钟之间的随机数。

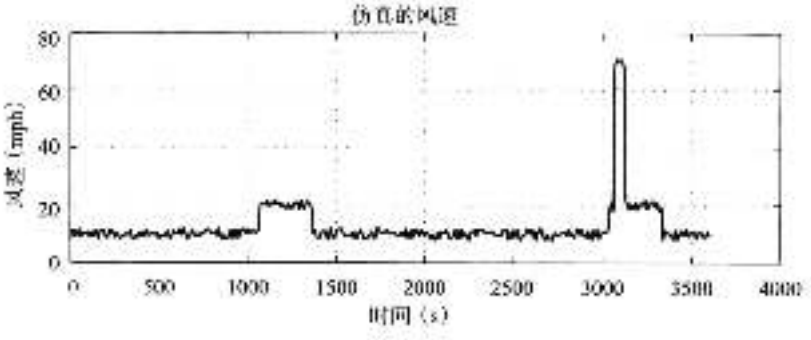


图 6.22 包含一次微爆气流的风速仿真

函数的根。下面的问题与求函数的实根有关。

37. 写一个程序，确定一个二次表达式的实根，假定由用户输入二次表达式的系数。如果根为复数，打印出提示信息。
38. 修改问题 37 中的程序，使得当根为复数时可以计算出根的实部和虚部。
39. 写一个 C++ 函数计算下面的数学函数：

$$f(x) = 0.1x^2 - x\ln x$$

假定对应的函数原型为：

```
double f(double x);
```

修改 6.10 节中的程序，使之可以搜索新函数的根而不是搜索多项式的根。通过在区间 [1, 2] 上搜索函数的根对程序进行测试。

40. 修改 6.10 节中的程序，使之可以搜索下面函数在用户指定区间上的根：

$$f(x) = \mathrm{sinc}(x)$$

使用本章中开发的 sinc 函数。

41. 在 6.10 节开发的程序中，我们搜索在两个端点上函数值符号相异的子区间，然后估计根在该子区间的中点上。一种更准确的根估计方法通常是采用通过函数值两点间连线与 x 轴的交叉点，如图 6.23 所示。使用相似三角形，可以使用下面的表达式计算出交点 c：

$$c = \frac{a*f(b)-b*f(a)}{f(b)-f(a)}$$

修改程序 chapter6_10，使用这种近似方式估算子区间上的根。

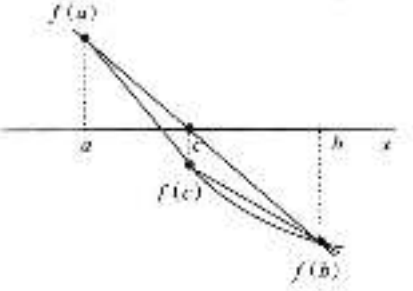


图 6.23 在 (a,b) 上的直线交点

42. 写一个可以被牛顿—拉弗森方法程序所使用的函数，用以计算多项式的值和它的导数。如果多项式本身的计算和导数的计算比较复杂的话，这些函数将十分有用。给出在程序中必须运行的修改。

数值积分。下面的问题与使用梯形法则进行数值积分有关。

43. 修改程序 chapter6_11，估算下面函数的积分。

$$f(x) = 3x-2x^2$$

44. 修改程序 chapter6_11，将梯形端点的 x 坐标和 y 坐标存储到数据文件中，以便后续进行图像绘制。

45. 写一个程序估算函数的积分，其中函数是由实验收集的数据点表示的，而非表达式表示。且这些数据点存储在一个文件中。文件中包含了代表梯形底的一组 x 坐标值和一组代表梯形高的 y 坐标。注意，数据点的数目确定了梯形的数目。你必须为每组新的数据点重新计算梯形的底，因为你不能假定所有的 x 坐标之间都有距离相同的间隔。

带返回值的函数。

46. 假定我们有 n 个不同的对象。在一行上对它们进行排列有许多不同的顺序。事实上，对于 n 个对象，有 $n!$ 种顺序，或者排列（permutation）。如果我们在 n 个对象，选择其中的 k 个，那么这 k 个对象有 $n!/(n-k)!$ 种可能的顺序。这表示，从 n 个不同的对象中选择 k 个对象进行排序的可能顺序数为 $n!/(n-k)!$。写一个名为 permute 的函数，它接收参数 n 和 k，并返回从 n 个对象中选择 k 个对象进行排列的可能顺序数。（如果我们考虑数字集合 1，2，3，那么两个数字的排列有 1，2，2，1，1，3，3，1，2，3 和 3，2。）假设对应的原型为

```
int permute(int n, int k);
```

47. 排列（同题 46）关注顺序，但是组合则不关注。因此，给定 n 个不同的对象，选择其中的 n 个对象，只有一种组合，但是一次选择 n 个对象则有 $n!$ 排列。从 n 个不同对象中选择 k 个对象的组合数等于 $n!/(k!)(n-k)!$。编写一个名为 combine 的函数，接收参数 n 和 k，返回从 n 个对象中选择 k 个对象的组合数。（如果我们考虑数字集合 1，2，3，那么两个数字的组合为 1，2，1，3 和 2，3。）假设对应的原型为

```
int combine(int n, int k);
```

48. 一个角的余弦值可以通过下面的无穷级数计算：

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

编写一个程序，从键盘读取角 x（以弧度为单位），然后在一个函数中使用级数的前 5 项计算角的余弦值。将计算的值和使用 C++ 库函数计算的余弦值都打印出来。

49. 修改问题 48 中的程序，使近似过程在从一项级数的绝对值开始小于 0.0001 时终止。除了打印出计算值外，还要打印出在级数近似过程中使用的级数项数。

一维数组

工程挑战：海啸预警系统

在一次8.9级的地震（这是日本历史上有记录以来最大的地震）袭击日本后，海啸波穿越了太平洋传播了数千英里。在太平洋海啸预警中心（PTWC）发出预警后，低洼地带的人们都被疏散了。这幅国家海洋和大气局（NOAA）在2011年3月11日发布的图像展示了海啸在穿越太平洋盆地时所预测的波高。最大的波高预测出现在震中附近。波浪在穿越太平洋深水区时波高将降低，但是在邻近沿海地区时波高将增加。一般而言，波浪的能量随着距离而减少，近岸高度也将降低。PTWC使用地震数据和实时海洋数据来计算可能的威胁。检测仪和DART（深海评估和海啸报告）浮标被用于监视以确定海啸的形式。当这些机制建立起来后，就可以生成海啸预报，形成海啸预警。本章我们使用浮标数据计算在20分钟时间内的最高波高。

教学目标

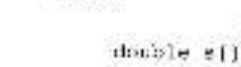
在本章中，我们所讨论的问题解决方案中包括：

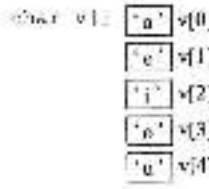
- 一维数组和容器
- 用于数据统计分析的自定义模块
- 用于数据排序和数据搜索的函数
- 用于计算某个事件概率的函数
- 字符串和字符串对象
- 在头文件cstring和string中定义的函数
- 自定义头文件

7.1 数组

在解决工程问题时，将与问题相关的数据进行可视化是很重要的。有时，数据仅由一个数组成，如圆的半径。有时则可能是由一对数字表示的平面上的一个点，其中一个数表示x坐标，另一个数表示y坐标。有时候我们需要处理一组类似的数据值，但又不希望给每个值一个单独的名字。例如，假定我们有100个温度测量值需要进行某些计算。显然，我们不希望为这些温度测量值使用100个不同的名字，因此我们需要一种方法通过使用一个标识符来对这一组数值进行处理。解决这一问题的方案之一就是使用称作数组的数据结构。

一维数组（one-dimensional array）可以视作按照一行或一列排列的值的列表，如下所示：





我们为数组赋予一个类型和一个标识符，然后使用偏移量（offset）来区分数组中的元素（element）或数值，偏移量也称作索引或下标。因此，在这个示例数组中，数组 s 中的第一个值可以通过 s[0] 引用，第二个值通过 s[1] 引用，数组 v 中的最后一个元素则通过 v[4] 引用。在 C++ 中，数组标识符表示第一个元素的地址，因此其偏移量总是从 0 开始，并以 1 个单位增长。

7.1.1 定义和初始化

数组使用声明语句来定义。数组中标识符后所跟的括号里的整型表达式指明了数组中的元素个数。注意，数组中的所有元素都必须具有相同的数据类型。下面给出了两个示例数组的声明语句：

```
double s[5];
char v[5];
```

数组可以在声明语句中初始化或者使用程序语句进行赋值。我们使用初始化列表（initialization list）在声明语句中对数组进行初始化。初始化列表是一个使用逗号分隔的数值列表。下面的语句定义并初始化了示例数组 s 和 v：

```
int s[6]= 0.5, 0.0, -0.1, 0.2, 0.15, 0.2;
char v[5] = {'a', 'e', 'i', 'o', 'u'};
```

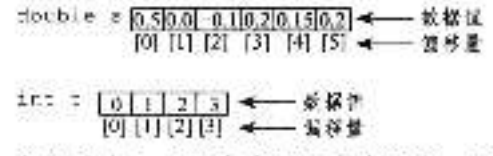
如果初始化序列比数组长度短，那么剩下的数组元素的值将被初始化为 0。因此，下面的语句定义了一个包括 100 个值的整型数组，每个值都被初始化 0：

```
int i[100]={0};
```

如果数组没有指定大小，但是使用了初始化列表，那么数组的大小将被设定为初始化序列中值的数目：

```
double s[]={0.5, 0.0, -0.1, 0.2, 0.15, 0.2};
int i[]={0, 1, 2, 3};
```

数组 s 和 i 的内存快照如下：



数组的大小必须在声明语句中指定，可以在括号中使用常量指定或者通过初始化列表指定。

数组声明语句：一维数组的声明将会分配一个连续的内存地址块，数组名对应于内存块的首地址。数组中的每个值可以通过数组名和偏移量来进行访问。数组中每个值的数据类型都必须相同。

语法

```
数据类型 标识符[大小] = 初始化列表];
```

示例

```
int data[5];// 为5个整型值分配连续块内存
```

内存分布：数据 | ? | ? | ? | ? | ?

偏移量 0 1 2 3 4

```
char vowels[5] = { 'a', 'e', 'i', 'o', 'u' };// 分配并初始化
```

内存分布：数据 | 'a' | 'e' | 'i' | 'o' | 'u'

偏移量 0 1 2 3 4

```
double t[100] = {0.0};// 分配内存并将所有值初始化为 0.0
```

内存分布：数据 | 0 | 0 | 0 | … | 0

偏移量 0 1 2 … 99

合法引用：

```
cout << vowels[2];
cout <<t[2];
```

非法引用：

```
cout << vowel[5]; //invalid offset
cout << t[-1]; //invalid offset
```

数组也可以在声明之后对其赋值。例如，假如我们希望给一个 double 类型的数组 g 赋值为 0.0,0.5,1.0,1.5,…,10.0。因为有 21 个不同的值，在初始化列表中列出这些值显得很冗长，但是在一个 for 循环内对数组进行赋值却很简单，如程序 chapter7_1 所示。

7.1.2 伪代码

```
main: set i to 0
      while i<21
          assign i*0.5 to t[i]
          increment i by 1
      print heading
      set i to 0
      while i<21
          print t[i]
          increment i by 1
/*-------------------------------------------------------------*/
/* Program chapter7_1                                          */
/* This program assigns a set of values to a                   */
/* one-dimensional array then prints a list of the array       */
/* offsets and values to standard output.                      */
/*-------------------------------------------------------------*/

#include<iostream> //Required for cout.
```

```
#include<iomanip> //Required for setw().
using namespace std;

int main()
{
  //Declare variables.
  double t[21]; //The array.
  int i; //The loop index.

  //Assign 21 value to array t.
  for(i=0; i<21; i++)
  {
     t[i] = i*0.5; //i provides offset and value.
  }

  //Print list of array offsets and values.
  //Print heading.
  cout << "21 values assigned to t"<< endl
       << "Offset Value" << endl;
  //Print list inside for loop.
  for(i=0; i<21; i++)

    cout << setw(5) << i << setw(10) << t[i] << endl;
  }
  return 0;
}
```

程序 chapter7_1 的输出如下：

```
21 values assigned to t
  Offset         Value
     0             0
     1           0.5
     2             1
     3           1.5
     4             2
     5           2.5
     6             3
     7           3.5
     8             4
     9           4.5
    10             5
    11           5.5
    12             6
    13           6.5
    14             7
    15           7.5
    16             8
    17           8.5
    18             9
    19           9.5
    20            10
```

在这里需要注意最后的偏移量为20，而不是21。这是因为第一个元素的偏移量是从0开始计算的，而不是1，所以最后一个元素的偏移量为20，而不是21。将偏移量指定为一个超过最大合法偏移量的值是一个常见的错误。这样的错误可能很难发现，因为编译器不会报错。程序将继续访问一个超过数组声明的内存空间范围的内存地址，访问数组边界之外的

内存地址可能会导致诸如“段错误”或“总线错误”之类的执行错误。而更常见的情况是，程序执行时无法检测到这些错误，但是会导致不可预测的程序结果，因为你的程序可能已经修改了一个作为其他用途的内存地址中的内容。

数组通常用于存储从数据文件中读取出来的信息。例如，假如我们有一个名为 sensor3.dat 的数据文件，其中包含了由地震检波器采集的 10 个时间和移动测量值。程序 chapter7_2 说明如何从数据文件中读出这些值并将它们赋给名为 time 和 motion 的数组。

```
/*                                                                    */
/* Program chapter7_2                                                 */
/*                                                                    */
/* This program reads time and motion values from an input file       */
/* and assigns the values to the arrays time and motion.              */
/* Input values are printed to standard output for verification.      */

#include<iostream> //required for cout
#include<fstream>  //required for ifstream

using namespace std;

int main()
{
   // Declare objects.
   double time[10], motion[10];
   ifstream sensor3(" sensor3.dat" );

  // Check for successful open and read data into arrays.
  if(!sensor3.fail())
  {
     for (int k=0; k<10; ++k)
     {
        sensor3 >> time[k] >> motion[k];
        cout << time[k] << '\t' << motion[k] << endl;
     }
  }
  else
  {
     cout << " Could not open file sensor3.dat..goodbye."  << endl;
  }
  return 0;
}
```

如果数据文件 sensor3.dat 中的数据集如下：

```
0.0     1
0.1     1
0.2     2.5
0.3     1.7
0.4     2
0.5     2.5
0.6     4.5
0.7     1.5
0.8     1.5
0.9     1
```

那么数组 time 和 motion 中将包含下面的值：

double time
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

double motion
1.0
1.0
2.5
1.7
2.0
2.5
3.5
1.5
1.5
1.0

程序 chapter7_2 的输出如下：

```
0       1
0.1     1
0.2     2.5
0.3     1.7
0.4     2
0.5     2.5
0.6     3.5
0.7     1.5
0.8     1.5
0.9     1
```

练习

给出下面每组语句中定义的数组内容。

1. `int x[10]={-5, 4, 3};`

2. `char letters[] = {'a', 'b', 'c'};`

3.
```
double x[4];
...
x[1] = -5.3;
x[2] = x[3] = fabs(x[1]);
```

4.
```
double time[9];
...
for (int k=0; k<=8; k++)
{
   time[k] = (k-4)*0.1;
}
```

5. 给出下面程序的输出：

```
#include<iostream>
using namespace std;
int main()
{
  int arr[5], i, k;
  for(i=0; i<5; ++i)
  {
     for(k=1; k<3; ++k)
     {
       arr[i] += k*i;
     }
     cout << "arr[" << i << "] is " << arr[i] << endl;
  }
  return 0;
}
```

7.1.3 计算与输出

数组元素的计算与简单对象的计算类似，但是必须使用一个偏移量来指定特定的数组元素。为了说明，下一个程序中使用一个 for 循环从一个数据文件中读取了给定数目的浮点值，并将这些值赋给数组 y。y 声明的大小为 100，因此程序将至多允许给 y 赋 100 个值，以避免超过数组边界。程序计算了数据的平均值并存储在 yAve 中。然后，程序将统计并打印出数组中大于平均值的数值数目。

```
/*-------------------------------------------------------------------*/
/* Program chapter7_3                                                */
/*                                                                   */
/* This program reads at most 100 values from a data                 */
/* file and determines the number of values greater                  */
/* than the average.                                                 */

#include<iostream> //Required for cin, cout, cerr.
#include<fstream> //Required for ifstream.
#include<string> //Required for string.
using namespace std;

int main()
{
  // Define maximum array size constant
  const int N = 100;

  // Declare and initialize objects.
  string filename;
  int count=0, numberOfValues;
  double y[N], yAve, sum=0;
  ifstream lab;

  // Prompt user for name of input file
  cout << "Enter name of the input file";
  cin >> filename;

  // Open data file and read data into an array.
  // Compute a sum of the values.
  lab.open(filename.c_str());
  if (lab.fail())
  {
    cout << "Error opening input file\n";
    exit(1);
  }
  /* File has been opened.                                           */
  /* Read number of data values                                      */
  lab >>numberOfValues;
  // Don't exceed the bound of the array.
  if(numberOfValues > N)
  {
    cerr << "Number of data values," << numberOfValues
         << "exceeds maximum array size of" << N << endl
         << N << "values will be read." << endl;
    numberOfValues = N;
  }
  int k;
  for (k=0; k<numberOfValues; ++k)
  {
        lab >> y[k];
        sum += y[k];
  }
```

```
// Compute average and count values that
// are greater than the average.
yAve = sum/numberOfValues;
for (int k=0; k<numberOfValues; ++k)
{
    if (y[k] > yAve)
      count++;
}

// Print count.
cout << count << "values greater than the average \n";

// Close file and exit program.
lab.close();
return 0;
}
//.....*/
```

如果这个程序的目的是计算数据文件中数值的平均值，那么数组就不是必需的。用于读取数据的循环可以将每个值读取到相同的对象中，并在下次读取数据之前将值累加到和中。但是由于我们需要将每个值与平均值进行比较，以确定超过平均值的数值数目，所以我们选择使用数组将每个值存储下来，这样我们就可以再次进行访问。

需要打印出的数组中的值通过指定偏移量确定。例如，下面的语句打印出了上面例子中数组 y 的第一个和最后一个值。

```
cout << "First and last array values: \n";
cout << y[0] << " " << y[numberOfValues-1] << endl;
```

下面的循环打印出了数组 y 所有的 100 个值，每个值占一行：

```
cout << "y values: \n";
for (int k=0; k<N; ++k)
{
   cout << y[k] << endl;
}
```

注意，数组中的值一次只能打印一个，因为一般而言，C++ 不支持数组上的批量输出操作。当打印一个大型数组时，我们可能需要在一行中打印多个数。下面的语句使用取模操作符在每行打印 5 个值后跳转到新的一行继续打印：

```
cout << "y values: \n";
for (int k=0; k<numberOfValues; ++k)
{
    if (k%5 == 0)
       cout << y[k] << endl;
    else
       cout << y[k] << " ";
}
cout << endl;
```

与这里给出语句相类似的语句也可以用于将数组的值写入数据文件中。例如，下面的语句使用文件流 sensor 将 y[k] 的值打印到数据文件中的一行里：

```
sensor << y[k] << endl;
```

因为使用了 endl 操纵符，下一个值将写入到数据文件新的一行中。

修改

1. 重写程序 chapter7_3，不使用数组。提示：计算出平均值，然后关闭并重新打开输入文件，统计出大于平均值的数值数目。在第二次读取文件时你需要清除 cofbit 吗？
2. 考虑下面的程序：

```
#include<iostream>
using namespace std;

int main()
{
  char greeting[] = "Hello World";
  int numbers[] = {1,2,3,4,5};
  cout << greeting << endl;
  cout << numbers << endl;
  return 0;
}
```

编译运行该程序，解释输出。

数组的最大尺寸（maximum size of an array），即可以赋给数组的元素数目，是在类型声明中确定的，并可以在后面的程序中用来避免超出数组的边界。如果最大尺寸被改变了，那么程序中可能有若干处都需要进行相应修改。如果使用一个符号常量来指定数组的声明大小，那么在改变数组的最大尺寸时将会变得简单。这样，要改变最大尺寸，只需要改变符号常量的值即可。在包含多个模块的程序中或在多个程序员共同工作的项目中，这条风格建议尤其重要。下面的许多程序都说明了使用符号常量定义数组的最大尺寸的用法。

表 7.1 中给出了包含了偏移括号在内的更新的优先级顺序。中括号和圆括号在其他操作符之前最先进行结合。如果圆括号和中括号在同一语句中，则按照从左到右的顺序结合；如果它们是嵌套的，那么最内层的优先计算。

表 7.1 操作符优先级

优先级	操作符	结合性	优先级	操作符	结合性
1	() []	最内层优先	7	&&	从左向右
2	~ -- + - !(类型)	从右向左（一元）	8	||	从左向右
3	* / %	从左向右	9	?:	从右向左
4	+ -	从左向右	10	= += -= *= /= %=	从右向左
5	<< >>	从左向右	11	,	从左向右
6	== !=	从左向右			

练习

假定数组 s 由如下语句定义：

```
int s[] {5, 8, 15, 21, 20, 41};
```

确定下面每组语句的输出：

```
1. for (int k=0; k<=5; k+=2)
   {
     cout << s[k] << " " << s[k+1] << endl;
   }
2. for (int k=0; k<=5; k++)
   {
     if (s[k]%2 == 0)
        cout << s[k] << " ";
   }
   cout << endl;
```

7.1.4 函数参数

当需要将数组中的信息传递给函数时，一般需要两个参数：一个参数指明数组；另一个参数指明所使用的数组元素数目。通过指明要使用的数组元素数目，函数将变得更灵活。例如，如果函数指明一个整型数组，那么该函数将可以使用任意整型数组；用于指出元素个数的参数可以保证我们使用正确的数组尺寸。此外，我们用到的数组的元素数目在不同的情况下都不尽相同。例如，数组可能从数据文件中读取出元素的值，此时元素的个数取决于程序运行时所使用的数据文件。但在所有这些例子中，数组都必须声明最大尺寸，所使用的元素数目应该小于等于最大尺寸。

考虑下面所给出的程序 chapter7_4，该程序从一个数据文件中读取一个数组，并调用函数确定数组中的最大值。对象 npts 用于统计从数据文件中读取出来并存储在数组中的数值的实际数目；npts 的值小于等于所定义的数组的大小，在这里是 100。必须注意的是我们不能为数组赋值超过 100 个，同时还需要准确记录赋给数组的数值数目。这个函数有两个形参，如函数原型语句中所示，分别是数组名和数组中存放的点的实际数目。图 7.1 中给出了程序 chapter7_4 的流程图。

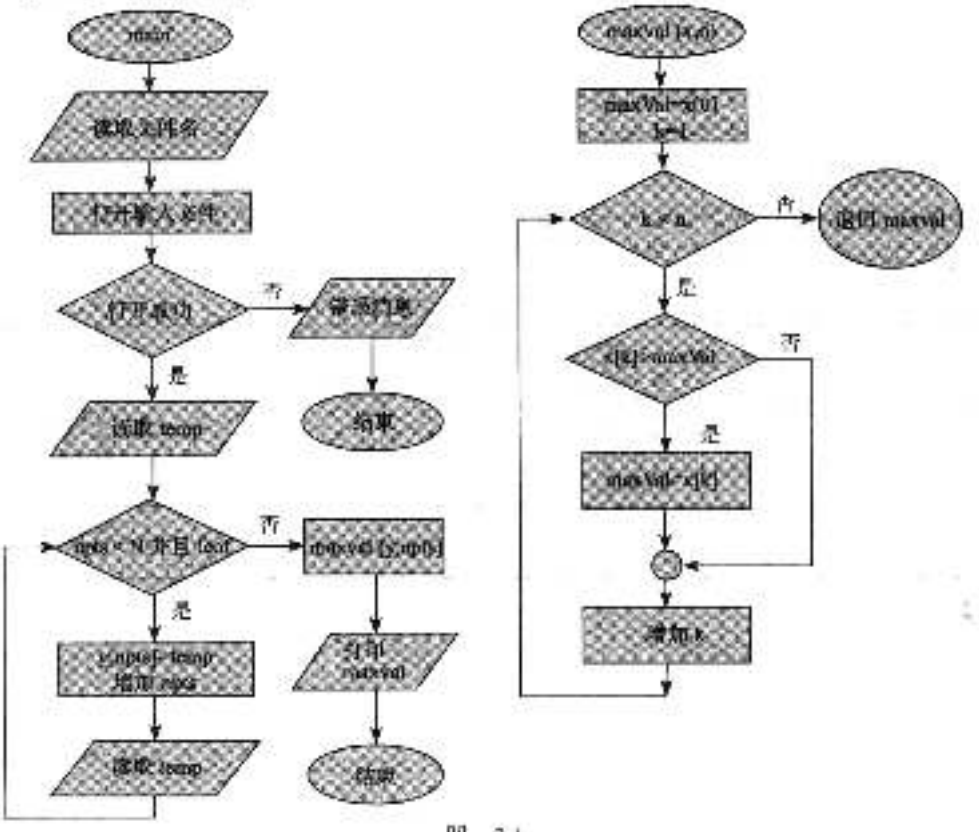


图 7.1

```
/*                                                              */
/* Program chapter7_4                                           */
/*                                                              */
/* This program reads values from a data file and               */
/* calls a function to determine the maximum value              */
/* with a function.                                             */
```

```
#include<iostream> //Required for cin, cerr.
#include<fstream> //Required for ifstream.
#include<string> //Required for string.
using namespace std;

// Define function prototypes.
double maxval (const double x[], int n);

int main()
{
  // Declare objects.
  const int N = 100;
  int npts 0;
  double y[N], temp;
  string filename;
  ifstream lab;

  // Prompt user for file name and open data file.
  cout << "Enter the name of the data file:";
  cin >> filename;
  lab.open(filename.c_str());
  if(lab.fail())
  {
    cerr << "Error opening input file\n";
    exit(1);
  }

  // Read a data value from the file.
  lab >> temp;

  // While there is room in the array and
  // and end of file was not encountered,
  // assign the value to the array and
  // input the next value.

  while (npts < N && !lab.eof() )
  {
      y[npts] = temp; // Assign data value to array.
      ++npts;         // Increment npts.
      lab >> temp;    // Input next value
  }

  // Find and print the maximum value.
  cout << "Maximum value: " << maxval(y,npts) << endl;

  // Close file and exit program.
  lab.close();

  return 0;
}
/*-------------------------------------------------------------*/
/* This function returns the maximum                           */
/* value in the array x with n elements.                       */

double maxval (const double x[], int n)
{
  // Declare local objects.
  double maxVal;

  // Determine maximum value in the array.
  maxVal = x[0];
  for (int k=1; k<n; ++k)
  {
    if (x[k] > maxVal)
    maxVal = x[k];
```

```
    }

    // Return maximum value. /
    return maxVal;
}
/*                                                                    */
```

图 7.2 中给出了程序 chapter7_4 的程序跟踪和内存快照。图 7.3 中给出了函数 maxval() 的单独的程序跟踪和内存快照。

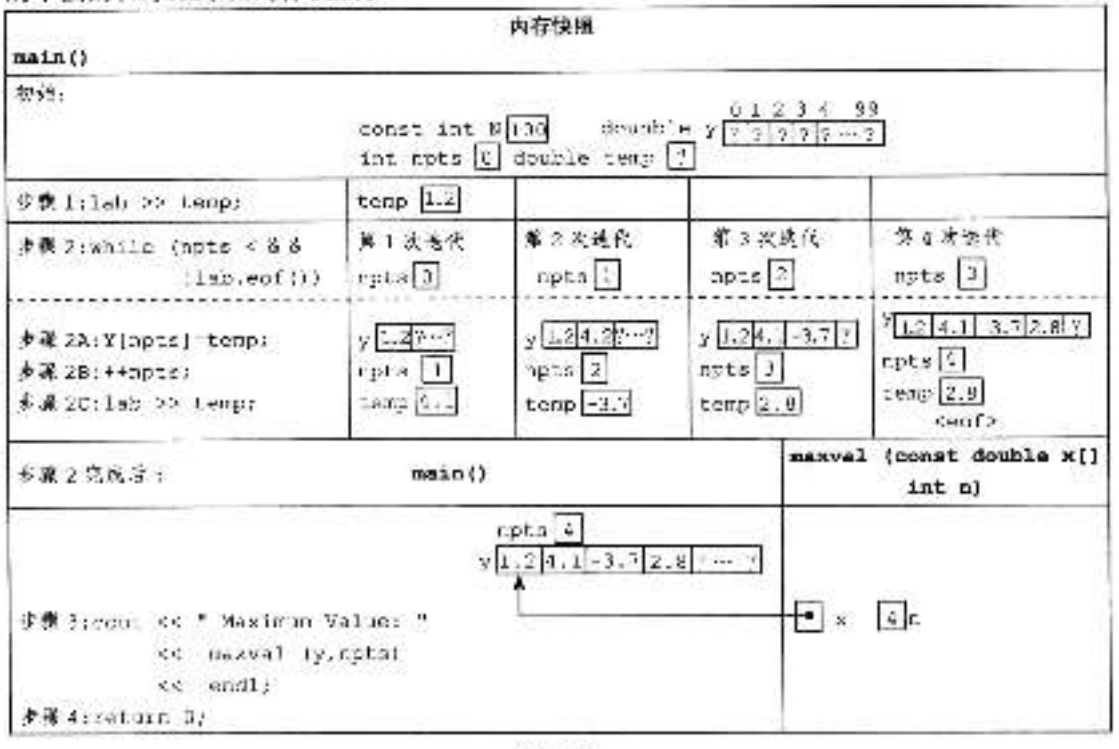


图 7.2

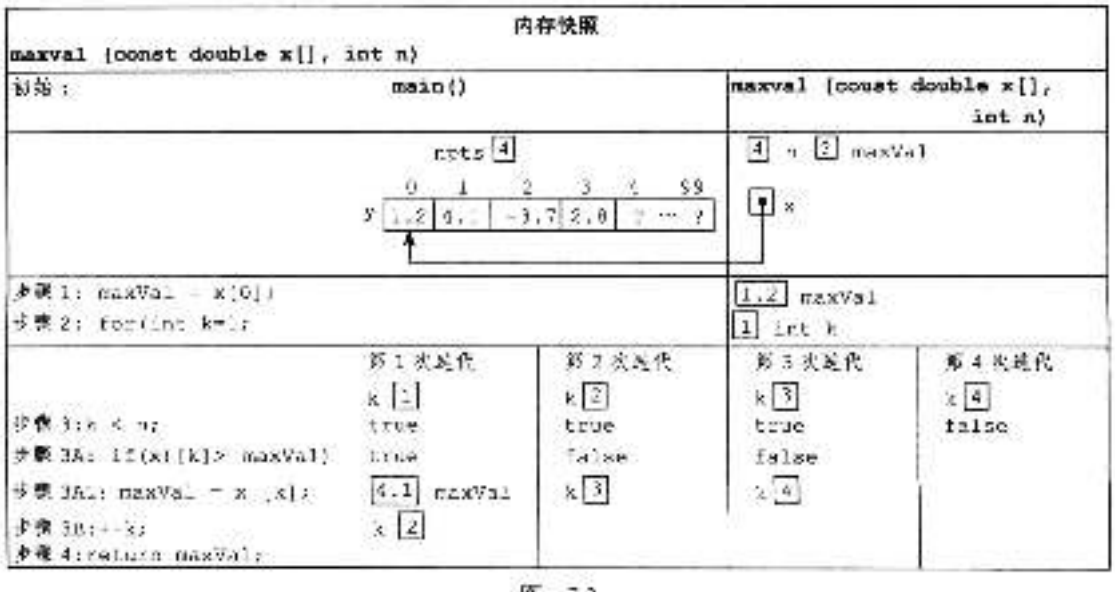


图 7.3

使用数组作为参数与使用简单对象作为参数有一个十分显著的差别。当一个简单对象作为参数时，默认情况总是采用值传递的方式。当数组作为参数时，总是采用引用传递的方式。我们前面提到引用传递的方式意味着传递给形参的是参数的内存地址，而不是值。当一个数组作为参数时，第一个元素的地址将被传递给函数。函数使用这个地址和偏移量来引用源数组的值，如图 7.2 和图 7.3 所示。任何在函数中对数组所做的修改都会直接反映到数组参数上。因为函数是直接访问数组参数的，所以我们必须小心注意，不要在函数中对数组进行不经意的修改。当然，当我们希望改变数组中的值时也会有例外，在本章后面的例子中我们将看到这样的情况。

因为我们希望确保 maxval 函数不会改变数组参数的值，所以我们在函数原型和函数头中使用了 const 限定符，这将防止函数为数组赋新值。任何为数组 x 中的元素赋值的尝试都会导致编译错误。

练习

假定定义了下面的对象：

```
int k=5;
double data[]={1.5, 3.2, -5.1, 9.8, 8.7, 5.21};
```

给出下面表达式的值，这些表达式都调用了本节给出的 maxval 函数。

1. maxval(data, 6);
2. maxval(data, 5);
3. maxval(data, k-3);
4. maxval(data, k%5);

7.2 解决应用问题：飓风等级

飓风是伴随着强风和暴雨的热带风暴（它们在北太平洋西部称作台风，在印度洋被称作气旋。）这些热带风暴（或者气旋）是一般在夏季或初秋形成的低气压区域。在卫星图像上可以很容易地看到大的旋转云团，由于风暴可能对人口密集区域造成危害，因此这些风暴总是处于人们的监控之下。如果风暴中的风速在 38 ~ 74 英里每小时之间，则称作热带风暴；如果风速超过了 74 英里每小时，则称作热带气旋或者飓风。Saffir-Simpson 分类根据风速定义了飓风强度等级。本节我们将对 Saffir-Simpson 分类进行更详细的定义，并开发一个程序从包含风暴和峰值风速的数据文件中读取信息。基于这些风速，我们打印出一份报告，其中包含了强度达到了飓风分类等级的风暴信息。

飓风强度的 Saffir-Simpson 等级用于根据风暴对人口密集区域可能造成的破坏量对飓风进行分类。其中 5 个等级的主要特征描述如下：

等级 1	风速 74 ~ 95 英里 / 时
	风暴潮 4 ~ 5 英尺
	财产损失小
等级 2	风速 96 ~ 110 英里 / 时
	风暴潮 6 ~ 8 英尺
	财产损失中等
等级 3	风速 111 ~ 130 英里 / 时
	风暴潮 9 ~ 12 英尺
	财产损失很大

等级4　风速131～155英里/时
风暴潮13～18英尺
财产损失极大

等级5　风速155英里/时以上
风暴潮18英尺以上
灾难性财产损失

表7.2中包含了在1950～2002年间袭击美国的12个最强飓风列表。(可以通过互联网获取更多有关这些飓风的信息。)

表7.2　1950～2002年间美国遭遇的最强飓风

飓风	年份	等级	飓风	年份	等级
Hazel	1954	4	Frederic	1979	3
Audrey	1957	4	Allen	1980	3
Donna	1960	4	Gloria	1985	3
Carla	1961	4	Hugo	1989	4
Camille	1969	5	Andrew	1992	5
Celia	1970	3	Opal	1995	3

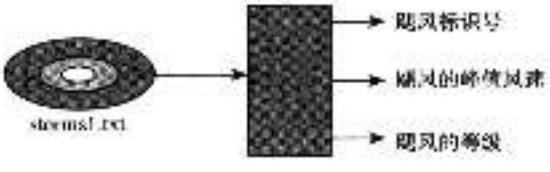
每年有可能形成飓风的风暴超过100个。编写一个程序从一个包含当前风暴信息的数据文件中读取数据，数据文件中的信息由一个标识号和到目前为止测得的风暴最高风速(单位为英里每小时，或者mph)。程序应当打印出所有风速达到了飓风标准的风暴列表。除了标识号(一个整数)外，还要打印出对应的峰值风速和对应的飓风强度等级。此外，还要在具有最高风速的飓风对应的标识号后打印出一个星号。

1. 问题描述

使用一个含有当前风暴信息的数据文件确定哪些风暴属于飓风。

2. 输入/输出描述

I/O草图表明数据文件为输入，输出为飓风信息。



3. 用例

假设数据文件中包含下面5组数据：

标识号	峰值风速	标识号	峰值风速
142	38	177	76
153	155	181	63
162	59		

对应的输出是下面的报告：

可以称为飓风的风暴

标识号	峰值风速 (mph)	等级
153*	155	4
177	76	1

问题一下，星号标出了具有峰值风速的风暴。

4. 算法设计

我们首先给出分解提纲，它将解决方案分为了一系列顺序的步骤。如果只是为了打印出可作为飓风的风暴信息，我们不需要数组。因为飓风状态只取决于风速，所以我们可以在读取文件时就完成这项工作。但是，我们还需要使用星号来指出风暴的峰值风速，所以我们需要将所有的信息存储到数组中。在我们确定最大值后，可以在数据打印时在最大风速的飓风信息中加入星号。

分解提纲

1）将风暴数据读入数组，确定最大风速；

2）计算强度等级，打印出可以视作飓风的风暴信息，并在最大值上加上星号。

我们将确定强度等级的步骤写入一个函数。

细化的伪代码

```
main: if file cannot be opened
          print error message
else
      read data into arrays, and determine max speed, npts
      set k to 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])
                  add 1 to k

category(mph):
       category =1
       if mph ≥ 96
             category =2
       if mph ≥ 111
             category =3
       if mph ≥ 131
             category =4
       if mph ≥ 155
             category =5
```

伪代码中的步骤已经足够详细，可以转换成 C++ 语句：

```
/*-------------------------------------------------------------*/
/* Program Chapter7_5                                          */
/*                                                             */
/* This program reads storm values from a data file            */

#include<iostream> //Required for cin, cout, cerr.
#include<fstream> //Required for fin.
using namespace std;
```

```
//Function Prototypes
double category(double speed);

int main()
{
  //Declare and initialize variables.
  const int MAX_SIZE = 500;
  int k(0), npts, id[MAX_SIZE];
  double mph [MAX_SIZE], max(0);
  ifstream fin("storm1.txt");
  if(fin.fail())
  {
    cerr << "Could not open file storm1.txt" << endl;
    exit(1);
  }
  //Read data and determine maximum mph.
  fin >> id[k] >> mph[k];
  while(!fin.fail())
  {
    if(mph[k] > max)
    {
      max = mph[k];
    }

    ++k;
    fin >> id[k] >> mph[k];
  }//end while
  npts = k;

  //Print hurricane report.
  if(max >= 74)
  {
    cout << "Storms that Qualify as Hurricanes \n"
         << "Identification\t Peak Wind(mph)\t Category\n";
  }
  else
  {
    cout << "No hurricanes in the file \n";
  }
  for(k=0; k<npts; ++k)
  {
    if(mph[k] >= 74)
    {
      if(mph[k] == max)
      {
        cout << "\t" << id[k] << "\t\t\t" << mph[k] << "\t
             << category(mph[k]) << endl;
      }
      else
      {
        cout << "\t" << id[k] << "\t\t\t" <<mph[k] << "\t
             << category(mph[k]) << endl;
      }
    }//end if k
  }//end for
  fin.close();
  return 0;
}
/*-------------------------------------------------------------*/
/* This function determines the hurricane intensity           */
```

```
/* category.                                                    */

double category(double speed)
{
  //Declare variables.
  int intensity(1);

  //Determine category.
  if(speed >= 155)
  {
     intensity=5;
  }
 else if(speed >= 131)
 {
   intensity = 4;
 }
 else if(speed >= 111)
 {
   intensity = 3;
 }
 else if(speed >= 96)
 {
   intensity = 2;
 }
 return intensity;
}
/*------------------------------------------------------------------*/
```

5. 测试

我们使用包含用例的文件进行测试，输出结果如下：

可以称为飓风的风暴

标识号	峰值风速 (mph)	等级
153*	155	4
177	76	1

修改

这些问题与本节开发的打印飓风强度报告的程序有关。

1. 修改程序，使其只打印风速最大的飓风信息。
2. 修改程序，使其还打印出数据文件中风暴的数目。
3. 修改程序，使其还打印出数据文件中飓风的数目。
4. 修改程序，使其在报告末尾打印出每个等级的飓风数目。
5. 修改程序，使其生成一个新数据文件，其中包含了每个飓风的信息及其强度。

7.3 统计表征数

分析从工程实验中收集到的数据是评估实验的重要组成部分。分析涉及的范围从简单的数据计算，如计算平均值，到更复杂的分析。许多使用数据进行的计算值或测量值都是统计表征数（statistical measurement），因为它们拥有随数据集改变而改变的统计属性。例如，我们每次计算 60 度角的正弦值都是一个准确值（即每次计算都是相同值），但是我们得到的每加仑汽油可行驶的英里数就是一个统计表征数，因为它的变化取决于温度、行驶速度、路

况，地形等多个参数。

7.3.1 简单分析

当分析一组实验数据时，我们通常计算最大值、最小值、平均值（mean）和中值（median）。本节我们使用数组作为输入，开发了用于计算这些值的函数。这些函数（存储在文件 stat_lib.cpp 中）对本书后面开发的许多程序以及本章结尾的问题解决方案都很有帮助。需要注意的是，这些函数假定在数组中至少有一个数值，并假设数组包含的都是 double 类型的值。

最大值和最小值。前一节中给出了确定数组中最大值的函数，这里给出了相似的确定数组中最小值的函数：

```
/*-------------------------------------------------------*/
/*  This function returns the minimum                    */
/*  value in an array x with n elements.                 */

double mlevel(const double x[], int n)
{
// Declare objects.
   double min_x;
// Determine minimum value in the array.
   min_x = x[0];
   for (int k=1; k<=n-1; ++k)
   {
      if (x[k] < min_x)
         min_x = x[k];
   }

// Return minimum value.
   return min_x;
}
/*-------------------------------------------------------*/
```

平均值。希腊字母 μ 用于代表平均值，如下面公式所示，其中使用了求和记号：

$$\mu = \frac{\sum_{k=0}^{n-1} x_k}{n}$$

这里

$$\sum_{k=0}^{n-1} x_k = x_0 + x_1 + x_2 + \cdots + x_{n-1}$$

该函数计算出了含有 n 个数值的 double 类型数组的平均值：

```
/*-------------------------------------------------------*/
/*  This function returns the average or                 */
/*  mean value of an array with n elements.              */

double mean(const double x[], int n)
{
   // Declare and initialize objects.
   double sum(3);

   // Determine mean value.
   for (int k=0; k<n; ++k)
   {
```

```
        sum += x[k];
    }

    //  Return mean value.
    return sum/n;
}
/*-------------------------------------------------------------*/
```

注意在声明语句中对象 sum 被初始化为 0，也可以在赋值语句中将对象初始化为 0。在这两种情况中 sum 的值都在函数被调用时被初始化为 0。

中值（median）。假定一组数值已经经过排序，那么中值就是这一组值中间的那个数值。如果数值数目为奇数，那么中值就是中间的那个数；如果数值数目为偶数，那么中值是中间两个数的平均值。例如，1，6，18，39，86 的中值为 18，而 1，6，18，39，86，91 的中值为中间两数的平均值，即（18 + 39）/2 或 28.5。假定有一组排好序的数值存放在数组中，数组中包含的数值数目为 n。如果 n 为奇数，那么中值的偏移量可以使用 floor（n/2）表示，如 floor（5/2）=2。如果 n 为偶数，那么两个中间的数值的偏移量可以表示为 floor（n/2）-1 和 floor（n/2），如 floor（6/2）-1=2 和 floor（6/2）=3。下一个函数确定了存储在数组中的一组值的中值。我们假定数值是排好序的（升序或者降序）。如果数组没有排序，本章后面将会开发一个函数用于数值排序，在计算中值的函数中可以调用该函数：

```
/*-------------------------------------------------------------*/
/*  This function returns the median                           */
/*  value in an array x with n elements                        */
/*  The values in x are assumed to be ordered.                 */

double median(const double x[], int n)
{
    //  Declare objects.
    double median_x;
    int k;

    //  Determine median value.
    k = floor(n/2);
    if (n%2 != 0)
        median_x = x[k];
    else
        median_x = (x[k-1] + x[k])/2;

    //  Return median value.
    return median_x;
}
/*-------------------------------------------------------------*/
```

下面完成该函数的过程，使用前面讨论中给出的两组数据。

7.3.2 方差和标准差

一组数据最重要的统计表征数之一就是方差。在给出方差的数学定义前，先给出一个直观的理解是很有帮助的。考虑图 7.4 中画出的数组 data1 和 data2 的数值。如果我们尝试画一条水平线通过每个图中的数值中点，这条线将近似等于 3.0。

因此两个数组的平均值近似相等，都是 3.0。但是，很明显这两组数据之间有一些特征上的差别。在 data2 中的数据与其平均值间的差异更大一些，或者说离均值更远一些。一组值的方差（variance）被定义为这组值与其平均值的差的平方和的均值；标准差（standard

deviation）则被定义为方差的平方根。因此，数组 data2 的方差和标准差要大于 data1 的方差和标准差。直观上看，方差（或者标准差）越大，数值围绕均值的波动就越大。

在数学上，方差使用 σ^2 表示，这里的 σ 是希腊字母 sigma。一组数据（我们假定数据存储在数组 x 中）的方差可以使用下面的公式计算：

$$\sigma^2 = \frac{\sum_{k=0}^{n-1}(x_k-\mu)^2}{n-1} \tag{7.1}$$

这个公式初看起来有点吓人，但是如果仔细看它，它会变得很简单，其中 $x_k-\mu$ 是 x_k 和平均值之间的差。这个差值经过平方后使我们总得到一个正值。然后我们将所有数据的差值的平方相加，将这个和除以 $n-1$，可以近似得到一个平均值。方差的定义有两种形式：样本方差（sample variance）的分母为 $n-1$，总体方差（population variance）的分母为 n。大部分工程应用使用样本方差，如公式（7.1）所示。因此，公式（7.1）计算了数据与平均值差的平方和。标准差被定义为方差的平方根：

$$\sigma = \sqrt{\sigma^2} \tag{7.2}$$

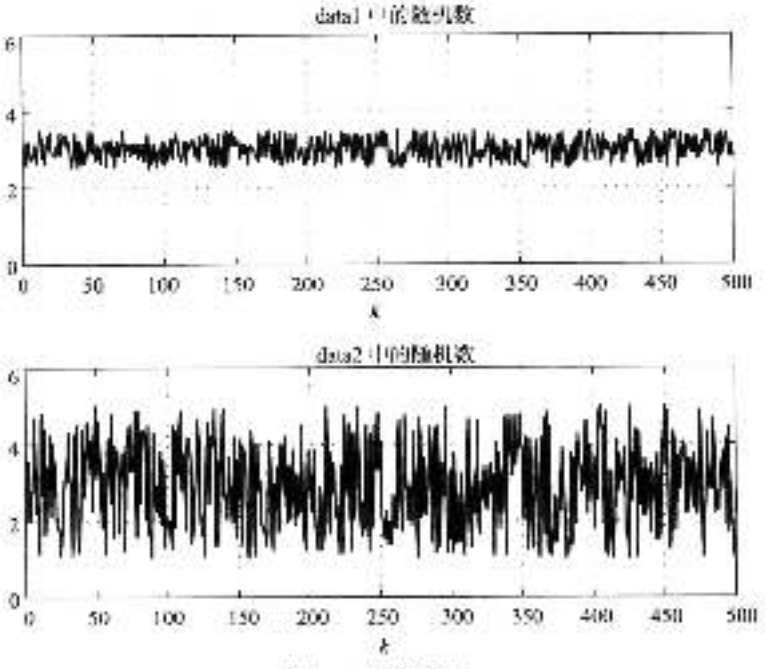


图 7.4 随机序列

方差和标准差都常用于工程数据分析，所以我们给出了计算这两个值的函数。注意计算标准差的函数调用了方差函数，而方差函数则调用了平均值函数，因此这些函数必须包含对应的函数原型语句。同时，注意数组中必须至少有两个值，否则方差函数将会尝试执行除数为 0 的除法。

```
//-----------------------------------------------------------/
//  This function returns the variance                       */
//  of an array with n elements.                             */

//  Function prototype:
double mean(const double x[], int n);
```

```
double variance(const double x[], int n) // Function header.
{
   // Declare objects.
   double sum(0), mu;

   // Determine variance.
   mu = mean(x,n);
   for (int k=0; k<n; ++k)
   {
      sum += (x[k] - mu)*(x[k] - mu);
   }

   // Return variance.
   return sum/(n-1);
}
/*-------------------------------------------------------------*/
/*  This function returns the standard deviation               */
/*  of an array with n elements.                               */

// Declare function prototypes.
double variance(const double x[], int n);

double std_dev(const double x[], int n) // Function header.
{

   // Return standard deviation.
   return sqrt(variance(x,n));
}
/*-------------------------------------------------------------*/
```

练习

假设数组 x 由下面的语句定义并初始化：

```
double x[]={12.3, 7.5, 6.3, 6.25, 9.01};
```

手工计算下面函数调用的返回值：

1. maxval(x,5)
2. median(x,5)
3. variance(x,5)
4. std_dev(x,5)
5. minval(x,4)
6. median(x,4)

7.3.3 自定义头文件

前面小节中所开发的函数在解决工程问题的过程中被频繁使用。为了方便使用它们，我们生成了一个包含这些函数原型的自定义头文件。这样就可以不在主函数中写出所有函数的函数原型，而可以使用一条包含自定义头文件的预处理指令。

名为 stat_lib.h 的自定义头文件包含了下列函数原型：

```
double maxval(const double x[], int n);
double minval(const double x[], int n);
double mean(const double x[], int n);
double median(const double x[], int n);
double variance(const double x[], int n);
double std_dev(const double x[], int n);
```

在 main 函数中包含的语句如下：

```
#include "stat_lib.h"
```

在下一节的程序中将对自定义头文件的用法进行说明。

除了使用 include 语句在程序中访问自定义头文件，程序还必须具有对包含统计函数的文件 stat_lib.cpp 具有访问权限。提供访问权限的细节与操作系统相关，并且可能涉及将文件名添加到完成编译、链接/载入等操作的操作系统命令中。

7.4 解决应用问题：语音信号分析

语音信号是一种可以通过麦克风转换成电信号的声学信号。电信号可以转换成一系列代表电信号值幅度的数字。这些数字可以存储到数据文件中用来供计算机程序对语音信号进行分析。假设我们对分析单词“zero”、“one”、…、“nine”的语音信号有兴趣，那么我们分析的目标就是从包含一个未知数字的语音数据文件中，以一定的方法分辨出正确的数字。

图 7.5 中包含了单词“zero”的语音图像。对类似这样的复杂信号的分析常常从计算前一小节所讨论的那些统计表征数开始。用于语音信号分析的其他测量值包括平均幅度（average magnitude）或者平均绝对值，其计算方法如下所示，这里 n 为数据集的数目：

$$\text{平均幅度} = \frac{\sum_{0}^{n} |x_i|}{n} \tag{7.3}$$

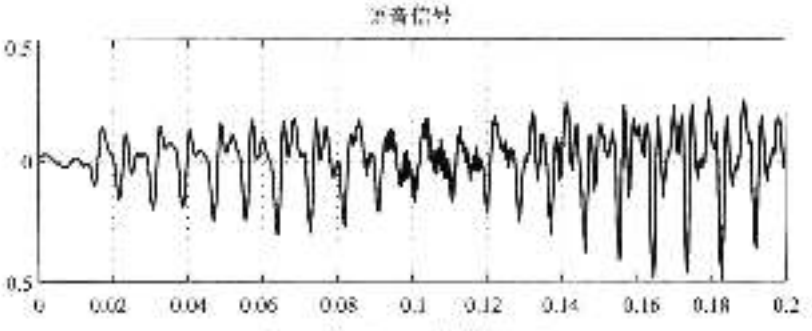


图 7.5 单词 zero 的语音表示

在语音分析中使用的另一个度量值是信号的平均强度，这里使用的是数据值平方的平均值：

$$\text{平均强度} = \frac{\sum_{0}^{n} x_i^2}{n} \tag{7.4}$$

在语音信号中零交叉的数目也是一个有用的统计量。这个统计量的值是语音信号中由正到负或者由负到正的转换次数；从非零值到零值的转换不是零交叉。

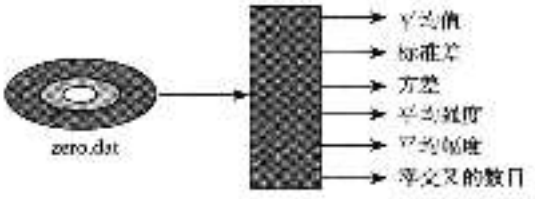
编写一个程序从名为 zero.dat 的数据文件中读取一个语音信号。这个文件中包含了表示单词“zero”语音信号的数值。文件中的每行都包含一个值，这个值表示由麦克风每隔 0.0002 秒所采集的信号值，所以 5000 个测量值代表 1 秒的数据。数据文件中只包含合法数据，不包含特殊的开头或结尾行；文件中最多包含 2500 个值。根据文件中的内容计算并打印出下列统计值：平均值、标准差、方差、平均强度、平均幅度、零交叉的数目。

1. 问题描述

计算并打印出一个语音信号波形的下列统计值：平均值、标准差、方差、平均强度、平均幅度、零交叉的数目。

2. 输入/输出描述

I/O 图表明输入为数据文件，输出为统计值。



3. 用例

作为用例，假定文件中包含下列数值：

2.5 8.2 −1.1 −0.2 1.5

使用计算器，我们可以计算出下面的值：

$$\text{平均值} = \frac{2.5+8.2-1.1-0.2+1.5}{5} = 2.18$$

$$\text{方差} = [(2.5-\mu)^2+(8.2-\mu)^2+(-1.1-\mu)^2+(-0.2-\mu)^2+(1.5-\mu)^2]/4 = 13.307$$

$$\text{标准差} = \sqrt{13.307} = 3.648$$

$$\text{平均强度} = \frac{[(2.5)^2+(8.2)^2+(-1.1)^2+(-0.2)^2+(1.5)^2]}{5} = 15.398$$

$$\text{平均幅度} = \frac{[|2.5|+|8.2|+|-1.1|+|-0.2|+|1.5|]}{5} = 2.7$$

零交叉数目 = 2

4. 算法设计

我们首先给出分解提纲，它将解决方案分为了一系列顺序的步骤：

分解提纲

1）对语音信号读入数组；

2）计算并打印统计量。

步骤 1 包括读取数据文件，并确定数据点的数目。步骤 2 包括计算和打印统计量 k，在可能的情况下使用已经开发的函数。main 函数中需要的其他统计函数的伪代码如下所示；图 5.1 中的结构图描述了 main 函数调用若干自定义函数的示例。

细化的伪代码

```
main:  read speech signal from data file and
       determine the number of points, n
       compute and print mean
       compute and print standard deviation
       compute and print variance
       compute and print average power
       compute and print average magnitude
       compute and print zero crossings
Additional Functions
ave_power(x, n):
       set sum to zero
       set k to zero
       while k <= n - 1
        add x[k]^2 to sum
```

```
        increment k by 1
      return sum/n
ave_magn(x, n):
      set sum to zero
      set k to zero
      while k <= n - 1
       add   x[k] | to sum
       increment k by 1
      return sum/n
crossings(x, n):
      set count to zero
      set k to zero
      while k <= n-2
       if x[k]*x[k+1] < 0
         increment count by 1
       increment k by 1
      return count
```

注意在有n个数据点的集合中，零交叉的可能数目为$n-1$，因为每个交叉是由一对值确定的。因此，最后一对测试的值的偏移量为$n-2$和$n-1$。伪代码中的步骤已经足够详细，可以转换成C++语句：

```
/*                                                                  */
/*  Program chapter7_6                                              */
/*                                                                  */
/*  This program computes a set of statistical                      */
/*  measurements from a speech signal.                              */

#include<iostream> //Required for cin, cout.
#include<fstream>  //Required for ifstream.
#include<string>  //Required for string
#include<cmath>  //Required for abs()
#include "stat_lib.h" //Required for mean(), variance(), std dev()
using namespace std;

//  Declare function prototypes and define constants.
double ave_power(double x[], int n);
double ave_magn(double x[], int n);

int crossings(double x[], int n);

int main()
{
   //  Declare objects.
   const int MAXIMUM = 2500;
   int npts(0);
   double speech[MAXIMUM];
   string filename;
   ifstream file_1;

  //  Prompt user for file name and
  //  open file
  cout << "Enter filename ";
  cin >> filename;
  file_1.open(filename.c_str());
  if( file_1.fail() )
  {
       cout << "error opening file " << filename
            << endl;
       return 0;
  }
```

```
   // Read information from a data file.
      while (npts <= MAXIMUM-1 && file_1 >> speech[npts])
      {
        npts++;
      } //end while

   //  Compute and print statistics.
      cout << "Digit Statistics \n";
      cout << "\tmean: " << mean(speech,npts) << endl;
      cout << "\tstandard deviation: "
           << std_dev(speech,npts) << endl;
      cout << "\tvariance: " << variance(speech,npts)
           << endl;
      cout << "\taverage power: " << ave_power(speech,npts)
           << endl;
      cout << "\taverage magnitude: "
           << ave_magn(speech,npts) << endl;
      cout << "\tzero crossings: " << crossings(speech,npts)
           << endl;

   //  Close file and exit program.
   file_1.close();
   return 0;
}
/*-------------------------------------------------------------*/
/*  This function returns the average power                    */
/*  of an array x with n elements.                             */

double ave_power(double x[], int n)
{
   //  Declare and initialize objects.
   double sum(0);

   //  Determine average power.
   for (int k=0; k<=n-1; ++k)
   {
      sum += x[k]*x[k];
   }

   //  Return average power.
   return sum/n;
}
/*-------------------------------------------------------------*/
/*  This function returns the average magnitude                */
/*  of an array x with n values.                               */

double ave_magn(double x[], int n)
{
   //  Declare and initialize objects.
   double sum(0);

   //  Determine average magnitude.
   for (int k=0; k<=n-1; ++k)
   {
      sum += abs(x[k]);
   }

   //  Return average magnitude.
   return sum/n;
}
/*-------------------------------------------------------------*/
```

```
/*  This function returns a count of the number                    */
/*  of zero crossings in an array x with n values.                 */

int crossings(double x[], int n)
{
   // Declare and initialize objects.
   int count(0);

   // Determine number of zero crossings.
   for (int k=0; k<=n-2; ++k)
   {
      if (x[k]*x[k-1] < 0)
        count++;
   }

   // Return number of zero crossings.
   return count;
}
/*----------------------------------------------------------------*/
```

5. 测试

该程序需要访问前一节中开发的头文件 stat_lib.h 和文件 stat_lib.cpp。下面的值是使用文件 zero.dat 为语音 "zero" 计算而得的值：

```
Digit Statistics
        mean: 0.002931
        standard deviation: 0.121763
        variance: 0.014826
        average power: 0.014820
        average magnitude: 0.089753
        zero crossings: 106
```

修改

1. 使用教师资源中的文件 two_a.dat、two_b.dat、two_c.dat 来运行程序。这些语音都是单词 "two" 的讲音，但是是由不同的人所说的。
2. 比较问题 1 中使用 3 个文件时的不同输出。输出结果说明了设计语音识别系统时说话者不同所面临的一些困难。

7.5 排序和搜索算法

对一组数据进行排序（sorting）是数据分析中另一种常见的操作。本书中给出了许多不同的排序算法，其中的原因之一就是虽然有如此多的排序算法，但不存在一种 "最好" 的算法。如果数据本身已经接近正确的顺序，那么有些算法会更快一些，但是当数据的排序是随机或者反序时，这些算法的效率会很低。因此，要选择对于特定应用而言最好的排序算法，你需要知道有关原始数据顺序的一些信息。在本书中我们只给出两种算法，而不试图对排序算法进行一个完整的讨论。我们鼓励你参考其他书籍，以对排序算法的完整讨论进行了解。本节我们讨论选择排序，这种算法易于理解且易于编码。

7.5.1 选择排序

选择排序（selection sort）算法从找出最小值的位置开始，并将最小值与数组的第一个

值进行交换。然后算法从第二个元素开始，寻找新的最小值，并将最小值与第二个元素交换。这个过程一直重复，直到倒数第二个元素为止，倒数第二个元素将与最后一个元素进行比较，如果它们的顺序不对，就将它们的值交换。到这里，整个数组的值都是升序排列了。这个过程可以通过下面数字的顺序重排序来说明：

原始顺序：

5	3	12	8	1	9

将最小值与第一个位置上的值进行交换：

1	3	12	8	5	9

将下一个最小值与第二个位置上的值进行交换：

1	3	12	8	5	9

将下一个最小值与第三个位置上的值进行交换：

1	3	5	8	12	9

将下一个最小值与第四个位置上的值进行交换：

1	3	5	8	12	9

将下一个最小值与第五个位置上的值进行交换：

1	3	5	8	9	12

数组的值现在就是升序排列了：

1	3	5	8	9	12

这些步骤在下面的函数中比较短，但使用列子中的数据对函数进行验证仍然不失为一个好主意。关注循环中下标 k、m 和 j 的变化。同时也要注意交换两个对象的值用了三步（而不是两步）。因为函数不需要返回值，所以返回类型为 void。

```
/*-------------------------------------------------------*/
/*  This function sorts an array with n elements         */
/*  into ascending order                                 */

void sort(double x[], int n)
{
   //  Declare objects.
   int m;
   double hold;

   //  Implement selection sort algorithm.
   for (int k=0; k<=n-2; ++k)
   {
      // Find position of smallest value in array
      // beginning at k
      m = k;
      for (int j=k+1; j<n; ++j)

         if (x[j] < x[m])
            m = j;

      // Exchange smallest value with value at k
      hold = x[m];
      x[m] = x[k];
```

```
        x[k] = hold;
    }

    // Void return.
    return;
}                                                    */
```

为了使该函数变成对数组中的值进行降序排列，在内层循环中应当搜索最大值而不是最小值。

为了调用该函数，应当使用下面的函数原型语句：

```
void sort(double x[], int n);
```

值得注意的是，该函数修改了源数组。为了保持源数组的顺序，在该函数执行前应当将源数组复制到另一个数组中，这样数组的原始顺序和排序后的顺序就都可知了。

修改

1. 编写一个 main 函数，它初始化一个数组，并调用 sort 函数，然后打印出排序后的数组值。
2. 修改 sort 函数，使它以降序对数值进行排列而不是升序排列。使用问题 1 中的程序对函数进行测试。

7.5.2 搜索算法

对数组进行的另一个常见操作就是在数组中搜索特定的值。我们可能希望知道数组中是否有特定的值，这个值出现了多少次，或者它第一次出现在数组中的位置。所有这些形式的搜索都只确定单个值，因此适合用函数实现。下面我们将开发几个函数用于数组的搜索，以后当你在程序中需要进行搜索时，可以使用这些函数中的某个来完成。当然，可能需要对它进行少量修改或者不需修改。

搜索算法可以分为两类：一类针对无序列表，一类针对有序列表。

7.5.3 无序列表

我们首先考虑对无序列表的搜索；因此假定元素不需要排序为升序或者其他可以帮助我们搜索数组的顺序。对于无序数组的搜索算法就是一种简单的顺序搜索（sequential search）：检查第一个元素，检查第二个元素，以此类推。我们可以以几种方式实现这个函数。我们可以将函数设计成一个返回整数的函数，当要求的值在数组中时返回该值在数组中的位置。当不在数组中时，返回 -1。我们还可以将函数设计成一个布尔函数，当元素在数组中时返回真（1），不在数组中时返回假（0）。所有这些设计方式都可以得到有效的函数，我们可以想象使用每种形式的程序。这里，我们给出一个函数，当要求的值在无序数组中时返回该值的位置，不存在则返回 -1。

```
int search(const int A[], int n, int value)
/* This function returns the position of value in array A.  */
/*  Returns -1 if value is not found in A.                  */
/*  Function assumes array A is unordered.                  */
{
   int index=0;
   while (index < n && A[index]!=value)
   {
          ++index;
```

```
    }
    if(index < n && A[index] == value)
            return(index);
    else
            return(-1);
}
```

7.5.4 有序列表

现在我们考虑对一个有序或者经过排序的列表进行搜索。考虑下面一组有序的值。假定我们要搜索的值是 25：

-7
2
14
38
52
77
105

当我们搜索到值 38 时，就知道 25 不在列表中，因为我们知道列表是按照升序排列的。因此，我们不需要像搜索无序列表时一样搜索整个列表；我们只需要搜索到我们要找的值所当在的那个位置。如果列表是升序排列，当当前的值比我们要找的值大时，搜索停止；如果列表为降序排列，当当前的值比我们要找的值小时，搜索停止。现在我们给出一个完成了对有序列表进行顺序搜索（sequential search on an ordered list）的函数。如果要求的值在顺序数组中，则函数返回该值的位置，否则返回 -1。

```
/*-------------------------------------------------------------*/
/*  This function returns the position of value in array A.    */
/*  Returns -1 if value is not found in A.                     */
/*  Function assumes:                                          */
/*    the array A has n elements in ascending order.           */

int search(const int A[], int n, int value)
{
    int index(0);
    while (index < n && A[index] < value)
    {
            ++index;
    }
    if(index < n && A[index] == value)
            return(index);
    else
            return(-1);
}
/*-------------------------------------------------------------*/
```

另一种应用广泛且效率更高的有序列表搜索算法是二分搜索（binary search）。二分搜索算法有时也称作分治算法，因为它从列表的中间开始，并确定要搜索的值是在列表的上半部分还是下半部分。如果值属于上半部分，算法则从上半部分的中间开始检查并按此过程重复，这样每次将搜索空间减半，直到找到目标值，或者确定目标值不在列表中为止。下面的函数实现了二分搜索：

```
/*-----------------------------------------------------------------*/
/*  This function returns the position of value in array list.     */
/*  Returns a value of -1 if value is not found in list.           */
/*  Function assumes:                                              */
/*    the array list has n elements in ascending order.            */

int bSearch(const int list[], int n, int value)
{
   int top(0), bottom(n-1), mid;

   while(top<=bottom)
   {
      //  Determine mid point of list.
      mid = (top - bottom)/2;

      //  Value is found.
      if(list[mid] == value)
          return mid;

      //  Look for value in top half of list.
      else if(list[mid] > value)
          bottom mid-1;

      //  Look for value in bottom half of list.
      else
          top mid+1;
   }
   //  Value was not found in the list.
   return -1;
}
```

修改

1. 修改针对有序列表的顺序搜索，使其返回指定的值在有序列表中出现的次数。
2. 修改二分搜索函数，使其能够正确搜索一个降序排列的列表，而不再针对升序列表。编写一个程序测试你的函数。

7.6 解决应用问题：海啸预警系统

海啸是在大量的水移动后出现的一系列的水波。水下的爆炸，如火山爆发和地震，都有可能导致大量水的移动，进而导致涌向低洼地区的海啸。太平洋海啸预警中心（Pacific Tsunami Warning Center，PTWC）使用地震数据和实时海洋数据来计算可能的威胁并在监测到海啸时发出警报。海洋数据收集自测量仪器和浮标，用于监视和确定海啸的形成。美国国家海洋和大洋局（National Oceanic and Atmospheric Administration，NOAA）国家航标数据中心（National Data Buoy Center，NDBC）开发和维护了一个用于浮标数据收集的网络和海岸监测站。生成的实时数据文件包含最近45天记录的数据。通过对这些文件中的原始数据进行取样和分析以生成海洋数据，包括风向、风速、有效波高、平均波周期、主波周期（dominant wave period）。为了支持该网络和其他数据收集项目，NDBC雇佣了大量的专业人士，包括工程师、气象学家、海洋学家和计算机科学家。

在本节的应用中，我们使用记录的数据计算取样时长为20分钟内的有效波高（Wave Height，WVHT）。有效波高以米计量，被定义为20分钟取样时间内所有波高中最高的1/3的平均值。编写一个程序从包含20分钟取样时间内波高测量值的数据文件中读取数据，并计算出WVHT。程序应该打印出WVHT，并打印出20分钟取样时段的起始时间和结束时

间。数据文件由一个开头行和20行数据组成。开头行具有下面的形式：

```
#YY MM DD HH MM WH(m)
```

数据文件中随后的行包含了时间和波高测量值，形式如下：

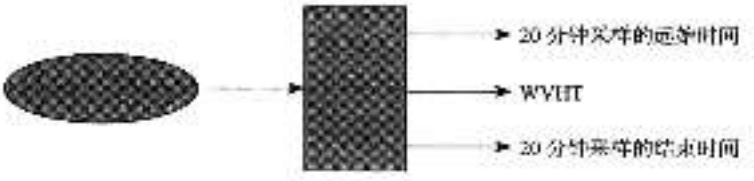
```
year(int) month(int) day(int) hour(int) minute(int)
    wave height(double)
```

1. 问题描述

计算20分钟取样时段内的有效波高（WVHT）。

2. 输入/输出描述

下面的I/O示意图给出了程序的输入和输出，输入是数据文件，输出为WVHT和时间信息。



3. 用例

假定数据文件JulySample.dat中包含下面的数据：

#YY	MM	DD	HH	MM	WH（m）
2011	07	11	03	28	1.23
2011	07	11	03	29	1.27
2011	07	11	03	30	1.29
2011	07	11	03	31	1.51
2011	07	11	03	32	1.72
2011	07	11	03	33	1.85
2011	07	11	03	34	2.01
2011	07	11	03	35	2.12
2011	07	11	03	36	1.92
2011	07	11	03	37	1.71
2011	07	11	03	38	1.32
2011	07	11	03	39	1.26
2011	07	11	03	40	1.21
2011	07	11	03	41	1.02
2011	07	11	03	42	1.12
2011	07	11	03	43	1.24
2011	07	11	03	44	1.27
2011	07	11	03	45	1.29
2011	07	11	03	46	1.33
2011	07	11	03	47	1.73

通过观察，我们可以知道最大的 6 个值为：

2.12 2.01 1.92 1.85 1.73 1.72

对应的输出如下：

起始时间：

2011 7 11 3 28

结束时间：

2011 7 11 3 47

WVHT 为 1.89167

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。为了找出有效波高，我们必须得到记录波高中最大的前 1/3 的平均值。我们的输入文件包含 20 个波高测量值，因此我们需要计算出最大的 6（20/3）个波高的平均值。因为数据文件是按照时间排序的，不是波高，我们的方法将需要读取数据并将波高存储在数组中。在读取数据时，我们将把波高数据按照降序排列，这种排序方法称为插入排序（insertion sort）。在所有的数据都被读取后，我们将计算出在排序好的数组中的前 6 个值的平均值。

分解提纲

1）读取数据的开头行和第一行数据；

2）打印开始时间；

3）读取波高数据并入数组并以降序排列；

4）计算 WVHT；

5）打印结束时间和 WVHT。

我们将把计算 WVHT 的步骤写在一个函数中。

细化的伪代码

```
main: if file cannot be opened
          print error message
          exit
      read header and first line of data
      print starting time

   //insertion sort
   set i to 1
   while(i < 20)
     read next line of data
     set pos to 0
     while pos < i  and  wave height < array[pos]
        increment pos by 1
     if pos equal to i assign wave height to end of array
        array[i] = wave height
     else insert wave height to maintain order
        set k to i
        while k > pos move values down to make room
            array[k] = array[k-1]
            decrement k by one
        array[pos] = wave height
        increment i by 1
   //end insertion sort
   calculate WVHT
   print ending time and WVHT
```

```
WVET(array,count)
    set i and sum to 0
    while i < count
       add array[i] to sum
       increment i by 1
    return sum/count
```

伪代码中的步骤已经足够详细，可以转换成 C++ 语句。

```
/*-------------------------------------------------------------*/
/* Program chapter7_1                                          */
/* This program inputs wave height data from an                */
/* input file then calculates the significant wave             */
/* height(WVET).                                               */

#include<iostream>  //Required for cin, cout, cerr.
#include<fstream>   //Required for ifstream.
#include<string>    //Required for getline().
#include<iomanip>   //Required for setw()
using namespace std;

//Function Prototypes
double average(double[], int);

int main()
{
  //Declare and initialize objects
  const int SAMPLE_SIZE = 20;
  double waveHeights[SAMPLE_SIZE], WVET, newVal;
  int year, month, day, hour, minute;
  string filename, header;
  ifstream fin;

  //Get filename and open file
  cout << " Enter name of input file: ";
  cin  >> filename;
  fin.open(filename.c_str());
  if(fin.fail())
  {
    cerr << " Could not open the file "  << filename
         << "  Goodbye."  << endl;
    exit(1);
  }
  //Read header from input file
  getline(fin,header);

  //Read first line of input data
  int i = 0;
  fin >> year >> month >> day >> hour
      >> minute >> waveHeights[i];

  //Echo header
  cout << header << endl;

//Print starting date and time.
  cout << " Starting time: "  << endl << year
       << setw(3) << month << setw(3) << day
       << setw(3) << hour << setw(3) << minute << endl;

  //Read remaining lines of input
  //Order waveHeight in descending order
  int pos;
  for(i=1; i<SAMPLE_SIZE; ++i)
```

```
    {
      fin >> year >> month >> day >> hour
          >> minute >> newVal;
      //find ordered position
      pos = 0;  //start at top
      while(pos < i && newVal < waveHeights[pos])
      {
        ++pos;
      }
      if(pos == i)
      {
        //newVal belongs at end of array
        waveHeights[i] = newVal;
      }
      else
      {
        //insert newVal at midpoint in array
        //Move values down to make room
        for(int k=i; k>pos; --k)
        {
          waveHeights[k] = waveHeights[k-1];
        }
        //Assign new value to array
        waveHeights[pos] = newVal;
      }
    }//end for

    //Calculate the WVHT
    //WVHT is defined as the average of the
    //the highest one-third of all wave heights.
    //Average top 1/3 of array elements.
    int top3rd = SAMPLE_SIZE/3;
    WVHT = average(waveHeights, top3rd);

    //Print ending date and time.
    cout << " ending time: "  << endl << year
         << setw(3) << month << setw(3) << day
         << setw(3) << hour << setw(3) << minute << endl;
    cout << " WVHT is "  << WVHT << endl;

    fin.close();
    return 0;
}
/*-------------------------------------------------------*/
/* This function returns the average of the first size   */
/* elements in array                                     */

double average(double array[], int size)
{
  double sum = 0.0;
  for(int i=0; i<size; ++i)
  {
    sum += array[i];
  }
  sum = sum/size;
  return sum;
}
/*-------------------------------------------------------*/
```

5. 测试

如果我们使用用例中的文件进行测试，输出如下：

```
Enter name of input file: JulySample.dat
WFT  MM DD HH MM WH(m)
Starting time:
2011  7 11  3 28
ending time:
2011  7 11  3 47
WVHT is 1.89167
```

修改

这些问题与本节所开发的计算 WVHT 的程序有关。

1. 修改程序，使其直接读取所有的波高数据并将其无序地存入数组，然后调用本节开发的函数

```
void sort(double x[], int n);
```

在计算 WVHT 之前对数据进行排序。

2. 修改程序，将抽样尺寸从 20 改为 200。
3. 修改计算 WVHT 的程序，在程序中不对数组进行排序。提示：读取并将最开始的三个值存入一个新数组，然后将其他的值与新数组中最小的值进行比较。

7.7 字符串

字符数组是一个元素被当做字符存储的数组。字符串（character string）可以用一个末尾为 null 字符（'\0'）的字符数组表示，其中 null 字符的 ASCII 值为 0。这种形式的字符串称作 C 风格字符串（C-style string），因为它源自 C 程序语言，并仍为 C++ 所支持。

字符串还可以使用第 2 章中介绍的 string 类来表示，string 类将在 7.9 节中讨论。

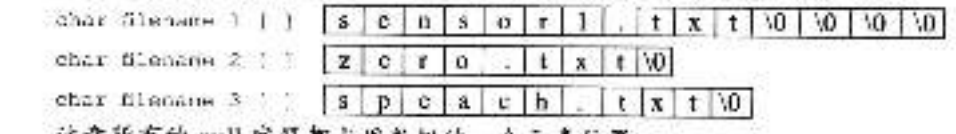
7.7.1 C 风格字符串定义和 I/O

字符串常量被双引号围绕，如 "sensor1.dat"、"r" 和 "15762"。字符串对象可以使用字符串常量或字符常量来定义和初始化，如下面的语句所示：

```
// Define string objects using string constants.
char filename1[15] = "sensor1.txt";
char filename2[] = "zero.txt";

// Define string object using character constants.
char filename3[] = {'s','p','a','a','c','h',
                    '.','t','x','t','\0'};
```

上面的每个语句都定义了一个 C 风格字符串对象，其内存快照如下所示：

char filename 1 [ ]	s	c	n	s	o	r	1	.	t	x	t	\0	\0	\0	\0
char filename 2 [ ]	z	c	r	o	.	t	x	t	\0						
char filename 3 [ ]	s	p	c	a	c	h	.	t	x	t	\0				



注意所有的 null 字符都占用数组的一个元素位置。

下面的语句使用输入操作符从键盘读取两个字符串，使用输出操作符将字符串打印到屏幕：

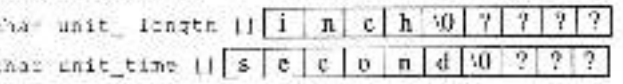
```
//  Declare objects.
char unit_length[10], unit_time[10];
```

```
...
// Read data into string.
cin >> unit_length >> unit_time;
cout << unit_length << ' ' << unit_time << endl;
...
```

假定从键盘输入了下面的数据：

```
feet second
```

输入操作符将输入一个字符串并将 null 字符放入字符串的结尾。如下面的内存快照所示：



但是输入操作符并不检查数组的最大尺寸。如果输入数据的字符数比字符数组的最大尺寸要大，那么输入操作符在赋值时将超出数组的边界，这时将发生错误。

回忆一下，输入操作符是忽略所有空白的。如果要求将空白作为字符串的一部分，则可以使用函数 getline() 来读取字符串，在下一个例子中对此作了说明，例子要求从一个 ASCII 映像文件头部中读取注释行，并将注释行打印到屏幕上。我们的例子使用 peek() 函数来查看输入流中的下一个字符。函数 peek() 是 istream 类的一个成员函数，它可以被任何 istream 对象调用。函数 peek() 返回输入流中下一个字符的值，但是并不将该字符从输入流中移除：

```
...
// Declare objects
ifstream image("ic.pgm");
char comment_line[100];
...
// Read and print a comment line from a data file.
// Comment lines begin with a #.
while(image.peek() == '#')
{
        image.getline(comment_line, 100);
        cout << comment_line << endl;
}
...
```

函数 getline() 是 istream 类的一个成员函数，它可以被任何 istream 对象调用。第一个参数是字符数组的名字。第二个参数是数组的最大长度。函数 getline() 将读取并存储字符，直到遇到换行字符或者已经读取到最大长度的字符数。函数将把 null 字符放在字符串末尾作为结束。如果遇到换行字符，getline() 函数将读取它并将它丢弃掉。函数 getline() 中可以使用的第 3 个参数用于指明一个除了换行符外的字符来表示输入的结束。

练习

假设从键盘输入了下面的内容：

```
The nice, knift and Scurry,
had only simple brains.
```

给出下面的程序片段生成的输出（假设每个片段都使用整个输入流）：

1. `char cstring1[10], cstring2[] = "Cheese";`
   `cin >> cstring1;`
   `cout << cstring1 << ' ' << cstring2 << endl;`
2. `char cstring1[10], cstring2[] = "Cheese";`
   `cin.getline(cstring1,10);`
   `cout << cstring1 << cstring2 << endl;`
3. `char cstring1[30], cstring2[50];`

```
cin.getline(cstring1,30);
cin.getline(cstring2,30);
cout << cstring1 << cstring2 << endl;
```

7.7.2 字符串函数

标准C++库中包含了大量处理C风格字符串的数值函数，如下面列出的这些。在程序中使用这些函数时必须包含头文件cstring：

strlen(s)：返回字符串s的长度。

strcpy(s,t)：将字符串t复制到字符串s中。

strcat(s,t)：将字符串t连接到字符串s的末尾。

strcmp(s,t)：对字符串s和t作逐字符的ASCII值的比较。如果s<t，则返回一个负数。如果s等于t，则返回0。如果s>t，则返回一个正数。

在cstring库文件中包含了相关函数的完整列表，可以在附录A中看到。为了说明这些函数的用法，现在我们给出一个简单的例子。

```
/*-------------------------------------------------------------*/
/*  Program chapter7_8                                         */
/*  This program illustrates the use of several                */
/*  C style string functions.                                  */

#include<iostream> //Required for cout
#include<cstring>  //Required for strlen(), strcmp(), strcpy()
                   //                    strcat().
int main()
{
   //  Declare and initialize objects.
   char strg1[]="Engineering Problem Solving: ";
   char strg2[]="Object Based Approach", strg3[75] = "";

   //  Print the length of each string.
   cout << "String lengths: " << strlen(strg1) << ' '
        << strlen(strg2) << ' ' << strlen(strg3) << endl;

   //  Swap strings if strg1 is larger than strg2
   if(strcmp(strg1,strg2) > 0)
   {
     strcpy(strg3,strg2);
     strcpy(strg2,strg1);
     strcpy(strg1,strg3);
   }

   //  Combine two strings into one.
   strcpy(strg3,strg1);
   strcat(strg3,strg2);
   cout << "strg3: " << strg3 << endl;
   cout << "strg3 length: " << strlen(strg3) << endl;
   return 0;
}
/*-------------------------------------------------------------*/
```

程序的输出如下：

```
String lengths: 29 21 0
strg3: Engineering Problem Solving: Object Based Approach
strg3 length: 50
```

7.8 string 类

string 类为 C 风格字符串提供了基于对象的选择。下面是 string 类中一些常用的成员函数：

size()：返回调用字符串的长度。

empty()：如果调用字符串不包含任何字符，则返回 true，否则返回 false。

substr(int start, int len)：返回从调用字符串 start 位置开始的长度为 len 的子字符串。

c_str()：返回等价的 C 风格字符串。

在 string 类中重载了大量数值型操作符以支持 string 对象的使用。赋值操作符，以及关系操作符 <、>、<=、>=、== 和操作符 +、+=，都可以用于字符串对象。二元操作符 "+" 将两个字符串连接起来，二元操作符 "+=" 将一个字符串连接到另一个字符串末尾。为了说明这些操作符和函数的用法，我们重写了程序 program7_8，在其中增加了一些语句，并使用了 string 类来替代 C 风格字符串（记住要使用 string 类，必须包含头文件 string）：

```
/*-------------------------------------------------------------*/
/*  Program chapter7_9                                         */
/*  This program illustrates the use of several                */
/*  operators and functions supported by the string class.     */

#include<iostream> //Required for cout
#include<string>   //Required for string, size()

int main()
{
   // Declare and initialize string objects.
   string strg1 = "Engineering Problem Solving: ";
   string strg2 = "Object Based Approach", strg3;

   // Print the length of each string.
   cout << "String lengths: " << strg1.size() << ' '
        << strg2.size() << ' ' << strg3.size() << endl;

   // Swap strings if strg1 is larger than strg2
   if(strg1 > strg2)
   {
     strg3 = strg2;
     strg2 = strg1;
     strg1 = strg3;
   }

   // Append a string.
   strg2 += " Using C++";

   // Concatenate two strings.
   strg3 = strg1 + strg2;

   cout << "strg3: " << strg3 << endl;
   cout << "strg3 length: " << strg3.size() << endl;
   return 0;
}
/*-------------------------------------------------------------*/
```

程序输出如下：

```
String lengths: 29 21 0
strg3: Engineering Problem Solving: Object Based Approach Using C++
strg3 length: 60
```

修改

1. 修改 sort() 函数，使它可以对一个字符串数组按照字母顺序排序。使用 string 类，编写一个程序测试这个函数。
2. 修改 bSearch() 函数，使它可以搜索一个有序的字符串列表。使用 string 类，编写一个程序测试这个函数。

7.9 vector 类

vector 类是包含在 C++ 标准模板库（Standard Template Library，STL）中的一个预定义类型，它提供数组的一个通用实现。使用 vector 类有几个优点：当使用数组时，我们必须注意不要超过数组的边界，并且我们必须跟踪数组的实际大小；vector 类中包含一个用于返回 vector 大小的方法，该返回 vector 容量（capacity）的方法，以及一个可以动态增加或减少 vector 对象容量的方法，这意味着当程序执行时 vector 的容量是可变的。

vector 的容量是指已经分配给它的内存，而 vector 的大小则是当前被使用的元素数目。由于 vector 的动态特性，这些数目并不总是相同的。表 7.3 中列出了在 vector 类中定义的一些常用方法。这些方法被 vector 类型的对象调用，因此方法的描述与方法对调用对象的作用有关。

表 7.3 定义在 vector 类中的方法

方法名	方法描述	方法名	方法描述
back()	返回最后一个元素的值	insert()	插入元素
begin()	返回指向第一个元素的迭代器	pop_back()	删除最后一个元素
capacity()	返回容量	push_back()	将元素添加到末尾
empty()	如果大小为 0 返回 true，否则返回 false	resize()	改变容量
end()	返回指向最后一个元素之后的迭代器	size()	返回大小
erase()	删除元素		

你会注意到这些方法中有两个方法返回了迭代器。迭代器和指针将在第 9 章讨论。

我们可以像下面这样在一个类型声明语句中定义一个 vector 类的实例：

```
vector<char> v1;          //Define vector of type char, capacity 0.
vector<int> v2(10);       //Define v2 with capacity for 10 integers.
vector<string> v3(n);     //Define v3 with capacity for n strings.
vector<double> v4(n,1.0); //Define v4 with capacity for n doubles.
                          //Initialize each element to 1.0.
```

注意指出每个 vector 将要存储的数据类型的语法声明。需要使用

```
vector<data type>
```

因为 vector 被定义为一个类模板。在 STL 中定义的模板通过实现无关特定数据类型的思想来支持泛型编程。例如，vector 的概念，或者称为数组，就是一个存储顺序块数据的容器。在 vector 的末尾对数据进行添加、删除很方便，vector 像数组一样支持使用偏移量对数据进行随机访问。因为这些思想是独立于数据的，所以在定义一个 vector 模板后，编译器将在 vector 结合特定类型声明后生成一个具体的模板实例。

下面的例子说明了 vector 类的用法以及 vector 的大小和容量之间的区别。注意，我们必须包含头文件 vector。因为 vector 是标准 C++ 库的一部分，所以将会自动被链接。

```
/*.....                                                    */
/* Program chapter7_10                                     */

#include<iostream>  //Required for cout
#include<vector>    //Required for vector
using namespace std;

int main()
{
 //Declare and initialize objects.
   vector<int> v(5);

//Print the capacity and the size.
  cout << "Capacity: " << v.capacity() << " Size: " << v.size()
       << endl;

// Assign values to v.
   for(int i=0; i<v.capacity(); i++)
   {
      v[i] = i;  //Random access
   }

//Print the capacity and the size.
  cout << "Capacity: " << v.capacity() << " Size: " << v.size()
       << endl;

//Add additional data to the end of v
  v.push_back(10);
  v.push_back(20);

//Print the capacity and the size.
  cout << "Capacity: " << v.capacity() << " Size: " << v.size()
       << endl;

   return 0;
}
/*.....*/
```

程序的一次示例运行输出如下：

```
Capacity: 5 Size: 5
Capacity: 5 Size: 5
Capacity: 10 Size: 7
```

为了完善和更新有关 STL 的信息，请访问网站：

http://www.sgi.com/tech/stl/table_of_contents.html

修改

1. 修改程序 chapter7_10，在 return 语句前增加语句打印出 vector v 中的每个元素，每个值占一行。
2. 修改程序 chapter7_10，替换掉下面的语句而使用偏移量将相同的值赋给 v 的末尾，不再使用 push_back() 方法。你的程序可以正常运行吗？解释原因。

```
//Add additional data to the end of v
 v.push_back(10);
 v.push_back(20);
```

参数传递

当函数定义中需要使用 vector 作为形参时，默认是按照值传递的。如果我们需要修改

vector 参数的值，则必须指定按照引用传递，如下面的程序所示。

```
/*-------------------------------------------------------------*/
/* Program chapter7_11                                         */
/* This program inputs a collection of points and              */
/* finds the Point that is closest to the origin               */
#include<iostream> //Required for cin, cout, cerr.
#include<fstream>  //Required for ifstream.
#include<string>   //Required for string.
#include<vector>   //Required for vector.
#include "Point.h" //Required for Point.
using namespace std;

// Function prototypes.
void readPointFile(istream& in,
                   vector<Point>& v); //Pass by reference
Point closeToOrigin(vector<Point> v);  //Pass by value.

int main()
{
 //Declare objects.
   Point p;
   string filename;
   ifstream file1;

  //Prompt user for file name and
  //open file.
  cout << "Enter filename ";
  cin >> filename;
  file1.open(filename.c_str());
  if( file1.fail() )
  {
        cerr << "error opening file " << filename << endl;
        exit(1);
  }

  //Build Point vector
  vector<Point> v;
  readPointFile(file1, v);

  //Find point closest to origin
  p = closeToOrigin(v);

  cout << "(" << p.getX() << "," << p.getY() << ")"
  << " is closest to the origin." << endl;
  return 0;
}
/*-------------------------------------------------------------*/
/*                                                             */
void readPointFile(istream& in,
                   vector<Point>& v)
{
  int npts;
  // Read number of data points.
     in >> npts;
     v.resize(npts);
     Point p;

  // Read Points and store in vector.
  // Points are formatted as x,y
     double x,y;
```

```
        char comma;
        for (int i=0; i<npts; ++i)
        {
            is >> x >> comma >> y;
            p.setX(x);
            p.setY(y);
            v[i] = p;
        }
    }
    /*-------------------------------------------------------------*/
    /*-------------------------------------------------------------*/
    /* This function returns the point                             */
    /* closest to the origin                                       */

    Point closeToOrigin(vector<Point> v)
    {
       Point p1, origin(0.0,0.0);
       int closest(0);  //offset of closest.
       for(int i=1; i<v.size(); ++i)
       {
         if(v[i]-origin < v[closest]-origin)
         {
           closest = i;
         }
       }
       return v[closest];
    }
    /*-------------------------------------------------------------*/
```

程序的一次示例运行得到的输出如下：

```
Enter filename pointdata.txt
(1.00,-0.50) is closest to the origin.
```

在C++ STL中定义的类支持在头文件algorithm中定义的一组顶层函数。这些函数完成排序、搜索和其他一些常见的任务。使用在algorithm中定义的函数可以节约问题解决方案的开发时间，并提高解决方案的效率。在下一节中，我们将开发一个问题解决方案，其中使用了在algorithm中定义的函数random_shuffle()。

修改

修改Point类，在其中添加一个print(ostream &) const方法。当使用语句"p.print(cout);"调用该函数时，输出结果应该类似程序chapter7_1的输出，即(X,Y)。

7.10 解决应用问题：概率计算

概率论在工程领域有很多应用，包括第6章讨论的设备可靠性、结构化分析和风险管理。风险管理是一个包括风险的识别、分析、沟通和处理的过程。在工程领域，风险被定义为：

风险 =（某个事件的概率）*（每个事件的损失）

本节我们将使用集合论中的公式计算一个事件的概率，并编写仿真来测试我们的结果。

实验和仿真是科学和工程中的主要范型（major paradigm）。仿真使用可以编制成软件的数学模型来更好地理解实验结果。许多物理实验在重复进行时并不会生成完全相同的结果。这种类型的实验称为非确定性的（nondeterministic），或者概率性（probabilistic）的实验。例如，如果实验是从一副扑克牌中抽出一张牌，并且进行了两次实验，那么结果很可能是不同的。但是也有可能在对随机的一副扑克牌进行两次连续抽取时得到相同的牌。

在两次独立的抽取中抽到相同牌的这样不大可能的事件中，没有大的损失出现，除了赌场将赌注压在了这个事件发生所造成的经济损失上。工程师设计了结构和新技术来提高生活的质量。当不大可能的事件发生时，损失可能是灾难性的。因此，准确计算事件发生概率的能力在风险分析中很重要。

在概率理论中，数值 $p(E)$ 表示事件 E 的概率（probability of event E），它被赋给一个事件。$p(E)$ 是事件 E 发生的概率。假定我们从一个有 52 张牌的队列中抽取两张牌，我们对这两张牌都是 King 的概率感兴趣，在这种情况下，事件 E 就是从一副扑克牌中抽取出两张 King 的这一事件。为了估计该事件的概率，我们可以进行从一副扑克牌中抽取 2 张牌的实验，抽取的总次数为 n，其中事件 E 发生的次数为 n_E。分数 n_E/n 就是 E 将发生的可能性的度量值。我们的直觉告诉我们，如果进行两次（$n=2$）实验，n_E 很有可能为 0。但如果我们进行 10 000（n=10 000）次实验，n_E 就很有可能大于 0。因此，当 n 变得很大时，$p(E)$ 可以用数 n_E/n 来表示。

为了估计从一副扑克牌中抽出 2 张 King 的概率，我们可以手工进行 n 次实验，统计事件发生的次数，或者我们可以编写仿真程序完成 n 次实验，并汇总结果。在每种情况下，我们都是统计一个概率实验的结果，并且结果是变化的。概率论使用基于集合论的公式来统计结果并计算概率。

假定我们有一个含有 n 个元素的集合 S，并选择一个数 r，使其满足 $1 \leq r \leq n$。那么每次从 S 中取出 r 个元素的可能组合数为：

$$\frac{n!}{r!(n-r)!}$$

注意从集合 S 中一次取出 r 个元素的组合数只取决于 r 和 n，而不取决于 S。这个数是从 n 个对象中一次取出 r 个（n choose r）的组合数，写作：

$${}_nC_r = \frac{n!}{r!(n-r)!}$$

我们可以考虑将实验的结果和发生的事件形成集合。由所有实验结果组成的集合称作实验的样本空间（sample space），由代表事件发生的结果形成的集合被称作事件空间（event space）。注意事件空间是样本空间的一个子集。一个集合是一个定义明确的、由不同元素形成的聚集，集合中元素的顺序并不重要。如果我们假定实验中所有的结果出现的可能性相等，而这一假设通常正确但很难证明，那么：

$$p(E) = \frac{\text{事件空间大小}}{\text{样本空间大小}}$$

我们现在将计算从 52 张牌中抽出 2 张 King 的概率。实验是从 52 张牌中抽出 2 张牌，因此，样本空间的大小为：

$${}_{52}C_2 = \frac{52!}{2!(52-2)!} = \frac{52(51)(50!)}{2!(50!)} = \frac{52(51)}{2} = 1326$$

事件则是从含有 4 张 King 的一副扑克牌中抽出 2 张 King，因此，事件空间大小为：

$${}_4C_2 = \frac{4!}{2!(4-2)!} = \frac{4(3)(2!)}{2!(2!)} = \frac{4(3)}{2} = 6$$

那么从 52 张牌中抽出 2 张 King 的概率为 $\frac{6}{1326} = 0.004\ 524\ 89$。因为我们的事件空间

大小为6，为了说明，我们列出了集合中的元素。

事件空间 = {{黑桃 King，红桃 King}，{黑桃 King，方块 King}，
{黑桃 King，梅花 King}，{红桃 King，方块 King}，
{红桃 King，梅花 King}，{方块 King，梅花 King}}

如同前一节所说的，我们可以使用计算机仿真对解析形式的结果进行验证。随着实验次数的增加，仿真的概率应当接近计算出的解析概率。

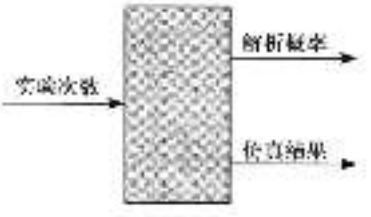
编写一个程序比较一个事件的解析概率和仿真的结果。该程序中的事件是从52张牌中抽出2张牌，且这两张均为红色，每张的点数都小于10。允许用户输入实验的次数以用于仿真。

1. 问题描述

比较从一副扑克牌中抽出2张牌的解析概率和仿真概率，其中抽出的两张牌均为红色且点数均小于10。

2. 输入/输出描述

I/O示意图表明输入值为仿真所要进行的实验次数，输出为解析概率和仿真结果。



3. 用例

在用例中，我们将通过确定事件空间的大小并用该值除以样本空间的大小，来计算解析概率。在一副52张的扑克牌中，包含8张不超过10点的红桃牌和8张不超过10点的方块牌，总计有16张不超过10点的红色牌。因此，事件空间大小为：

$$_{16}C_2 = \frac{16!}{2!(16-2)!} = \frac{16(15)(14!)}{2!(14!)} = 120$$

样本空间在本节早前已经计算过，为1 326；因此抽出两张点数不超过10点的红色牌的概率为 $\frac{120}{1326} = 0.090\,497\,74$。

为了手工完成实验，我们需要：

- 洗好一副牌。
- 从牌堆中抽取2张牌。
- 查看结果。
- 如果两张牌都是点数不超过10点的红色牌，则记录这次事件，再将抽出的牌放回牌堆。

我们重复实验n次，然后用记录的事件数目除以n。如果你有一副扑克牌，则进行10次实验。你的结果是多少？虽然重复这个实验是乏味的，但是我们还是必须准确记录事件数目。因此，使用计算机仿真是必要的。

为了支撑我们的仿真，我们将开发一个Card类和一个CardDeck类。我们的Card类

需要两个属性：一个表示花色，一个表示点数。当我们抽出一张 Card 时，需要“查看结果”，所以要包括两个访问方法，还要包括一个方法以下面的格式显示卡牌：

如果点数是 14，花色为 'S'（黑桃），则显示 Ace of Spades；

如果点数为 10，花色为 'D'（方块），则显示 10 of Diamonds；

以此类推。

我们不需要在类中包含修改方法，因为卡牌不是可变的，这意味着在一个卡牌对象被创建后，其值就不能再改变。

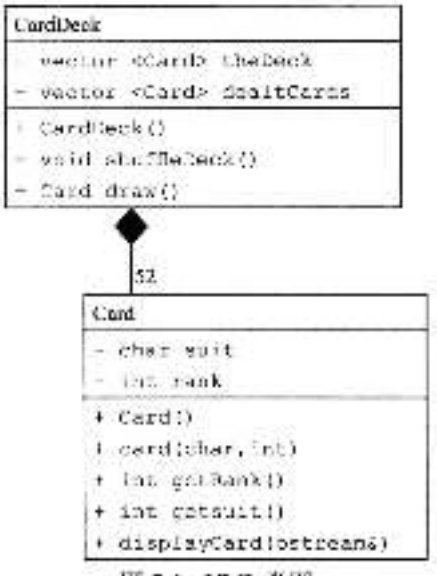


图 7.6 UML 类图

我们的 CardDeck 类需要两个属性：一个存储 52 张扑克牌的 vector，一个保存我们所抽取的牌的 vector。CardDeck 的功能要求有一个洗牌的方法和一个从 CardDeck 抽牌的方法。洗牌的方法是将被抽出的牌的位置替换掉，我们将使用在 <algorithm> 中定义的方法 random_shuffle() 来完成洗牌过程。

图 7.6 中给出了 Card 和 CardDeck 的类图，注意其中的实心方块连接符将两个类图连在一起。这个连接符表示组合。定义为一个 CardDeck 有一个（“has-a”）Card。图中的数字 52 表示一个 CardDeck 中有 52 个 Card 对象。我们将在第 10 章中讨论类的组合和 UML 类图。各个类的定义在下面给出。

类定义：

Card.h

```
/*-------------------------------------------------------------*/
/*We use the following compiler directives to avoid            */
/*including the Card.h file muliple times                      */

/*It is a good idea to always use these directives             */
/*in custom header files.                                      */
#ifndef CARD_H
#define CARD_H

/* Card Class declaration                                      */
/* filename: Card.h                                            */

#include<iostream> //Required for cout,cin
using namespace std;

class Card
  public:
  //Constructors
    Card(); //Default
    Card(char suit,
         int rank);//parameterized
  //Accessors
    int getRank() const;
    char getSuit() const;
```

```
    //Formatted Display Method
    //Example: if char is 'S' and rank is 11
    //output will be:
    //Jack of Spades
      void displayCard(ostream& outStream) const;
    private:
      char suit;
      int rank;
};
#endif  //end compiler directive ifndef
/*-------------------------------------------------------------*/
```

Card.cpp

```
/*-------------------------------------------------------------*/
/* Card class implementation                                   */
/* filename: Card.cpp                                          */

#include "Card.h" //Required for Card
#include<cctype> //Required for toupper()
#include<string> //Required for string
#include<iostream> //require for ostream
using namespace std;

//Constructors
Card::Card():rank(2), suit('S')
{
}
Card::Card(char ch, int i):rank(i)
{
 suit=toupper(ch);
}

//Accessor Methods
int Card::getRank() const
{
  return rank;
}

char Card::getSuit() const
{
  return suit;
}

//Formatted display method
void Card::displayCard(ostream& out) const
{
   string suitString;
   //Establish suit string
   //Constructors and mutators guarantee uppercase suit
   switch(suit)
    case 'S':
      suitString = "Spades";
      break;
    case 'H':
      suitString = "Hearts";
      break;
    case 'D':
      suitString = "Diamonds";
      break;
```

```
    case 'C':
      suitString = "Clubs";
      break;
    default:
      suitString = "Invalid Suit";
  }//end switch suit

  if(rank >= 2 && rank < 11)
  { //output the rank and suit string
    cut << rank << " of " << suitString;
  } //end if
  else
  { //Establish rank string(Ace, King, Queen, or Jack)
    switch(rank){
      case 11:
        out << "JACK of " << suitString;
        break;
      case 12:
        out << "QUEEN of " << suitString;
        break;
      case 13:
        out << "KING of " << suitString;
        break;
      case 14:
        out << "ACE of " << suitString;
        break;
    }//end switch rank
  }//end else
  return;
}//end Display
/*-----------------------------------------------------------*/
```

CardDeck.h

```
/*-----------------------------------------------------------*/
#ifndef CARDDECK_H
#define CARDDECK_H
/* CardDeck class declaration                                */
/* filename: CardDeck.h                                      */
#include "Card.h" //Required for Card
#include <vector> //Required for vector                     */
using namespace std;

class CardDeck {
public:
  CardDeck();
  void shuffleDeck();
  Card draw();
private:
  vector<Card> theDeck;
  vector<Card> del Cards;
};
#endif
/*-----------------------------------------------------------*/
```

CardDeck.cpp

```
/*-----------------------------------------------------------*/
/* CardDeck implementation                                   */
/* filename: CardDeck.cpp                                    */
```

```
#include "CardDeck.h"
#include<ctime> //Required for time()
#include<algorithm> //Required for random shuffle()
using namespace std;

CardDeck::CardDeck()
{
  /* 13 cards in each suit.                                    */
  /* 52 new cards.                                             */
  for(int i=0; i<13; ++i)
  {
    theDeck.push_back(Card('S',i));
    theDeck.push_back(Card('H',i));
    theDeck.push_back(Card('D',i));
    theDeck.push_back(Card('C',i));
  }
  srand(time(NULL)); //Must seed RAND_ time!
}
Card CardDeck::draw()
{
/* Draw and return one card.                                   */
  if(theDeck.empty())
  {
    exit(1);
  }
  Card aCard  = theDeck.back(); //Draw card.
  theDeck.pop_back();//Remove card.

  //Retain card for shuffle.
  delCards.push_back(aCard);
  return(aCard);
}

void CardDeck::shuffleDeck()
{
 /* Replace drawn cards                                        */
 for(int i=0; i<delCards.size(); ++i)
 {
   theDeck.push_back(delCards[i]);
 }
 //Clear the vector.
 delCards.resize(0);

 //Use the top level function from Algorithm.
 random_shuffle(theDeck.begin(), theDeck.end());
}
/*************************************************************/
```

4. 算法设计

我们首先给出分解提纲。

分解提纲

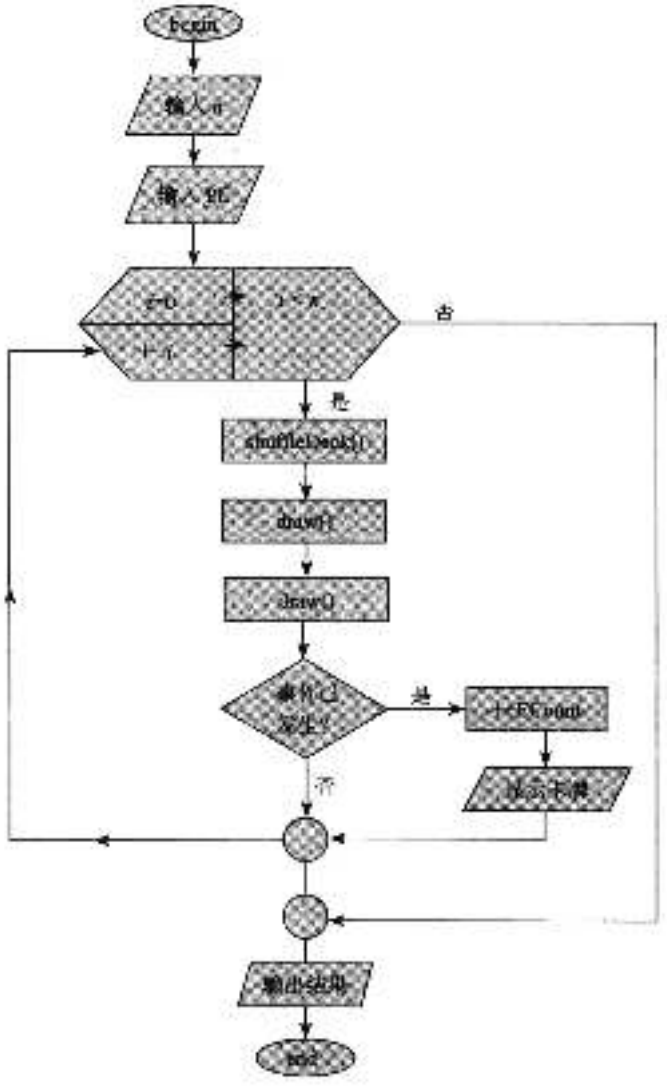
1）读取实验次数；

2）读取解析概率；

3）进行仿真；

4）打印概率的比较情况。

步骤1需要提示用户输入实验的数目。步骤2需要提示用户输入解析结果。步骤3要使用一个循环来完成n次实验。步骤4中，打印出结果的比较情况。算法的流程图在下面给出。



我们的流程图提供了足够的细节来开发问题解决方案。

```
/*                                                              */
/* Program Chapter7_12                                          */
/* This program runs a simulation to calculate                  */
/* the probability of drawing 2 red cards, each                 */
/* with a face value less than 10, from a                       */
/* shuffled deck of 52 cards.  The results are                  */
/* then compared to analytical results.                         */
#include<iostream> //Required for cout.
#include "Card.h" //Required for Card.
#include "CardDeck.h" //Required for CardDeck.                  */
using namespace std;
```

```
int main()
{
  //Declare objects.
  CardDeck theDeck;
  Card card1, card2;
  double pE(0), eventCounter(0);
  int n(0);
  bool isRed;

  cout << "Enter the analytical result ";
  cin >> pE;
  cout << "Enter number of experiments to run ";
  cin >> n;

  for(int i=0; i<n; ++i)
  {
    theDeck.shuffleDeck();
    card1 = theDeck.draw();
    card2 = theDeck.draw();

    //Check if the event occured.
    if( (card1.getSuit() == 'H' || card1.getSuit() == 'D')
      &&(card2.getSuit() == 'H' || card2.getSuit() == 'D'))
    {
      isRed = true;
    }
    else
    {
      isRed = false;
    }
    if( isRed && (card1.getRank() < 10 && card2.getRank() < 10) )
    {
      ++eventCounter;
      cout << "Event " << eventCounter << endl;
      card1.displayCard(cout);
      cout << " ";

      card2.displayCard(cout);
      cout << endl;
    }//end if
  }//end for
  cout << "Analytical results: " << pE << endl
       << "Simulated results: " << eventCounter << '/' << n
       << " = " << eventCounter/n << endl;
  return 0;
}
/*------------------------------------------------------------------*/
```

5. 测试

我们将对解决方案进行两次测试，实验的次数选择了较小的数目，这样可以确保我们能准确地统计事件发生的次数。

仿真 1

```
Enter the analytical result 0.0904877
Enter number of experiments to run 97
Event 1
```

```
4 of Diamonds 8 of Diamonds
Event 2
5 of Diamonds 2 of Hearts
Event 3
3 of Hearts 2 of Hearts
Event 4
4 of Diamonds 7 of Hearts
Event 5
7 of Hearts 8 of Hearts
Event 6
5 of Hearts 2 of Diamonds
Event 7
8 of Diamonds 4 of Hearts
Event 8
9 of Hearts 8 of Diamonds
Event 9
4 of Diamonds 9 of Hearts
Analytical results: 0.0904977
Simulated results: 9/97 = 0.0927835
```

仿真2

```
Enter the analytical result 0.0904577
Enter number of experiments to run 97
Event 1
6 of Hearts 4 of Hearts
Event 2
8 of Hearts 5 of Hearts
Event 3
8 of Diamonds 7 of Hearts
Event 4
6 of Hearts 5 of Diamonds
Event 5
3 of Diamonds 8 of Hearts
Event 6
4 of Diamonds 3 of Hearts
Event 7
3 of Hearts 4 of Hearts
Analytical results: 0.0904577
Simulated results: 7/97 = 0.0721645
```

为了在我们的仿真中采用较大的仿真次数运行，我们将程序中的打印语句注释掉，以加快执行。

仿真3

```
Enter the analytical result 0.0904977
Enter number of experiments to run 101371
Analytical results: 0.0904977
Simulated results: 9171/101371 = 0.0904697
```

仿真4

```
Enter the analytical result 0.0904977
Enter number of experiments to run 101371
Analytical results: 0.0904977
Simulated results: 9207/101371 = 0.0957791
```

```
仿真5
Enter the analytical result 0.0904977
Enter number of experiments to run 1111111
Analytical results: 0.0904977
Simulated results: 100566/1111111 = 0.0905012
```

修改

下面的练习需要对程序 chapter7_12 进行修改，但是不需要对 Card 和 CardDeck 类进行修改。

1. 修改程序 chapter7_12，计算出抽取 2 张花牌的概率。
2. 修改程序 chapter7_12，计算出抽取 5 张黑牌的概率。
3. 修改程序 chapter7_12，计算出抽取 4 张 A 的概率。

本章小结

关键术语

alphanumeric character（字母数字字符）
array（数组）
binary search（二分搜索）
event space（事件空间）
C-style string（C 风格字符串）
character string（字符串）
mean（平均值）
median（中值）
null character（空字符）
offset（偏移量）
one-dimensional array（一维数组）
sample space（样本空间）
selection sort algorithm（选择排序算法）
sequential search（顺序搜索）
sorting（排序）
standard deviation（标准差）
statistical measurements（统计表征数）
string class（string 类）
subscript（下标）
variance（方差）
vector class（vector 类）
whitespace（空白）
zero crossing（零交叉）

C++ 语句总结

包含 C 风格字符串头文件

```
#include<cstring>
```

包含 string 类头文件

```
#include<string>
```

包含 vector 类头文件

```
#include<vector>
```

数组和 vector 声明

```
int a[5], b[]={2, 3, 1};
char vowels[]={'a', 'e', 'i', 'o', 'u'};
string words[100];
vector<double> nums;
```

注意事项

1. 标识符 i 通常用作一维数组的下标。
2. 使用符号常量声明数组的大小，这样易于修改。

调试要点

1. 只有当需要在内存中保证所有数据可用时才使用数组。
2. 如果在 C 风格字符串的末尾打印出了非预期的字符，说明空字符可能已经从字符串里丢失了。
3. 在引用数组中的元素时注意不要超过最大偏移量。
4. 数组声明时必须尽可能大，或者超过所要存储的数据的最大数目。
5. 因为在函数中对数组的引用总是按照引用传递的，注意不要不小心在函数中改动数组的值。

习题

判断题

1. 如果初始化序列比数组长度短，那么其他的值都被初始化为 0。
2. 如果数组定义时没有指明大小，但是给出了初始化序列，那么数组的大小是不确定的。
3. 当下标或偏移量的值超过了数组的最大合法偏移量，那么将总会导致执行错误。
4. 如果我们只说明了数组标识符，而没有指定偏移量，那么数组中的所有值都会被打印。
5. 以 ASCII 码比较，字符串“Smith”小于“Johnson”。

多选题

6. 一个数组是（　　）。
   (a) 一组具有相同对象名的值，且所有值的数据类型相同
   (b) 一组存在相邻内存单元而具有不同数据类型的元素
   (c) 一个包含多个相同数据类型的值的对象
   (d) 一个存储了多个相同数据类型的值的内存地址
7. 数组中单个元素的寻址是指明（　　）。
   (a) 数组名和元素的偏移量　　(b) 数组名
   (c) 元素在数组内的偏移量后跟数组名　　(d) 元素在数组中的偏移量
8. 偏移量指出了特定元素在数组中的（　　）。
   (a) 位置　　(b) 值
   (c) 范围　　(d) 名字

内存快照问题

给出在下面每组语句执行后对应的内存快照情况。使用“?”表示一个未被初始化的数组元素。

9.
```
int t[5];
...
t[0] = 5;
for(int k=0; k<4; k++)
   t[k+1] = t[k] + 3;
```

10.
```
char s[] = "Hello", t[] = {'a', 'e', 'i', 'o', 'u'}, name[10];
strcpy(name,"Sue");
```

程序输出

在问题 11 ~ 13 中使用了下面的语句：

```
string strng1 = "X", strng2 = "zoo", strng3 = "xyz";
```

11. 下面语句的输出是什么？

```
cout << string1.size() << endl;
```

12. 下面语句的输出是什么？

```
string s = string2;
cout << s.at(1) << endl;
```

13. 下面语句的值是什么？

```
string1 == string2;
cout << string1 << endl;
```

编程题

线性插值。下面的问题与存储在文件 tunnel.dat 中的风洞测试数据有关。文件中每行包含了一组出航迎角（以度为单位）和对应的升力系数组成的数据，其中航迎角为降序排列。

14. 编写一个程序读取风洞测试数据，并允许用户输入一个航迎角。如果航迎角在数据集中，则程序应当使用线性插值来计算对应的升力系数。（你可能需要参考 2.5 节中有关线性插值的小节。）
15. 修改问题 14 中的程序，在读取数据文件中的值后，打印出数据文件中航迎角的范围提供给用户。
16. 编写一个函数用于验证航迎角是降序排列的。如果角度不是有序的，函数返回 0，否则返回 1。假设对应的函数原型为

```
int ordered(double x[], int num_pts);
```

17. 编写一个函数，它接受两个一维数组，分别对应于航迎角和相应的升力系数。函数应当将航迎角按照降序排列，同时保持相应的升力系数仍与原航迎角相对应。假定该函数原型为：

```
void reorder(double x[], double y[]);
```

18. 修改问题 14 中的程序，让它使用问题 16 中的函数来确定数据是否按照要求的顺序排列。如果没有按照要求的顺序排列，则使用问题 17 中的函数对数据进行重新排序。

噪声信号。在工程仿真中，我们常常希望生成一个具有特定均值和方差的浮点序列。在第 6 章中开发的函数允许我们生成限制在 a 和 b 之间的数，但是它并不能让我们指定均值和方差。通过使用概率论的结论，可以得到下面定义在均匀随机序列中其理论均值 μ 与理论方差 σ^2 之间的关系：

$$\sigma^2 = \frac{(b-a)^2}{12}, \quad \mu = \frac{a+b}{2}$$

19. 编写一个程序使用第 5 章中开发的 rand_float 函数生成一个 4 ~ 10 之间的随机浮点数序列，然后将计算出的均值和方差与理论值进行比较。当使用的随机数越来越多时，计算出的值与理论值之间将越来越接近。
20. 编写一个程序，使用第 5 章中开发的 rand_float 函数生成两个具有 500 个点的序列。每个序列的理论均值为 4，但是一个序列的理论方差为 0.5，另一个序列的理论方差为 2。检查计算出的均值并与理论均值比较。（提示：使用前面两个公式每到两个计算出未知量的等式，然后手工计算出未知量。）
21. 编写一个程序，使用第 5 章中开发的 rand_float 函数生成两个具有 500 个点的序列。每个序列的方差都是 3.0，但是一个序列的均值为 0.0，另一个序列的均值为 −4.0。比较均值和方差的理论值和计算值。（提示：使用前面两个公式得到两个计算出未知量的等式，然后手工计算出未知量。）
22. 编写一个名为 rand_nov 的函数，生成一个随机浮点数序列，其中数列的均值和方差由函数的输入参数指定。假定对应的函数原型为：

```
double rand_nov(double mean, double var);
```

使用第 5 章中开发的 rand_float 函数。

密码学。数百年来，设计密码的科学吸引了很多人。一些最简单的密码包括将一个或一组字符用另一个或另一组字符替换。为了更容易地对这些消息进行解码，解码器需要指出替换字符的 key（用于表示字符替换的规则）。最近，计算机已经被成功用于破解那些原以为不可破解的密码。下一组问题考虑了简单的密码和它们的解码方案。生成文件来测试这些程序。

23. 在不知道密码的编码方案时要解码一个简单密码的步骤之一就是统计每个字符的出现次数。然后，我们知道在英语中最常见的字符是 'e'，将编码后的消息中出现最多的字符用 'e' 替代。基于我们已知的英语中各字符的出现频度以及编码后消息中各字符的出现频度，我们进行类似的替换。这种解码过程通常提供了足够准确的替换。同时可以确定出不正确的替换。对于这个问题，编写一个程序读取数据文件，并确定文件中每个字符的出现次数；然后打印出每个字符及其出现的次数。如果某个字符不存在，则不打印。（提示：基于字符的 ASCII 码值，使用数组存储每个字符的出现次数。）
24. 另一种简单的密码将秘密消息编码在文本中，真正的消息由每个单词的第一个字符组成。在单词之间没有空格，但是人们可以很容易地将解码的字符串分解成单词。编写一个程序从数据文件中读取数据，通过单词的首字符确定秘密消息。（提示：将每个单词的首字符存储到一个字符串中。）
25. 假定问题 24 中的秘密消息存储在每个单词的第二个字符中。编写一个程序从数据文件中读取数据，确定存储在文件中的秘密消息。
26. 假定问题 24 中真正的秘密消息是由从第一个字符开始向后的二个字符表示的。编写一个程序读取数据文件，并使用这种解码方案确定存储在文件中的秘密消息。
27. 编写一个程序使用一个包含 26 个字符的名为 key 的字符数组对数据文件中的文本进行编码。key 的内容从键盘读取；数组中的第一个字符用来替代数据文件中的字母 a。第二个字符替代数据文件中的字母 b，以此类推。假定所有的标点符号都使用空格替代。编码时检查并确定数组 key 没有将两个不同的字符映射到同一个字符中。
28. 编写一个程序解码问题 27 中的输出文件。假定本程序从键盘读取的 key 与前一问题中的 key 相同，并将其用于解码过程中。注意，你不能恢复出标点符号。

回文（palindrome）是不管从前读还是从后读都一致的单词（noon），句子（Draw a level award）或数字（18781）。（注意在确定单词、句子或数字是否为回文时忽略空白。）

29. 编写一个程序从标准输入读入一行文本，确定读取的这行文本是否是回文。
30. 编写一个程序从数据文件中读取文本。程序应当读取数据文件中每一行（读取所有的行直到文件结束为止），打印出每一行，并报告一条消息说明这一行是也是回文。

概率计算。

31. 为 CardDeck 类添加一个名为 randomDraw() 的方法。该方法应当返回从牌堆中随机抽出的一张牌。使用这个方法来仿真从一副扑克牌中抽出两张龙牌的概率。
32. 为 CardDeck 类添加一个名为 isEmpty() 的方法。当牌堆为空时方法应返回 true，否则返回 false。编写一个驱动来测试这个新方法。
33. 编写一个程序，使用 Card 和 CardDeck 类来处理 13 张牌，统计这些牌中的龙牌点数。假设 A 为 4 点，K 为 3 点，Q 为 2 点，J 为 1 点。因此，如果有可张 A、一张 K 和 3 张 J，那么总点数为 14。程序应当显示处理的一手牌和这一手牌的点数。
34. 编写一个仿真来估计从一副 52 张的扑克牌中抽出 13 张不含龙牌的概率。
35. 编写一个仿真来估计抽出的 5 张牌中含有 2 张 K 和一张 A 的概率。
36. 编写一个仿真来估计抽出的 5 张牌中含有两对牌的概率。

自定义数据类型。

37. 定义一个类来实现行的概念。你的类中应当有两个私有数据成员：length，fixedPoint。类中包括一组完整的访问方法、修改方法和构造函数，其他的方法还包括：
    - 打印位置在行的长度

- 移动行
- 改变行的长度

画出 UML 类图，并编写一个驱动来测试你的类。

38. 定义一个类来实现圆的概念。你的类应当包括两个私有数据成员：radius、centerPoint。类中包括一组完整的访问方法、修改方法和构造函数，其他的方法还包括：
    - 打印位置和圆的半径
    - 移动圆
    - 改变圆的大小

    画出 UML 类图，并写一个驱动来测试你的类。

39. 定义一个类来表示日期。你的类应当包括三个数据成员：month、day 和 year。类中包括一组完整的访问方法、修改方法和构造函数，其他的方法还包括：
    - 以 month/day/year（10/1/1999）的形式打印日期
    - 以 month day,year（October 1, 1999）的形式打印日期

    画出 UML 类图，并写一个驱动来测试你的类。

40. 定义一个以军用形式表示时间的类。你的类应当包括三个私有数据成员：hours、minutes 和 seconds。军用时间的范围为 00:00:00（中午 12 点）～23:59:59（晚上 11:59:59）。类中应包括一组完整的访问方法、修改方法和构造函数。其他的方法还包括：
    - 以 12 小时格式打印出时间
    - 计算两个时间之间的差（重载“-”操作符）

海啸预警系统。波浪陡度是波高（Wave Height，WH）与波长（Wave Length，WL）的比值，它是波浪稳定性的指示器。当波浪陡度超过 1/7，波浪就变得不稳定并开始破碎。假定某个数据文件中有在下面格式的头部：

```
#YY  MM  DD  HH  MM  WH(m)  WL(m)
```

每个后续行都包含时间和波高测量值，形式如下：

```
year(int)  month(int)  day(int)  hour(int)  minute(int)  wave height(double)  wave length(double)
```

41. 编写一个程序计算以上面格式给定的输入文件中的平均陡度。程序应当确定并打印出超过平均陡度的时间所占百分比。输入文件可以是任意长度。

42. 给定一个具有前述格式的输入文件，编写一个程序生成关于该输入文件的报告。程序应当计算并报告数据文件中每个条目的陡度，同时在陡度超过 1/7 时给出一条警告消息。程序应当打印出关于波浪测量值中前 50% 的波高、波长和陡度的报告。输入文件长度任意。

二维数组

工程挑战：地形导航

火星探测漫游者员于在火星表面移动并检查土壤和岩石的详细信息。自动导航系统使用漫游者的一对立体照相机对前面的地形拍照。在获得立体图像后，漫游者上的软件将自动生成 3D 地形图。系统的可穿越性和安全性可以通过岩石的高度和密度、地形的倾斜度和粗糙度来确定。漫游者需要从几十条可能的路线中选取最短、最安全的路线向预定地点移动。在从一个地点移动到另一个地点时，漫游者都要进行就地的地理导航。可完成移动的机械臂与人类的手臂相似，也有肘和腕，并且能够直接将仪器放在所感兴趣的岩石或土壤目标上。本章我们将分析来自地形地图的离散数据，以确定地形中各点的数目和位置。

教学目标

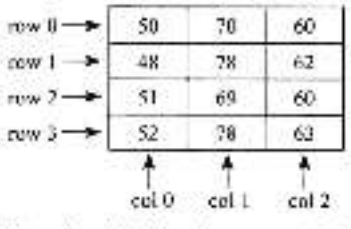
本章我们所讨论的问题解决方案中包括：

- 矩阵计算
- 来自数据文件的输入
- 求和函数和计算平均数的函数
- 解决联立方程的技术
- 用 vector 实现二维数组

8.1 二维数组

一组可以看做一行或一列的数据可以很容易地使用一个一维数组来表示。但是，在很多例子中数据最好的表现方式是网格或者数据表的方式。

假设我们希望显示过去 4 天中每隔 8 小时所记录的温度值表。要显示的数值可以以下面四行三列的数组形式表示，其中四行对应于过去的四天，而三列对应于 24 小时内所记录的温度数目：



在 C++ 中，数据表可以使用一个二维数组表示。二维数组中的每个元素都使用一个标识符与两个偏移量来引用，一个行偏移量（row offset）和一个列偏移量（column offset）。行偏移量和列偏移量都从 0 开始，每个偏移量都有自己的一对方括号。因此，假定前面的数组有标识符 temp，那么 temp[2][1] 的值为 69。在数组元素中常见的错误包括使用圆括号而不

是方括号，如 temp(2)(3)，或者只使用一对方括号或圆括号，如 temp[2,3] 或 temp(2,3)。

在二维数组中所有的值都必须具有相同的数据类型。一个数组不能出现诸如在一行整数之后跟着一行浮点数这样的情况。

8.1.1 声明和初始化

为了声明一个二维数组，我们需要在声明语句中指明行数和列数，其中行数在前。行数和列数都在方括号中，如下面的声明语句所示：

```
int temp[4][3];
```

temp 的声明语句会为 temp 分配一块连续的内存，并足够存放 12 个整型数。

二维数组可以使用声明语句和一个初始化列表进行初始化，如：

```
int temp[4][3] = {50, 70, 60, 48, 75, 62, 51, 69, 60, 52, 78, 63};
```

注意我们在初始化列表中列出的数值是按照逐行逐行赋值的，因为这是 C++ 编译器为内存赋值的顺序。

下一个声明语句说明了在初始化列表中使用分块形式显式地为二维数组的每行赋值的用法。这里赋值的结果将和上面相同。

```
int temp[4][3] = {{50, 70, 60}, {48, 75, 62},
                  {51, 69, 60}, {52, 78, 63}};
```

如果类似声明语句中提供的初始化列表比数组的长度要短，那么剩下的数组的其他元素都将被初始化为 0。作为示例，我们给出下面的声明语句：

```
int t2[4][3] = {{50, 70, 60}, {48, 75, 62},
                {51, 69, 60}, {52, 78, 63}};
```

该语句将会为数组 t2 的左上部分赋值，如下所示。

50	70	60	0
48	75	62	0
51	69	60	0
52	78	63	0
0	0	0	0
0	0	0	0
0	0	0	0

因此，分块的用法允许我们初始化二维数组的特定子集。

二维数组在声明时可以将行数留空，但是必须有一个初始化列表。列表中的数值的数目，或者分块的数目将确定行数。因此，下面的两个类似声明语句都正确定义了数组 temp：

```
int temp[][3] = {{50, 70, 60}, {48, 75, 62}, {51, 69, 60},
                 {52, 78, 63}};
int temp[][3] = {50, 70, 60, 48, 75, 62, 51, 69, 60, 52, 78, 63};
```

在声明一个二维数组时，行数是唯一可以留空的维数。将列数留空将会导致编译错误。

> **二维数组**：二维数组的声明需要一个行数和一个列数。大小为（行数）*（列数）的一块连续的内存将分配给数组，数组名存放了内存的首地址。访问数组的值需要在数组名后跟行偏移量和列偏移量。

语法

数据类型 标识符 [行数][列数] [=初始化列表];

示例

```
int data[2][5];    // 分配用于 10 个整数的连续存储空间
```

data 的内存快照：

int data:	?	?	?	?	?
	?	?	?	?	?

```
double t[2][2] = {{3.0, 5.0}, {2.1, 7.2}}; // 内存分配和初始化
```

t 的内存快照：

double t:	3.0	5.0
	2.1	7.2

合法引用：

```
cout << data[0][0];
```

非法引用：

```
cout << t x[1,2];    // 非法偏移量
```

练习

给出下面每条语句中定义的数组的内存快照。

1. int a[3][2] = {{1},{2},{3}};
2. int b[2][3] = {1,2,3,4,5,6};
3. int c[][4] = {{1,2,3,4}, {0,5,0,6}, {8,3, 5, 1}};
4. int d[][4] = {2,2,3,0,3,0,6,5,-5,-1};

数组也可以在程序语句中赋值。对于二维数组，通常需要使用两个嵌套的 for 循环来初始化一个数组，通常使用 i 和 j 来代表偏移量。下面的语句定义并初始化了一个数组：

```
// Declare objects
int t[5, 4];
...
// Assign values to array.
for (int i=0; i<5; ++i)
{
   for (int j=0; j<4; ++j)
   {
      t[i, j] = i;
   }
}
```

数组 t 的内存快照如下所示：

偏移量	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
int t	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4

以行列形式表示：

int t	0	0	0	0
	1	1	1	1
	2	2	2	2
	3	3	3	3
	4	4	4	4

二维数组还可以从数据文件中读取数据进行赋值。在下面的程序中我们从一个名为engine.dat的数据文件中读取温度值，数据文件的第一行包含两个整数。第一个整数说明了引擎的数目（行数）。第二个整数说明了字段的数目（列数）。数据文件中剩下的行中包含了（行数）*（列数）个温度值。这些值将被读取并存储在数组中。符号常量NROWS和NCOLS用来代表行数和列数。当行数和列数使用符号常量表示时，改变数组的大小会变得很容易；否则，改变数组的大小将需要修改若干语句。

```
/*-------------------------------------------------------------*/
/* Program Chapter8                                            */
/* This program reads and stores temperature values            */
/* for multiple trials from the input file engine.dat          */

#include<iostream> //Required for cerr, cout
#include<fstream>  //Required for ifstream
using namespace std;
int main()
{
// Declare objects.
   int numEngines, numTrials;
   ifstream data;

   //Open input file.
   data.open("engine.dat");
   if(data.fail())
   {
      cerr << "error" not open engine.dat";
      exit (1);
   }
   //Input row and column size
   data >> numEngines >> numTrials;

   //Declare constants for array declaration
   const int NROWS(numEngines);
   const int NCOLS(numTrials);
   double temps[NROWS][NCOLS];

   //Read temperature data and store in array.
   for (int i=0; i<NROWS; ++i)
   {
      for (int j=0; j<NCOLS; ++j)
      {
         data >> temps[i][j];
      }
   }
   data.close();

   //Echo input.
   cout << numEngines << ' ' << numTrials << endl;
   for (int i = 0; i<NROWS; ++i)
   {
      for (int j=0; j<NCOLS; ++j)
      {
         cout << temps[i][j] << '\t';
      }
      cout << endl;  //end of ith row
   }
   return 0;
}
/*-------------------------------------------------------------*/
```

下面给出了程序 chapter8_1 中第一次循环的程序跟踪和内存快照：

```
        engine.dat
3 3
335.6 339.4 421.7
299.6 332.4 340.5
320.1 329.8 339.3
```

main()	内存快照			
步骤 1: data1 >> rows >> cols;	int rows	3	int cols	3
步骤 2: const int NROWS(rows);	int NROWS	3		
步骤 3: const int NCOLS(cols);	int NCOLS	3		
步骤 4: double temps[NROWS][NCOLS];	double temps	? ? ? ? ? ? ? ? ?		
步骤 5: for(int i=0;	int i	0		
步骤 5A: i<NROWS, true(i=0)				
步骤 6: for(int j=0;	int j	0		
步骤 6A: j<NCOLS; true(j=0)	true(j=1)		true(j=2)	
步骤 6B: data1>>temps[i][j];				
步骤 6C: ++j;				
temps	335.6 ? ? ? ? ? ? ? ?	335.6 339.4 ? ? ? ? ? ? ?	335.6 339.4 421.7 ? ? ? ? ? ?	
步骤 5B: ++i	i	1		
步骤 7: data1.close();				

程序 chapter8_1 生成的输出如下：

```
3,3
335.6 339.4 421.7
299.6 332.4 340.5
320.1 329.8 339.3
```

练习

给出下面每组语句定义的数组的内存快照，使用“?”表示未被初始化的元素。

1. int d[3][1]={{1},{4},{6}};
2. int g[4][2]={{5,2},{-3,3}};
3. double h[4][4]={{0,0}};
4. int p[3][3]={{0,0,0}};

```
...
for (int k=0; k<3; ++k)
```

```
    {
      p[i][k] = 1;
    }
5. int g[5][5];
   ...
   for (int i=0; i<5; ++i)
   {
     for (int j=0; j<4; ++j)
     {
       g[i][j] = i + j;
     }
   }
6. int g[5][5];
   ...
   for (int i=0; i<=4; ++i)
   {
     for (int j=0; j<=4; ++j)
     {
       g[i][j] = pow(-1.0,j);
     }
   }
```

8.1.2 计算与输出

在对二维数组进行计算和输出时，必须指明两个偏移量来引用数组的元素。为了说明，考虑下一个程序，该程序从一个包含电气设备在10周的时间内的功率输出的数据文件中读取数据。数据文件中每行包含7个值，代表一周中每天的功率输出。数据被存储在一个二维数组中，然后打印出一个包含每周中每一天的平均功率的报告。

```
/*------------------------------------------------------*/
/*  Program chapter8_2                                  */
/*                                                      */
/*  This program computes power averages                */
/*  over a 10-week period.                              */
#include<iostream>    //Required for cin, cout, ears
#include<fstream>     //Required for ifstream
#include<string>      //Required for string
using namespace std;

int main()
{
   // Declare objects.
   const int NROWS = 10;
   const int NCOLS  7;
   double power[NROWS][NCOLS], col_sum;
   string filename;
   ifstream data1;

   // Open file and read data into array
   cout << "Enter name of input file.\n";
   cin >> filename;
   data1.open(filename.c_str());
   if(data1.fail())
   {
     cout << "Error opening data file\n";
     exit(1);
```

```
//
// Set format flags.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(1);
for (int i=0; i<NROWS; ++i)
{
   for (int j=0; j<NCOLS; ++j)
      data1 >> power[i][j];
}
// Compute and print daily averages.
for (int j=0; j<NCOLS; ++j)
{
  col_sum = 0;
  for (int i=0; i<NROWS; ++i)
     col_sum += power[i][j];
  cout << "Day" << j+1 << " Average " << col_sum/NROWS << endl;
}
// Close file and exit program.
data1.close();
return 0;
}
//-------------------------------------------------------------------*/
```

注意日平均值是将每一列的值相加并将得到的和除以行数（在这里是每周的天数）计算得到的。然后列数将用于计算日数。程序的一次示例输出如下：

```
Day 1: Average = 236.4
Day 2: Average = 199.3
Day 3: Average = 271.6
Day 4: Average = 259
Day 5: Average = 277.0
Day 6: Average = 305.6
Day 7: Average = 276.1
```

将来自二维数组的信息写入文件中与将一维数组的信息写入文件类似。在两种情况下，都必须指明一个换行指示符用以标识什么时候开始一个新行。下面的语句将一组距离测量值写入一个名为 dist.txt 的数据文件，每行有 5 个值：

```
// Declare objects.
double dist[20][5];
ofstream data1;
...
// Write information from the array to a file.
data1.open("a:\\dist.txt");
for (int i=0; i<20; ++i)
{
   for (int j=0; j<5; ++j)
   {
      data1 << dist[i][j] << ' ';
   }
   data1 << endl;
}
```

在输出语句中 dist[i][j] 之后所打印的空格是需要的，这样保证每个值由空格分隔。

练习

假定下面的语句声明了数组 g：

```
int g[2][3]={{0.0,0.0},{1.,1.,1.},{2.5,2.1}};
```

给出下面每组语句执行之后 sum 的值。

```
1. sum = 0;
   for (int i= 0; i<3; i++)
   {
     for (int j=0; j<3; ++j)
     {
       sum += g[i][j];
     }
   }
```

```
2. sum = 1;
   for (int i=1; i<3; i++)
   {
     for (int j=0; j<2; j++)
     {
       sum *= g[i][j];
     }
   }
```

```
3. sum = 0;
   for (int j=0; j<3; ++j)
   {
     sum += g[2][j];
   }
```

```
4. sum = 0;
   for (int i=0; i<3; ++i)
   {
       sum += g[i][i];
   }
```

8.1.3 函数参数

当数组用作函数参数时，默认是按照引用传递而非值传递。如在第7章所讨论的，这意味着函数中的数组参数直接引用源数组，而非数组的复制。因此，在我们不希望改变源数组的值时，必须小心。

当使用一维数组作为函数参数时，函数只需要数组第一个元素的地址，该地址由数组名指出。当使用二维数组作为函数参数时，函数也需要有关数组列数（column size）的声明。一般来说，函数头和函数原型需要给出二维数组列数的声明。为了说明，假设我们需要编写一个程序计算包含四行四列的数组的所有元素之和。计算这个和需要两个嵌套的循环，如果我们将计算和的步骤放在一个函数中，程序的可读性将更好。这样程序就可以使用一条语句来调用函数，如下所示：

```
//.....................................................................
#include <iostream>    //Required for cout
using namespace std;
const int NROWS=4, NCOLS=4;

//  Function prototypes.
int sum(int x[][NCOLS]);
...
int main()
{
   //  Declare objects.
   int a[NROWS][NCOLS];
   ...
   //  Reference function to compute array sum.
   cout << "Array sum = " << sum(a) << endl;
```

如果我们希望在程序的其他几个地方重新计算数组的和，这时候采取就显得更加有效率了。如果有几个不同的数组，则可以使用相同的函数来计算它们的和：

```
#include <iostream>    //Required for cout
using namespace std;
const int NROWS=4, NCOLS=4;
```

```
// Function prototypes.
int sum(int x[][NCOLS]);
...
int main()
{
   // Declare objects.
   int a[NROWS][NCOLS], b[5][NCOLS];

   // Reference function to compute array sums.
   cout << "Sum of a = " << sum(a) << endl;
   cout << "Sum of b = " << sum(b) << endl;
```

现在我们给出在这些语句中调用的函数：

```
/*-------------------------------------------------------*/
/*  This function returns the sum of the values in       */
/*  an array with NROWS rows and NCOLS columns.          */
/*  PreCondition: Array x has NROWS and NCOLS.           */
/*  PostCondition: Sum of integer values is returned.    */

int sum(int x[][NCOLS])
{
   // Declare and initialize local objects.
   int total(0);
   // Compute a sum of the array values.
   for (int i=0; i<NROWS; ++i)
   {
      for (int j=0; j<NCOLS; ++j)

         total += x[i][j];
      }

   }

   // Return sum of array values.
   return total;
}
/*-------------------------------------------------------*/
```

在下一个例子中，我们开发了一个函数用于计算数组中元素的部分和。假定计算的元素位于数组左上角的子数组（subarrary）中，函数的参数包括源数组、子数组的行数和列数。函数原型为

```
// Function prototype
int partial_sum(int x[][NCOLS], int r, int c);
```

因此，如果我们想计算数组 a 中阴影区域的元素之和，可以使用下面的调用语句

```
partial_sum(a,2,3);
```

2	5	−3	9
0	−5	9	7
2	6	3	3
−2	10	4	6

这个调用将计算在左上角由两行三列组成的子数组中的元素之和，函数将返回 6。该函数如下所示：

```
/*-------------------------------------------------------*/
/*  This function returns the sum of the values          */
/*  in a subarray of an array with a declared            */
/*  NCOLS column size of NCOLS.                          */
```

```
int partialSum(int x[][NCOLS],int m, int n)
{
   // Declare and initialize local objects.
   int total(0);
   // Compute a sum of subarray values.
   for (int i=0; i<m; i++)
   {
      for (int j=0; j<n; j++)
      {
         total += x[i][j];
      }
   }

   // Return sum of subarray values.
   return total;
}
/*------------------------------------------------------------*/
```

在函数作用域内，该函数可以使用由符号常量 NCOLS 确定大小可变的二维数组。

练习

假定下面的语句来自 main 函数中：

```
int a[4][4] = {-2, 3, -1, 9}, {0, -3, 5, 7},
              {2, 6, 3, 2}, {-2, 10, 4, 6}};
```

确定下面调用本节开发的函数 partialSum 的返回值：

1. partialSum(a,1,4);
2. partialSum(a, ,1);
3. partialSum(a,4,2);
4. partialSum(a,2,4);

作为最后一个例子，我们将修改程序 chapter8_1，在其中添加两个新函数。假如我们想确定数据文件 engine.dat 中每次试验的引擎温度的平均值，然后找出每次试验中低于平均温度的引擎。因为每次试验都是作为数组中的一列存储的，所以我们可以编写一个函数返回 double 类型二维数组中指定列的平均值。我们还要编写一个函数从输入文件中读取数据，并将这些数据存储到一个 double 类型的二维数组中。下面给出了修改后的程序和函数定义。

```
/*------------------------------------------------------------*/
/* Program chapter8_3                                         */
/* This program reads and stores temperature values           */
/* for multiple trials from the input file engine.dat,        */
/* then determines which engines performed below the          */
/* average trial temperature.                                 */

#include<iostream> //Required for cerr, cout
#include<fstream>  //Required for ifstream
using namespace std;

//Declare constants and Function Prototypes
const int MAXCOLSIZE(10);
const int MAXROWSIZE(10);
double columnAvg(const double a[][MAXCOLSIZE],
                 int colNum, int rowc);
void input2Dfstream& in, double a[][MAXCOLSIZE],
             int (max, int cols);
int main()
{
   //Declare objects.
   int numEngines, numTrials;
   ifstream data;
```

```
    double temps[MAXROWSIZE][MAXCOLSIZE];
    double avgTemp;

    //Open input file.
    data1.open("engine.dat");
    if(data1.fail())
    {
       cerr << "could not open engine.dat";
       exit(1);
    }

    //input row and column size
    data1 >> numEngines >> numTrials;

    //Read temperature data and store in array.
    input2D(data1, temps, numEngines, numTrials);

    data1.close();
    //Generate Report
    for(int i=0; i<numTrials; ++i)
    {
      //Calculate average engine temperature for each trial.
      avgTemp = columnAvg(temps, i, numEngines);

      //Generate Report
      cout << "\nTrial " <<(i+1)<<"\tAverage Engine Temperature "
           << avgTemp << endl;
      cout << "-------------------------------------------\n";
      for(int j=0; j<numEngines; ++j)
      {
        if(temps[j][i] < avgTemp)
        {
          cout << "Engine " << (j+1)
               << " performed below the average temp." << endl;
        }
      }
    }
    return 0;
}//end main
/*-------------------------------------------------------------*/

/*                                                             */
/* This function reads data from an input stream and           */
/* assigns the data to the 2D array, arr.                      */
/* Pre-conditions:                                             */
/*  The istream is has been defined.                           */
/*  The integer cols is <= MAXCOLSIZE                          */
/*  The integer rows is <= MAXROWSIZE                          */
/* Post conditions:                                            */
/*  rows*cols values are assigned to the array, arr            */

void input2D(istream& in, double arr[][MAXCOLSIZE],
             int rows, int cols)
{

   for (int i=0; i<rows; ++i)
   {
      for (int j=0; j<cols; ++j)
      {
         in >> arr[i][j];
      }
   }
}
```

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

/*------------------------------------------------------------------------*/
/* This function returns the average of a column in the array            */
/* arr. The integer colNum specifies the column number                   */
/* Pre-conditions:                                                       */
/*  The array arr has rowSize rows of valid data                         */
/*  The integer colNum is < MAXCOLSIZE                                   */
double columnAvg(const double arr [[MAXCOLSIZE], int colNum,
                 int rowSize)
{
  //Declare and initialize local variables
  double avg = 0.0;

  //Sum all values in column colNum
  for(int i=0; i<rowSize; i++)
  {
     avg += arr[i][colNum];


  //Return the average
  return avg/rowSize;
}
/*  ------------------------------------------------------------------------*/
```

注意，在函数 input2D() 的初始注释块中包含了前置条件（pre-condition）和后置条件（post-condition）。前置条件描述了在调用函数时假定为真的条件。如果前置条件不为真，就不能保证函数正确地被执行。后置条件描述了在函数执行期间对形参所做的改变。函数 columnAvg() 没有后置条件，因为形参没有被修改。当编写的函数可能用于多个应用时，包含前置和后置条件将是一个好的做法。

程序 chapter8_3 生成的输出如下：

```
Trial 1 Average Engine Temperature 319.35
------------------------------------------
Engine 2 performed below the average temp.

Trial 2 Average Engine Temperature 340.75
------------------------------------------
Engine 1 performed below the average temp.
Engine 2 performed below the average temp.
Engine 3 performed below the average temp.

Trial 3 Average Engine Temperature 375.85
------------------------------------------
Engine 2 performed below the average temp.
Engine 3 performed below the average temp.
```

修改

1. 使用函数 input2D() 和 columnAvg() 对程序 chapter8_2 进行修改。不要修改函数定义。
2. 修改程序 chapter8_3，打印出在每次试验中高于平均温度的引擎。
3. 修改程序 chapter8_3，打印出在每次试验中低于平均温度的引擎。

8.2 解决应用问题：地形导航

地形导航是机器人宇宙飞船的关键组件。机器人宇宙飞船就是没有人类在其中的宇宙飞船，这些宇宙飞船可以着陆，如火星探测漫游者，或者在表面上方，如火星侦察轨道器。一艘机器人宇宙飞船上有大量的计算机，其中存储了有关飞船即将要工作的地区的地形信息。通过了解任意时间自己所在的位置（也许是有 GPS 接收机的帮助），飞船可以选择最好的路线到达指定位置。如果目的地变更，飞船可以根据自己的内部地图重新计算路线。

为飞船导航的计算机软件必须经过地形信息和拓扑的测试。在计算机数据库中有大片网格状地区的海拔信息。一种与地形导航有关的测量地形网络“难度”的方式就是确定网格中峰点的数目，这里的峰点就是一个其周围所有点的海拔都低于它的点。对于这个问题，我们假定在下面的图中，与网格位置 [m][n] 邻接的 4 个位置的值用于确定 [m][n] 位置是否是峰点：

	grid [m-1][n]	
grid [m][n-1]	grid [m][n]	grid [m][n+1]
	grid [m+1][n]	

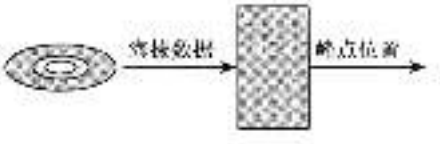
编写一个程序从数据文件 grid1.dat 中读取海拔数据，打印出峰点的数目和它们的位置。假定数据文件的第一行包含了网格信息的行数和列数。右行数和列数之后是按照行的顺序排列的海拔数据。网格的最大尺寸为 25 × 25。

1. 问题描述

确定并打印出在一个海拔网格中峰点的数目和它们的位置。

2. 输入 / 输出描述

I/O 示意图表明输入为包含海拔数据的数据文件，输出为峰点的位置列表。



3. 用例

假设下面的数据代表了一个 6×7 的网格，数据中的峰点已经被加上了下划线：

```
5039 5127 5236 5256 5248 5310 5299
5150 5392 5410 5401 5320 5620 5321
5250 5350 5430 5421 5330 5631 5210
5110 5429 5430 5411 5433 5630 5319
4920 5129 4921 5621 5722 4521 5129
5023 5129 4822 4872 4794 4862 4245
```

为了标出峰点的位置，我们需要为数据使用一个地址方案。因为我们将使用 C++ 实现解决方案，我们使用二维的数组索引来表示。因此，我们假定左上角的位置为 [0][0]，当我们向下移动时，行号以 1 个单位递增，当向右移动时，列号以 1 个单位递增。那么这些峰点的位置为 [2][1]、[2][5] 和 [4][3]。

为了确定峰点，我们将潜在的峰点与其周围的 4 个邻接点进行比较。如果所有 4 个邻接点都小于潜在的峰点，那么该潜在的峰点就是真的峰点。注意在网格边界的点不可能是

潜在的峰点，因为我们没有它们周围所有4个邻接点的海拔信息。

4. 算法设计

我们首先给出分解提纲，它将解决方案分为一系列顺序的步骤。

分解提纲

1）将地形数据读入数组；

2）确定并打印峰点的位置。

步骤1中包括读取数据文件并将信息存入二维数组。步骤2是一个估计所有潜在峰点的循环，如果潜在峰点被确认为真的峰点则打印出它们的位置。我们将编写一个布尔函数来确定一个位置是否是峰点。

细化的伪代码

```
main:  read nrows and ncols from the data file
       read the terrain data into an array
       set i to 1
       while i < nrows - 1;
         set j to 1
         while j < ncols - 1:
           if !ispeak[grid,i,j]
             print peak location
           increment j by 1
         increment i by 1
ispeak:
       if !(grid[i-1][j]<grid[i][j]) &&
          (grid[i-1][j]<grid[i][j]) &&
          (grid[i][j-1]<grid[i][j]) &&
          (grid[i][j+1]<grid[i][j]))
          return true;
       else
          return false;
```

伪代码中的步骤已经足够详细，可以转换成C++语句：

```
/*-------------------------------------------------------*/
/*  Program Chapter8_4                                   */
/*                                                       */
/*  This program determines the locations of             */
/*  peaks in an elevation grid of data.                  */

#include <iostream>   //Required for cin, cout
#include <fstream>    //Required for ifstream
#include <string>     //Required for string
using namespace std;

// Function prototypes.
bool isPeak(const double grid[][N], int r, int c);

int main()
{
   // Declare objects.
   int const N = 25;
   int nrows, ncols;
   double elevation[N][N];
   string filename;
   ifstream file1;

   // Prompt user for file name and open file for input.
   cout << "Enter the name of the input file.\n";
```

```
    cin >> filename;
    file1.open(filename.c_str());
    if(file1.fail())
    {
      cerr << "Error opening input file\n";
      exit(1);
    }
    file1 >> nrows >> ncols;
    if(nrows > N || ncols > N)
    {
      cerr << "Grid is too large, adjust program.";
      exit(1);
    }
    //  Read information from data file into array.
    for (int i=0; i<nrows; ++i)
    {
      for (int j=0; j<ncols; ++j)
      {
        file1 >> elevation[i][j];
      }
    }
    //  Determine and print peak locations.
     cout << "Top left point defined as row 0, column 0 \n";
     for (int i=1; i<nrow-1; ++i)
     {
       for (int j=1; j<ncols-1; ++j)
       {
          if(isPeak(elevation, i, j))
          {
               cout << "Peak at row: " << i
                    << " column: " << j << endl;
          }
       }
     }
     // close file
     file1.close();
    // Exit program.
    return 0;
}
bool isPeak(const double grid[][N], int i, int j)
{
  if ((grid[i-1][j]<grid[i][j]) &&
      (grid[i+1][j]<grid[i][j]) &&
      (grid[i][j-1]<grid[i][j]) &&
      (grid[i][j+1]<grid[i][j]))
      return true;
  else
      return false;
}
//-------------------------------------------------------------*/
```

5. 测试

下面的输出是使用对应于用例中的数据文件所得到的（在该文件的第一行中必须包含海拔数据的行数和列数）：

```
Top Left point defined as row 0, column 0
Peak at row: 2 column: 1
Peak at row: 3 column: 5
Peak at row: 4 column: 3
```

修改

按照下面的要求修改寻找峰点的程序：

1. 打印出网格中峰点的数目。
2. 打印出谷点的位置，而不是打印峰点。假定谷点是一个比邻接点海拔都要低的点。编写一个名为 isvalley 的函数来做的程序调用。
3. 找出并打印给数据中最高点和最低点的位置及其海拔。编写一个名为 extremes 的函数供你的程序使用
4. 假定点与点之间在水平和垂直方向上的距离都是 100 英尺，给出峰点距离网格左下角的英尺数。
5. 修改函数 isPeak()，使用 8 个邻接点来判断峰点，而不再只使用 4 个邻近点判断。
6. 为函数 isPeak() 添加看管条件。

8.3 二维数组和 vector 类

vector 类可以用来实现二维数组的概念。使用 vector 类比使用内建数组要更有优势。使用 vector 类允许在声明语句中使用变量来定义需要的行数和列数，因此不再需要符号常量。在 vector 类中还包含了一组用于动态确定和修改 vector 大小和容量的方法。为了定义一个二维的 vector，我们将一个指定数据类型的 vector 定义为我们需要的二维 vector 的内嵌数据类型。如下面的声明语句所示：

```
vector< vector<double> > arr(3,4);          //3 rows, 4 columns

vector< vector<int> > table(nRows, nCols); //nRows, nCols

vector< vector<char> > tags;               //empty is OK
```

注意在每条声明语句中最后两个 >> 字符之间的空格。为了编译成功，必须添加这个空格。

二维 vector 声明： vector 类可以用于实现二维数组的概念。要使用 vector 类，程序中必须包含编译器指令 #include<vector>。

语法

```
vector<vector<type specifier> > identifier(size, size));
                                  ↑
                              需要空格
```

示例

```
vector<vector<double> > temp(rows,cols);
vector<vector<int> > table(10,4);
vector<vector<char> > tags;
vector<vector<Point> > image(height,width);
```

为了说明如何使用 vector 类来实现一个二维数组，我们将编写一个类似于程序 chapter8_3 的程序，然后比较这两个程序。下面的程序调用了函数 input2DVec() 来读取文件 engine.dat 中的数据，并将数据置于一个 vector，然后调用 columnAvgVec() 来计算 vector 中每列的平均值。

```
/*                                                          */
/* Program Chapter8_5                                       */
/* This program illustrates the use of the vector           */
/* class to implement a two dimensional array               */
```

```
#include<iostream>  //Required for cout
#include<fstream>   //Required for ifstream
#include<vector>    //Required for vector
using namespace std;

//Function Prototypes
void input2DVec(istream&in,vector<vector<double> >&arr);
double columnAvgVec(vector<vector<double> >arr,int cNum);

int main()
{
   //Open input file
   ifstream fin("engine.dat");
   if(fin.fail())
   {
     cerr << "Could not open file engine.dat" << endl;
     exit(1);
   }
   //File open successful, declare objects
   int rows, cols;
   double colAvg;
   fin >> rows >> cols;


   //Define vector of vectors of type double
   vector< vector<double> > temps(rows,cols);

   //Call function to input data
   input2DVec(fin,temps);

   //Print the average value for each column
   for(int j=0; j<cols; ++j)
   {
     cout << "The average value of column " << j << " is "
          << columnAvgVec(temps,j) << endl;
   }
   return 0;
}
/*                                                          */

/*-----------------------------------------------------------*/
/* This function reads data from an input stream and        */
/* assigns the data to the 2D vector, arr.                  */
/* Pre-conditions:                                          */
/*   The istream has been defined.                          */
/*   istream source has sufficient data.                    */
/* Post-conditions:                                         */
/*   The vector arr is filled                               */
void input2DVec(istream&in,vector<vector<double> >&arr)
{

   //Declare and initialize local objects
   double val;
   int rows = arr.size();    //row size
   int cols = arr[0].size(); //all rows have same col size

   //Fill the array
   for (int i=0; i<rows; ++i)
   {
      for (int j=0; j<cols; ++j)
      {
```

```
        in >> arr[i][j];
      }
    }
}
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/* This function returns the average of column colNum                */
double columnAvgVec(vector<vector<double> >arr, int colNum)
{
  //Declare and initialize local objects.
  double avg = 0.0;
  int r = arr.size();
  //Sum all values in column colNum
  for(int i=0; i<r; ++i)
  {
    avg += arr[i][colNum];
  }

  //Return the average
  return (avg/r);
}
/*-------------------------------------------------------------------*/
```

数据文件 engine.dat 中包含下面的数据：

```
4 3
333.6 339.4 421.7
299.6 332.4 340.5
320.1 329.8 329.3
322.1 361.4 401.9
```

程序 chapter8_4 生成的输出如下：

```
The average value of column 0 is 319.35
The average value of column 1 is 340.75
The average value of column 2 is 375.85
```

首先，我们比较数组 temps 的类型声明语句。

```
double temps[MAXROWSIZE][MAXCOLSIZE];
vector< vector<double> > temps(rows,cols);
```

我们看到在程序 chapter8_3 中使用符号常量来定义数组 temps。程序 chapter8_5 则不需要符号常量。行数和列数都从数据文件中读取得到，并直接用于定义二维 vector temps。

函数参数

当 vector 作为参数传递给函数时，默认是按照值传递的。如果函数希望修改 vector 参数的值，则需要按照引用传递。函数 input2Dvec() 从输入流中读取数据，并将值赋给 vector arr。在程序 chapter8_5 中调用该函数时，vector temps 被作为参数传递给该函数。我们希望赋给形参 arr 的值也被赋给参数 temps，因此需要按引用传递。现在我们比较两个输入函数的函数原型：

```
void input2D(istream& in, double a[][MAXCOLSIZE], int, int);
void input2Dvec(istream& in,vector<vector<double> >&arr);
```

注意形参 arr 前使用了“&”，而形参 a 前则没有使用。这是因为在 C++ 中数组总是按照引用传递的。还可以看到在函数 input2D() 中需要两个额外的参数，因为在 vector 类中包

含了返回 vector 大小的方法，所以在函数 input2DVec() 中不需要传递额外的信息。函数将通过调用 vector 参数的方法 vector::size() 来获取这些信息。

最后，我们比较计算列平均值的两个函数。

```
double columnAvg(const double a[][MAXCOLSIZE], int colNum,
                 int rows);
double columnAvgVec(vector<vector<double> >arr,int cNum);
```

注意在形参 a 前加上了 const 限定符以防止该参数被修改，但是在形参 arr 前则没有，因为 vector 对象的默认参数传递方式是按值传递。

按值传递会创建参数的一份副本。在程序 chapter8_5 中，vector 的大小比较小，所以创建该参数的副本不需要花费太多额外的空间或时间。但是，如果我们要处理一个很大的 vector，那么传递 vector 的地址将更有效率，同时使用 const 限定符防止对参数的修改，如下所示：

```
double columnAvgV2(const vector<vector<double> > &arr,
                   int cNum);
```

如果使用上面的原型，程序 chapter8_5 中对该函数的调用不需要修改。但是，函数头必须修改成与函数原型一致。

练习

写出下面每个代码段的内存快照：

1.
```
int r(3), c(5);
vector<vector<char> > rags(r,c);
```

2.
```
vector<vector<int> > series(4,4);
```

3. 编写一个函数找出并返回一个二维 vector 中指定列中的最大值，包括前置和后置条件。使用下面的函数原型：

```
double maxColumnVal(vector<vector<double> > v,
                    int colNum);
```

4. 编写一个函数找出并返回一个二维 vector 中指定列中的最小值，包括前置和后置条件。使用下面的函数原型：

```
double minColumnVal(const vector<vector<double> >
                    &v, int rowNum);
```

8.4 矩阵

矩阵（matrix）是一个分布在矩形网格中的数的集合，下面是一个 4 行 3 列的矩阵，其大小可以记为 4 × 3：

$$A=\begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & 1 \end{bmatrix}$$

注意矩阵中的值被写在一个大的方括号中。

在数学记号中，矩阵通常用大写的粗体字母表示。为了引用矩阵中的某个元素，需要使用行号和列号，这里行号和列号都是从 1 开始计数。在形式化的数学记号中，大写字母代表整个矩阵，带有下标的小写字母表示特定的元素。因此，在矩阵 A 中，$a_{3,2}$ 的值为 −2。如果

矩阵的行数和列数相同，则矩阵为方阵（square matrix）。

矩阵可以使用二维数组来存储，但是我们在将矩阵记号转换成C++语句时必须小心，因为矩阵的下标和数组的下标有差异。矩阵记号假定行号和列号都是从1开始的，而C++语句中假定二维数组的行号和列号是从0开始的。

8.4.1 行列式

矩阵的行列式（determinant）是使用整个矩阵计算所得的一个值。行列式在工程中有广泛的应用，包括计算倒数和解联立方程。对于一个2×2的矩阵A，其行列式被定义为：

$$|A| = a_{1,1}a_{2,2} - a_{2,1}a_{1,2}$$

因此，下面矩阵的行列式为8：

$$A = \begin{bmatrix} 1 & 3 \\ 1 & 5 \end{bmatrix}$$

对于3×3矩阵A，行列式定义如下：

$$|A| = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} - a_{1,1}a_{2,3}a_{3,2} - a_{1,2}a_{2,1}a_{3,3} - a_{1,3}a_{2,2}a_{3,1}$$

如果矩阵A如下

$$A = \begin{bmatrix} 1 & 3 & 0 \\ -1 & 5 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

那么$|A|$等于5+6+0−0−4−(−3)，即10。

数组a存储的矩阵如下：

	col 0	col 1	col 2
row 0	1	3	0
row 1	−1	5	2
row 2	1	2	1

那么a的行列式的计算等式为：

```
determinate = a[0][0]*a[1][1]*a[2][2]-a[0][1]*a[1][2]*a[2][0]-a[0][2]*a[1][0]*a[2][1]
              -a[2][0]*a[1][1]*a[0][2]-a[2][1]*a[1][2]*a[0][0]-a[2][2]*a[1][0]*a[0][1]
            = 5-6+0-0-4-(-3)
            = 10
```

需要重点注意的是矩阵的下标从1开始，而数组的偏移量从0开始。

要计算超过3行3列的矩阵的行列式需要更多的计算过程。在本章结尾将会对这一过程进行讨论。

8.4.2 转置

矩阵的转置（transpose）是一个新的矩阵，新矩阵的行是原矩阵的列，而新矩阵的列则是原矩阵的行。我们在矩阵名的后面使用上标T来表示矩阵的转置。例如，考虑下面的矩阵和它的转置：

$$B=\begin{bmatrix}2 & 5 & 1\\ 7 & 3 & 8\\ 4 & 5 & 21\\ 16 & 13 & 0\end{bmatrix},\quad B^{\mathrm{T}}=\begin{bmatrix}2 & 7 & 4 & 16\\ 5 & 3 & 5 & 13\\ 1 & 8 & 21 & 0\end{bmatrix}$$

如果我们考虑一对元素，可以看到在位置（3, 1）的值现在已经移到了位置（1, 3）。在位置（4, 2）的值现在已经移动到了位置（2, 4）。事实上，我们是将行和列的编号进行了交换，这样就将位置（i, j）的值移动到了位置（j, i）。同时，还可以看到转置后的大小与原矩阵的大小是不同的，除非原矩阵是一个方阵（即行数和列数相同的矩阵）。

现在我们开发一个函数来计算矩阵的转置。函数的形参中必须包含代表原矩阵和转置的二维数组。为了给函数增加一些灵活性，我们假定原矩阵的行数和列数通过符号常量进行定义，这些符号常量为 NROWS 和 NCOLS。该函数没有返回值，因此返回类型为 void，注意，在使用该函数的程序中必须包含 NROWS 和 NCOLS 的定义：

```
/*-------------------------------------------------*/
/*  This function generates a matrix transpose.    */
/*  NROWS and NCOLS are symbolic constants         */
/*  that must be defined in the calling program.   */

void transpose(int b[][NCOLS], int tr[][NROWS])
{
   //  Declare objects.

   //  Transfer values to the transpose matrix.
   for (int i=0; i<=NROWS-1; ++i)
   {
      for (int j=0; j<=NCOLS-1; ++j)
      {
         tr[j][i] = b[i][j];
      }
   }

   //  Value return.
   return;
}
/*-------------------------------------------------*/
```

8.4.3 矩阵加法和减法

两个矩阵的加法（或减法）是将矩阵中对应位置的元素相加（或相减）。因此，进行加减的矩阵必须具有相同的大小，操作的结果是另一个大小相同的矩阵。考虑下面的矩阵：

$$A=\begin{bmatrix}2 & 5 & 1\\ 0 & 3 & -1\end{bmatrix},\quad B=\begin{bmatrix}1 & 0 & 2\\ 1 & 4 & -2\end{bmatrix}$$

矩阵的和与差如下：

$$A+B=\begin{bmatrix}3 & 5 & 3\\ -1 & 7 & -3\end{bmatrix},\quad A-B=\begin{bmatrix}1 & 5 & -1\\ 1 & -1 & 1\end{bmatrix},\quad B-A=\begin{bmatrix}-1 & -5 & 1\\ -1 & 1 & -1\end{bmatrix}$$

8.4.4 矩阵乘法

矩阵乘法（matrix multiplication）不是将矩阵对应位置的元素相乘。矩阵 A 和 B 的乘积

中位置 $c_{i,j}$ 的值是第一个矩阵的第 i 行和第二个矩阵的第 j 列的乘积：

$$c_{i,j}=\sum_{k=1}^{n} a_{i,k}b_{k,j}$$

第 i 行和第 j 列相乘要求第 i 行和第 j 列的元素数目相等。因此，第一个矩阵（A）每行的元素数目必须与第二个矩阵（B）每列的元素数目相等。因此，如果 A 和 B 都为 5 行 5 列，那么它们的乘积也是 5 行 5 列。进一步，我们可以计算 AB 和 BA。但一般来说它们是不相等的。

如果 A 为 2 行 3 列，B 为 3 行 3 列，那么乘积 AB 将为 2 行 3 列。为了说明，考虑下面的矩阵：

$$A=\begin{bmatrix}2 & 5 & 1\\ 0 & 3 & -1\end{bmatrix},\quad B=\begin{bmatrix}1 & 0 & 2\\ -1 & 4 & -2\\ 5 & 2 & 1\end{bmatrix}$$

乘积 $C=AB$ 中的第一个元素为

$$\begin{aligned}c_{1,1}&=\sum_{k=1}^{3} a_{1,k}b_{k,1}\\ &=a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,3}b_{3,1}\\ &=2(1)+5(-1)+1(5)\\ &=2\end{aligned}$$

类似地，我们可以计算出 A 与 B 乘积中的其他元素：

$$AB=C=\begin{bmatrix}2 & 22 & -5\\ -8 & 10 & -7\end{bmatrix}$$

在本例中，我们无法计算 BA，因为 B 的每行中的元素数目与 A 的每列中的元素数目不相等。

一种用于确定一个矩阵乘积是否存在的简单方法是按照边对边的方式写出最后的乘积大小。如果靠近里面的两个数相等，那么乘积存在；乘积的大小已靠近外面的两个数决定。为了说明这种方法，考虑前一个例子中 A 的大小为 2×3，B 的大小为 3×3。因此，如果我们想要计算 AB，可以按照边对边写出大小：

$$2\times3\quad 3\times3$$

内侧的两个数都是 3，所以 AB 存在。其大小则由外侧的两个数决定，为 2×3。如果我们希望计算 BA，再次按照边对边写出各自的大小：

$$3\times3\quad 2\times3$$

内侧的两个数不同，所以 BA 不存在。

现在我们给出一个计算乘积 $C=AB$ 的函数。在该函数中，数组的大小都为 $N\times N$，其中 N 是一个符号常量：

```
/*-------------------------------------------------------*/
/*  This function performs a matrix multiplication       */
/*  of two NxN matrices using sums of products.          */
/*  N is a symbolic constant that must be defined        */
/*  within the scope of the function.                    */
```

```
void matrixMult(int a[][N], int b[][N], int c[][N])
{

   // Compute sums of products.
   for (int i=0; i<N; ++i)
   {

      for (int j=0; j<N; ++j)
      {
         a[i][j] = 0;
         for (int k=0; k<N; ++k)
         {
            c[i][j] += a[i][k]*b[k][j];
         }
      }
   }
   // Void return.
   return;
}
/*-------------------------------------------------------*/
```

练习

在接下来这些问题中使用笔算完成有关矩阵 A、B、C 的表达式。如果表达式可计算，则使用本节开发的函数编写程序测试你的答案。这些函数在教师资源的文件 matrix.cpp 中可以找到。

$$A=\begin{bmatrix}2 & 1\\ 0 & -1\\ 3 & 0\end{bmatrix},\ B=\begin{bmatrix}-2 & 2\\ -1 & 5\end{bmatrix},\ C=\begin{bmatrix}3 & 2\\ -1 & -2\\ 0 & 2\end{bmatrix}$$

1. $|B$
2. C^T+A^T
3. $A-B$
4. AB
5. BA
6. $B(C^T)$
7. $(CB)C^T$

在本章结尾的习题中，我们将使用本节所讨论的矩阵操作来定义其他的矩阵操作。

8.5 数值方法：解联立方程

在工程问题中经常会有解联立方程的需求。解联立方程有很多方法，每种方法都有各自的优点和缺点。本节我们将介绍用于解联立线性方程（simultaneous linear equation）的高斯消元法（Gauss elimination）。这里的方程称作线性方程（linear equation）是因为方程中只包含线性（一维）项，如 x、y 和 z。在我们给出该方法的细节之前，首先给出关于方程的解的图形分析。

8.5.1 图形分析

一个带有两个变量的线性方程，如 $2x-y=3$，定义了一条直线，通常写作 $y=mx+b$ 的形式，其中 m 代表直线的斜率，b 代表 y 轴截距。因此，$2x-y=3$ 也可以写作 $y=2x-3$。如果我们有两个线性方程，那么它们可以表示成两条交于一点的直线，或者两条永不相交的平行线，或者是表示相同的一条直线，图 8.1 中画出了这三种可能的情况。

表示成两条相交直线的方程可以很容易地分辨出来，因为它们的斜率不同，如 $y=2x-3$ 和 $y=-x-3$。

表示成两条平行线的方程有相同的斜率，但是 y 轴上的截距不同，如 $y=2x-3$ 和 $y=2x+1$。表示成同一直线的方程的斜率和 y 轴上的截距都相同，如 $y=2x-3$ 和 $3y=6x-9$。

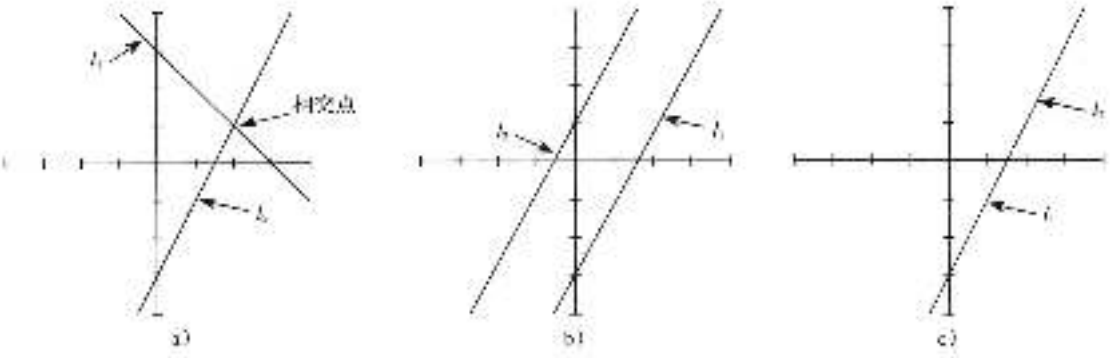


图 8.1 两条直线

如果一个线性方程包含三个变量 x、y 和 z，那么它代表着三维空间中的一个平面。如果我们有两个带三个变量的方程，那么它们可以表示成两个平面，这两个平面可能相交于一条直线，或者相互平行，或者代表同一个平面，图 8.2 中画出了这三种可能的情况。如果我们有三个带三个变量的方程，那么这三个平面可能交于一点、交于一个面、没有交点，也可能代表同一个平面。图 8.3 中给出了三个方程表示的三个不同平面的相交关系。这种思想可以扩展到三个以上变量的情况，但是在这样的情况下很难进行图形化描述。

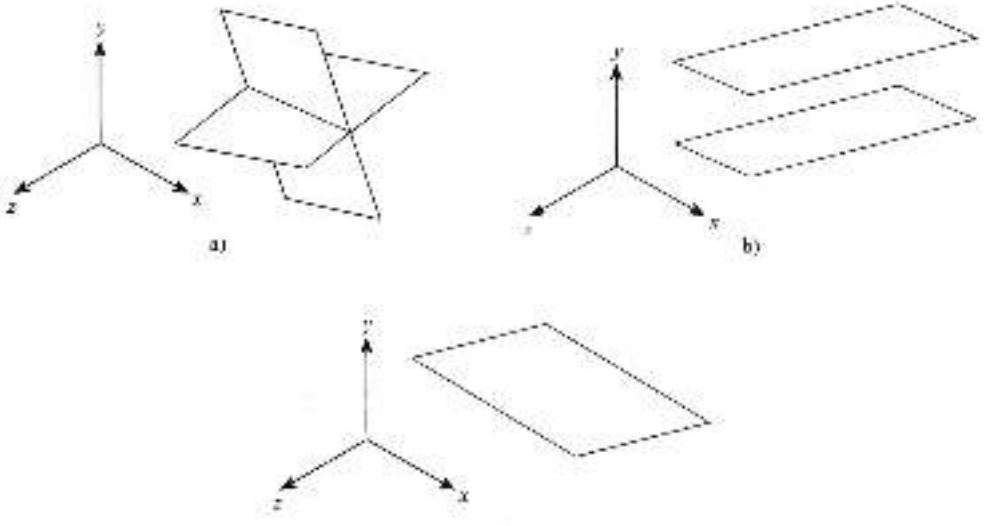


图 8.2 两个平面

我们将超过三个变量的方程所定义的点集称作超平面（hyperplane）。一般来说，考虑一组包含 n 个未知量的、由 m 个线性方程组成的方程组，其中每个方程都定义了一个不同于其他方程的超平面。如果 $m < n$，那么方程组的解是不确定的，不存在唯一解。如果 $m = n$，且没有代表着平行平面的方程，则方程组有唯一解存在。如果 $m > n$，那么方程组超定义，不存在唯一解。一组方程也称作方程组（system of equation）。有唯一解的方程组也称作非奇异（nonsingular）的方程组，无唯一解的方程组称作奇异方程组。

作为一个特例，考虑下面的方程组：

$$3x + 2y - z = 10$$

$$-x+3y+2z=5$$

$$x-y-z=-1$$

该方程组的解为点（2，5，-6），将这些值带入方程中计算，可以验证该点是方程组的解。

本节所开发的解决方案不需要依赖于前一节所提到的有关矩阵的相关材料。但是，如果你阅读了相关内容，你会发现有趣的事情，即一个线性方程组可以表示成矩阵乘法的形式。为了说明，我们将前面的方程组表示成下面的矩阵：

$$A=\begin{bmatrix}3 & 2 & -1\\ -1 & 3 & 2\\ 1 & -1 & -1\end{bmatrix},\ X=\begin{bmatrix}x\\ y\\ z\end{bmatrix},\ B=\begin{bmatrix}10\\ 5\\ -1\end{bmatrix}$$

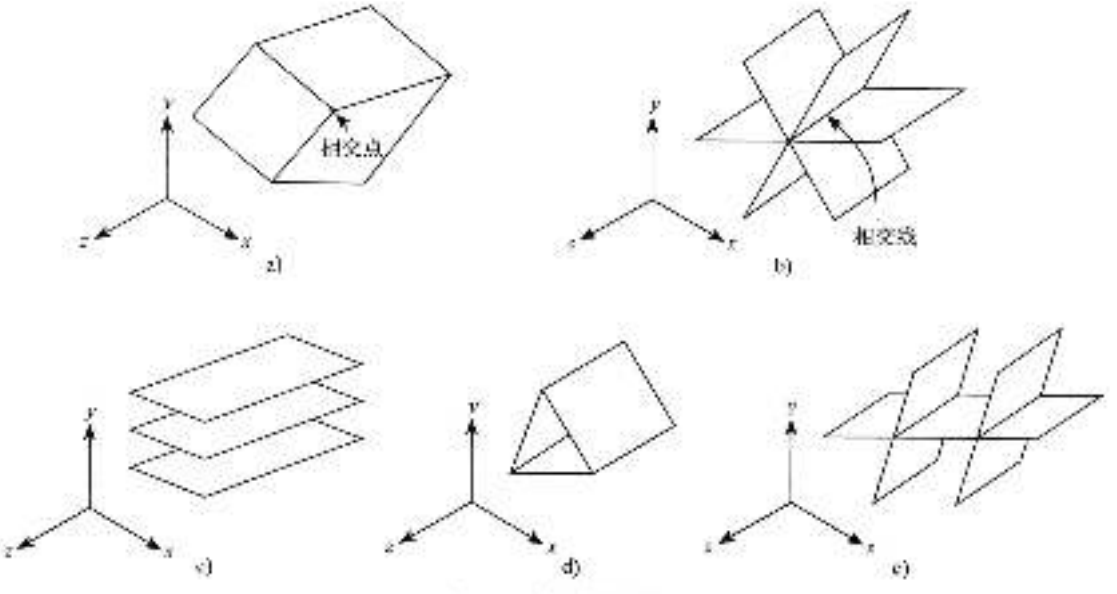


图 8.3　三个不同的平面

然后，使用矩阵乘法，我们发现方程组可以写成下面的形式

$$AX=B$$

通过上面的矩阵乘法，将会使你相信这个矩阵等式产生了原来的方程组。

在许多工程问题中，我们对于确定一个方程组是否有一个一般解都很感兴趣。如果存在一般解，那么我们希望将它计算出来。在本节的下一部分，我们给出了解一组联立线性方程的高斯消元法。

8.5.2　高斯消元法

在给出关于高斯消元法的一般描述前，我们首先对前面给出的一个特别的例子使用该方法：

$$3x+2y-z=10 \qquad \text{（第一个方程）}$$

$$-x-3y+2z=5 \qquad \text{（第二个方程）}$$

$$x-y-z=-1 \qquad \text{（第三个方程）}$$

第一步是消元（elimination），在该步骤中，在每个方程中的第一个变量将被消去。通过将第一个方程乘以一个特定系数后与其他方程相加，可以达到消元的目的。这里的项涉及变量 x，在第二个方程中则是 $-x$。因此，如果将第一个等式乘以 1/3，再与第二个等式相加，那么可以得到一个新的不含 x 的方程：

$$\begin{array}{rl} -x-3y+2z=5 & \text{（第二个方程）} \\ x+\frac{2}{3}y-\frac{1}{3}z=\frac{10}{3} & \text{（第一个方程的 1/3 倍）} \\ \hline 0x+\frac{11}{3}y+\frac{5}{3}z=\frac{25}{3} & \text{（和）} \end{array}$$

修改后的方程组为

$$\begin{aligned} 3x-2y-z&=10 \\ 0x-\frac{11}{3}y+\frac{5}{3}z&=\frac{25}{3} \\ x-y-z&=-1 \end{aligned}$$

现在我们采用类似的方法从第三个方程中消去第一个变量：

$$\begin{array}{rl} x-y-z=-1 & \text{（第三个方程）} \\ -x-\frac{2}{3}y+\frac{1}{3}z=-\frac{10}{3} & \text{（第一个方程的 -1/3 倍）} \\ \hline 0x-\frac{5}{3}y-\frac{2}{3}z=-\frac{13}{3} & \text{（和）} \end{array}$$

修改后的方程组为

$$\begin{aligned} 3x+2y-z&=10 \\ 0x+\frac{11}{3}y+\frac{5}{3}z&=\frac{25}{3} \\ 0x-\frac{5}{3}y-\frac{2}{3}z&=-\frac{13}{3} \end{aligned}$$

现在除了第一个方程外，我们已经从所有其他的方程中消去了第一个变量。

下一步是从除了第一个和第二个方程外的其他方程中消去第二个变量，通过将方程与第二个方程的倍数形式相加：

$$\begin{array}{rl} 0x-\frac{5}{3}y-\frac{2}{3}z=-\frac{13}{3} & \text{（第三个方程）} \\ 0x+\frac{5}{3}y+\frac{25}{33}z=-\frac{125}{33} & \text{（第二个方程的 -5/11 倍）} \\ \hline 0x-0y+\frac{3}{33}z=-\frac{18}{33} & \text{（和）} \end{array}$$

修改后的方程组为

$$\begin{aligned} 3x+2y-z&=10 \\ 0x+\frac{11}{3}y+\frac{5}{3}z&=\frac{25}{3} \\ 0x+0y-\frac{3}{33}z&=-\frac{18}{33} \end{aligned}$$

因为在第三个方程之后没有其他方程，所以算法部分结束了。

现在我们通过回代法（back substitution）来计算方程的解。在最后一个方程中只有一个变量，所以我们可以将方程乘以一个特定的因子使变量的系数变为 1。因此，我们将最后一个方程乘上 33/3 或者 11，得到

$$0x+0y+z=-6$$

将 z 的值替换到倒数第二个方程中，得到

$$0x - \frac{11}{3}y + \frac{5}{3}(-6) = \frac{25}{3}$$

化简方程，将所有常数项放在等式右边，得到

$$0x + \frac{11}{3}y = \frac{55}{3}$$

现在方程中只有一个变量，我们将方程乘以一个特定的因子使变量的系数变为 1：

$$0x + y = 5$$

现在我们回到下一个方程，这是本例中最后一个方程：

$$3x + 2y - z = 10$$

回代所有我们已经确定的值，得到

$$3x + 2(5) - (-6) = 10$$

或者

$$3x = -6$$

因此，x 的值为 −2。

高斯消元法包括两部分：消元和回代。首先修改方程，在所有位于第 k 个方程之后的方程中将第 k 个变量消掉。然后从最后一个方程开始，计算最后一个变量的值。接下来，使用计算出的值和倒数第二个方程，计算出倒数第二个变量。以此类推，将回代过程持续进行，直到计算出所有变量的值为止。如果某个变量的系数全部为 0 或者很接近于 0，那么这个方程组是病态的（ill conditioned）或者没有唯一解。

通过一种称作轴旋转（pivoting）的过程可以提高高斯消元法的准确性。行轴旋转要求在进行高斯消元之前对行进行重排序。列轴旋转要求在高斯消元前对列进行重排序。完整的轴旋转包括行和列的轴旋转。在本章结尾的问题中将会讨论这一过程。

练习

使用高斯消元法找出下面联立线性方程组的解：

1. $-2x + y = -3$
   $x + y = 3$

2. $3x + 5y - 2z = 8$
   $2x + 3y - z = 1$
   $x - 2y - 3z = -1$

8.6 解决应用问题：电路分析

在电路分析中经常需要求一组联立方程的解。这些方程通常来自于描述电流流入流出节点时的电流方程或者描述电路中网孔环路的电压方程。例如，考虑图 8.4 中所示的电路。描述三个封闭环路中电压的方程如下：

$$-V_1 + R_1 i_1 + R_2 (i_1 - i_2) = 0,$$
$$R_2 (i_2 - i_1) + R_3 i_2 + R_4 (i_2 - i_3) = 0,$$
$$R_4 (i_3 - i_2) + R_5 i_3 + V_2 = 0.$$

如果我们假定电阻的值（R_1，R_2，R_3，R_4 和 R_5）和电压源（V_1 和 V_2）的值已知，那么方程组中的未知量是网孔电流（i_1，i_2 和 i_3）。那么我们可以将方程组重写成下面的形式：

$$(R_1 + R_2)i_1 - R_2 i_2 - 0i_3 = V_1$$
$$-R_2 i_1 + (R_2 + R_3 + R_4)i_2 - R_4 i_3 = 0$$
$$0i_1 - R_4 i_2 + (R_4 + R_5)i_3 = -V_2$$

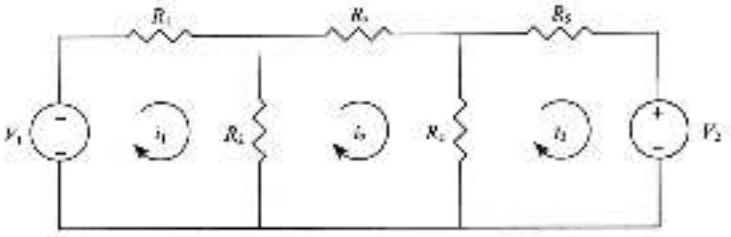


图 8.4 带有两个电压源的电路

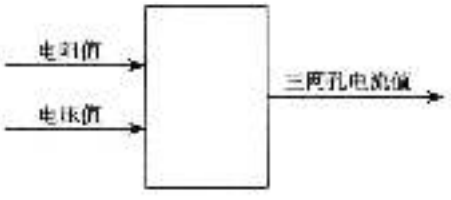
编写一个程序，允许用户输入五个电阻的值和两个电压源的值，并将这些值存储在一个二维数组中。程序应当计算出三个网孔电流值。

1. 问题描述

计算图 8.4 中所示电路的网孔电流。

2. 输入/输出描述

下面的 I/O 示意图给出了程序的输入和输出，输入是电阻值和电压值，输出为三个网孔电流值。



3. 用例

通过使用电阻值和电压值，可以定义一个含有三个方程的方程组，根据问题的定义，可以得到下面的方程组：

$$(R_1 + R_2)i_1 - R_2 i_2 + 0i_3 = V_1$$
$$-R_2 i_1 + (R_2 + R_3 + R_4)i_2 - R_4 i_3 = 0$$
$$0i_1 - R_4 i_2 + (R_4 + R_5)i_3 = -V_2$$

例如，假定每个电阻的值为 1 欧姆，每个电压源都是 5 伏特，那么对应的方程组如下：

$$2i_1 - i_2 + 0i_3 = 5$$
$$-i_1 + 3i_2 - i_3 = 0$$
$$0i_1 - i_2 + 2i_3 = -5$$

一旦方程组确定，就可以使用前一节中的方法来解此用例中的方程组。对于该方程组，其解为 $i_1 = 2.5$，$i_2 = 0$，$i_3 = -2.5$。

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。

分解提纲

1）读取电阻值和电压值；

2）确定方程组的系数；

3）使用高斯消元法计算电流值；

4）打印出电流值。

在步骤1中我们要读取用于确定电路的信息，在步骤2中使用这些信息来确定方程组的系数。然后在步骤3中，我们完成消元和回代的详细实现。为了让main函数短小易读，可以为消元和回代步骤各编写一个函数。解决方案的结构图在图6.1中给出。

联立方程组的系数存储在一个二维数组中，解存储在一个一维数组中。变量index用于指出哪个变量正在进行消元，该变量的值域为0～$n-1$，以与C++的下标相对应。

使用伪代码来描述高斯消元法比较困难，因为在算法中必须指出具体的下标。仔细检查下面基于用例的伪代码，确信自己能够轻松地处理下标操作。

```
Refinement in Pseudocode
main:     read resistor values and voltage values
          specify array coefficients, a[i,j]
          set index to zero
          while index <= n - 2
                eliminate(a,n,index)
                increment index by 1
          back_substitute(a,n,soln)
          print current values
eliminate(a,n,index):
          set row to index +1
          while row <= n-1
                set scale_factor to -a[row][index] / a[index, index]
                set a[row][index] to zero

                set col to index +1
                while col <= n
                      add a[index][col] * scale_factor
                            to a[row][col]
                      increment col by 1
                increment row by 1
back_substitute(a,n,soln):
          set soln[n-1] to a[n-1][n] / a[n-1][n-1]
          set row to n-2
          while row >= 0
                set col to n-1
                while col >= row +1
                      subtract soln[col] * a[row][col]
                            from a[row][n]
                      subtract 1 from col
                set soln[row] to a[row][n] / a[row][row]
                subtract 1 from row
```

条理清晰、逻辑合理的伪代码可以相对直接地转换为C++代码：

```
/*-------------------------------------------------------*/
/*  Program chapter6_5                                   */
/*                                                       */
/*  This program uses Gauss elimination to               */
/*  determine the mesh currents for a circuit.           */

#include <iostream>   //Required for cin, cout
using namespace std;
```

```
// Define global constant for number of unknowns.
const int N   3;

//  Declare function prototypes.
void eliminate(double a[][N+1], int n, int index);
void back substitute(double a[][N+1],
                     int n, double soln[N]);
int main()
{
  //  Declare objects.
  double r1, r2, r3, r4, r5, v1, v2,
         a[N][N+1], soln[N];

  //  Get user input.
  cout << "Enter resistor values in ohms: \n"
       << "(R1, R2, R3, R4, R5): \n";
  cin >> r1 >> r2 >> r3 >> r4 >> r5;
  cout << "Enter voltage values in volts: \n"
       << "(V1, V2): \n";
  cin >> v1 >> v2;

  //  Specify equation coefficients.
  a[0][0] = r1 + r2;
  a[0][1] = a[1][0] = -r2;
  a[0][2] = a[2][0] = a[1][2] = 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;

  //  Perform elimination step.
  for (int index=0; index<N ; index++)
  {
    eliminate(a,N,index);
  }

  //  Perform back substitution step.
  back substitute(a,N,soln);

  //  Print solution.
  cout << "\nSolution: \n";
  for (int i=0; i<N; ++i)
  {
    cout << "Mesh Current " << i+1 << ": " << soln[i] << endl;
  }

  //  Exit program.
  return 0;
}
/*-------------------------------------------------------*/
/*  This function performs the elimination step.         */
void eliminate(double a[][N+1], int n, int index)
{
   //  Declare objects.
   double scale_factor;

   //  Eliminate object from equations.
```

```
    for (int row=index+1; row<n; ++row)
    {
       scale_factor = -a[row][index]/a[index][index];
       a[row][index] = 0;
       for (int col=index+1; col<=n; ++col)
       {
          a[row][col] += a[index][col]*scale_factor;
       }
    }

    // Void return.
    return;
}
/*------------------------------------------------------------*/
/* This function performs the back substitution.              */

void back_substitute(double a[][N+1], int n,
                     double soln[]);
{
    // Perform back substitution in each equation.
    soln[n-1] = a[n-1][n]/a[n-1][n-1];
    for (int row=n-2; row>=0; --row)
    {
       for (int col=n-1; col>=row+1; --col)
       {
          a[row][n] -= soln[col]*a[row][col];
       }
       soln[row] = a[row][n]/a[row][row];
    }

    // Void return.
    return;
}
/*------------------------------------------------------------*/
```

为了处理更大的方程组，必须修改符号常量 N 的值，高斯消元法的步骤则不需要任何修改。使用用例中的数据与程序交互可以得到下面的输出：

```
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: 2.5
Mesh Current 2: 0
Mesh Current 3: -2.5
```

程序假定方程组有一个解，这意味着没有方程是同样的方程或者平行的方程。可以在程序中添加对于这些条件进行检测的语句或函数。

修改

使用本节开发的程序来回答下面的问题：

1. 如果电阻都是 5 欧姆，电源都是 10 伏特，确定网孔电流。
2. 使用本节讨论的矩阵乘法来验证你在问题 1 中的答案。（这个问题假定你已经阅读了前一节有关矩阵和 vector 的内容。）
3. 如果电阻值分别为 2、8、6、6 和 4 欧姆，电压源值的为 40 和 20 伏特，计算网孔电流。
4. 使用回代法对问题 3 中的方程组的解进行验证。

8.7 高维数组

C++允许定义超过二维以上的数组。例如，下面的语句定义了一个三维数组：

```
int b[2][4][2];
```

要引用某个元素，必须指定三个偏移量，如果你将数组映射到图8.5所示的三维空间中，每个偏移量分别对应于x轴、y轴和z轴坐标。因此，图中阴影部分的位置对应于b[2][0][1]。

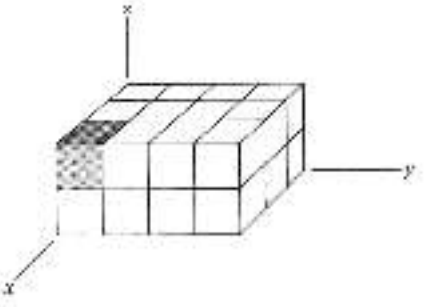


图8.5 三维数组

大部分工程问题需要的数组都可以使用一维或者二维数组来满足要求。但是，偶尔也有问题适合使用高维数组来解决。这些问题一般都涉及由多个参数指定的数据；此外，参数要么是顺序值，要么是可以转换成顺序形式的参数。例如，假定有一组采集自大型化学反应容器底部的温度测量值的数据。此外，这组数据是在化学反应期间以一定间隔采集的。在这种情况下，我们可以选择使用三维数组，使用第一维偏移量指示时间，其他两维指示底部的位置。为了满足C++关于偏移量的要求，偏移量需要从0开始，那么偏移量[3][2][5]表示在第4个时间点上，在位置[2][5]上的温度。

二维以上的数组很少使用，因为它们很难被可视化表示。但是，可以使用一种简单的方式对二维以上的数组进行可视化。首先，将一个三维数组看做一个建筑，建筑有楼层，每层中有矩形方格式的房间。假定每个房间都包含一个宝。那么代表一个建筑的三维数组使用三个偏移量来指定一个房间：第一个偏移量是楼层号，其他两个偏移量指明了房间在指定楼层中的行号和列号。

那么一个四维数组就是像图8.6中所示的一行建筑。第一个偏移量指明建筑，剩下的三个偏移量指明建筑中的房间。

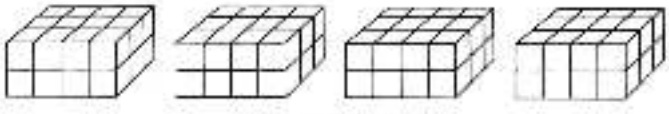


图8.6 四维数组

一个五维数组就是像图8.7中所示的一块建筑群。前两个偏移量指明在块中的建筑，剩下的三个偏移量指明建筑中的房间。

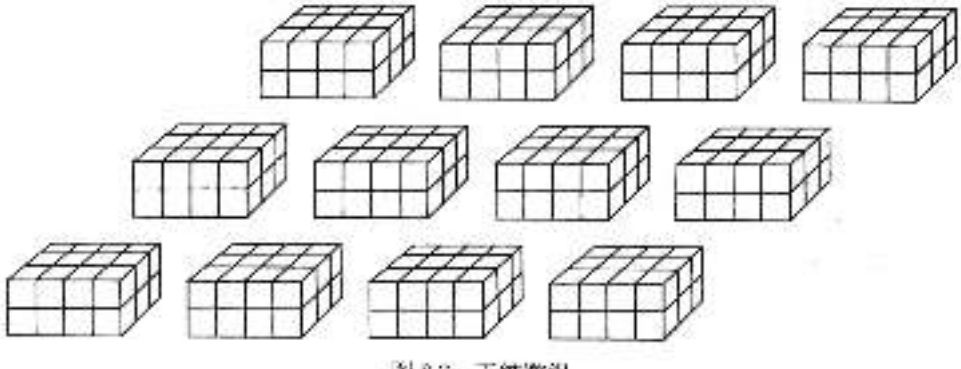


图8.7 五维数组

这种类比可以继续到一行建筑块、一个城市的建筑块、一组城市、一个州的城市等。虽然我们已经向你展示了如何对高维数组进行可视化，但是还是要提醒你注意高维数组的使用。高维数组有许多涉及偏移量的计算；每当次处理数组中的值时，不仅需要额外的偏移量，还需要额外的循环。一般而言，高维数组在程序调试和维护时也更复杂。为此，只有在高维数组可以被简化成完全可视化的问题和步骤时才应当使用它们。

本章小结

数组是一种能用于存储程序分析所需工程数据的数据结构。如果数据的最自然表示方式是表格或者网格，则可以使用二维数组。本章开发了许多例子用于说明有关二维数组的定义、初始化、计算以及输入、输出等，同时也包括了二维数组作为函数参数的情况。还介绍了用于解决线性方程组的高斯消元法，并使用C++程序实现了该方法。在对二维数组建模的例子中使用了vector类。

关键术语

column offset（列偏移量）
declared column size（声明列大小）
determinant（行列式）
Gauss elimination（高斯消元法）
hyperplane（超平面）
ill conditioned（病态的）
inner product（内积）
matrix（矩阵）
matrix multiplication（矩阵乘法）
nonsingular（非奇异的）
pivoting（轴旋转）
post-condition（后置条件）
pre-condition（前置条件）
row offset（行偏移量）
simultaneous linear equations（联立线性方程）
square matrix（方阵）
subarray（子数组）
systems of equations（方程组）
transpose（转置）
two-dimensional array（二维数组）
two-dimensional vector（二维向量）

C++ 语句总结

二维数组声明

```
double x[10][5];
```

注意事项

1. 使用符号常量声明数组的大小，这样易于修改。
2. 在文档描述中，将二维数组描述为由行、列组成的网格。
3. 对象i和j通常用于表示一个二维数组的偏移量。
4. 在函数定义中包含前置条件和后置条件。

调试要点

1. 在函数原型中形参列表中必须声明数组的列的大小。
2. 在引用一个多维数组中的元素时要注意不要超过最大的偏移量。
3. 在引用多维数组中的元素时，确保每个偏移量都在自己的一对括号中。
4. 在将矩阵记号转换成C++时，记住矩阵中的第一行和第一列的标号都是从1开始，而不是从0开始。
5. 多维矩阵会让程序变得变得复杂，只有在需要使用的时候才使用它们。
6. 在声明一个二维vector时，记住在类型声明语句中最后两个“>>”字符之间要有空格。

习题

给出下面每组语句执行后对应的内存快照，使用“?”表示未被初始化的数组元素：

1.
```
int x[4][5];
...
for(int r=0; r<3; ++r)
  for(int c=0; c<4; ++c)
    x[r][c] = r * c;
```

2.
```
int x[3][5];
for(int r=0; r<=4; ++r)
  for(int c=0; c<r; ++c)
    x[r][c] = 7;
```

程序输出

问题3和4与下面的语句有关：

```
int sum, k, i, j;
int x[4][5] = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},{16,17,18,19,20}};
```

3. 给出下面语句执行后 sum 的值。

```
sum = x[2][0];
for(int k = 1; k<5; ++k)
  sum += x[2][k];
```

4. 给出下面语句执行后 sum 的值。

```
sum = 0;
for(i=1; i<=3; ++i)
  for(int j=0; j<=3; ++j)
    if(x[i][j] > x[i-1][j+1])
      sum++;
```

5. 给出下面语句执行后 size 的值。

```
vector<vector<double>> power(5,6);
int size = power.size();
```

6. 给出下面语句执行后 size 的值。

```
vector<vector<int>> power(5,10);
int size = power[0].size();
```

多选题

7. 在下面的类型声明语句执行后，A[1][2]被赋予的值是（　　）。

```
int A[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

(a) 2　　(b) 3

(c) 4　　(d) 5

(e) 6

8. 在下面的类型声明语句执行后，B[2][2]被赋予的值是（　　）。

```
int B[3][3] = {{1, 2}, {3, 4}, {5, 6}};
```

(a) 0　　(b) 2

(c) 4　　(d) 6

(e) B[2][2]的值未定义

9. 下面关于返回一个整型二维数组中第 n 行最大值的函数原型中，（　　）是正确的。

(a) `void fun(int[][], int n, int largest);`

(b) `int fun(const int a[][COLSIZE]);`

(c) `int fun(const int a[][COLSIZE], int n);`

(d) int fun(const int [n][COLSIZE], int n);
(e) void fun(const int[][], COLSIZE, int n);

10. 下面关于将一个整型二维数组中每个元素的值减去 n 的函数原型中，(　　) 是正确的。
(a) void fun(int[][], int, n);
(b) int fun(const int a[][COLMAX], int n);
(c) void fun(const int a[][n], int COLSIZE);
(d) void fun(int a[][COLSIZE], int n);
(e) int fun(const int[][], int n);

编程题

发电数据。数据文件 power1.dat 中包含了 10 周内的发电输出数据，单位为兆瓦。每行数据包含 7 个浮点数，代表着 1 周内的数据。在开发下面的程序时，使用符号常量 NROWS 和 NCOLS 来代表存储数据的数组的行数和列数。

11. 编写一个程序计算并打印出这段时间内的平均发电输出量，同时打印出超过平均发电量的天数。
12. 编写一个程序打印出发电量最低的那天所在的周编号和日编号。如果有若干天的发电量都是最低，则将这若干天都打印出来。
13. 编写一个函数计算一个具有 NROWS 行和 NCOLS 列的二维数组中指定列的平均值。函数的参数应当是浮点数组和所要计算的列。假定相应的函数原型如下：

```
double col_ave(double a[][NCOLS],int col);
```

14. 编写一个函数计算并返回一个 double 类型的一维 vector 的平均值。假定相应的函数原型如下：

```
double avgVec(const vector<vector<double> > &v);
```

15. 编写一个程序，打印出报告，列出每周中从第一天一第七天中每天的平均发电量。打印信息的格式如下：

```
Day x: Average Power Output in Megawatts: xxxx.xx
```

16. 编写一个函数计算一个拥有 NROWS 行和 NCOLS 列的二维数组中指定行的平均值。函数的参数应当是浮点数组和所要计算的行。假定相应的函数原型如下：

```
double row_ave(double a[][NCOLS],int row);
```

17. 编写一个程序，打印出报告，列出第一周一第十周中每周的平均发电量。打印信息的格式如下：

```
Week x: Average Power Output in Megawatts: xxxx.xx
```

18. 编写一个程序，计算并打印出发电输出量数据的平均值和方差。

温度分布。在一个等温的薄金属板上，其每边的温度分布都可以使用图 8.8 所示的二维网格进行建模。一般来说，在网格中指定的点的四边都具有相同的温度。而靠近内部区域的温度通常被初始化为 0，但是它们的温度会随着周围点的温度而变化。假定内部的一个点的温度值可以通过其周围 4 个相邻点的温度的平均值计算得到：图 8.8 中阴影部分所示的点表示网格中标记为 x 的点的邻接点。每当内部的某个点温度变化时，其邻接点的温度也会变化。这种变化一直持续到达到热平衡为止，此时所有的温度都保持相同。

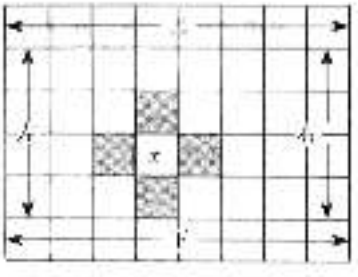


图 8.8　一个金属板上的温度网格

19. 编写一个程序，对一个有 6 行 8 列网格的温度分布建模。允许用户输入四边的温度，使用一个网格来存储温度。因此，当点更新时，它的新值将用于更新下一个点。按行移动，持续对点进行更新，直到所有的更新值之间的温度差异小于一个用户指定的容忍值为止。使用 vector 来实现网格。
20. 修改问题 19 中生成的程序，使更新按列进行。比较两个程序使用不同的容忍值时所得到的平衡

值。平衡值应当十分接近于小的容忍值。

21. 修改习题 19 中的程序，使用两个网格，使得程序对网格进行更新时看起来每个点的更新都是同时发生的。因此，所有的温度都使用一组网格值来进行更新。在这里需要使用两个网格，以保证在计算每个新的温度时所有的旧温度信息都是可用的。

22. 修改习题 19 中的程序，使用数组而不是 vector 类来实现网格。

高斯消元法。高斯消元法的奇异性可以使用一个称作轴旋转的过程来克服。为了完成行轴旋转，我们首先对方程进行重排序，使得第一个系数的绝对值最大的方程变为第一个方程。然后我们从第一个方程后面的方程中消去第一个变元。然后从第二个方程开始，使得第二个系数的绝对值最大的方程变为第二个方程。然后我们从第二个方程后面的方程中消去第二个变元。一直按照该过程进行下去，消去其他变元。假定方程中所包含的变元数为 N。

23. 使用 8.4 节中开发的程序作为基本，开发一个函数，接收一个大小为 $N\times(N+1)$ 的 double 类型数组 a 为第一个参数。第二个参数是一个大小为 N+1 的 double 类型数组 soln。函数应当解出数组 a 所代表的方程组，将解存在数组 soln 中。假定相应的函数原型为

```
void gauss(double a[][N+1], double soln[N]);
```

24. 编写一个函数，接收一个二维数组和一个由相关系数 j 标明的主元素。函数应当从第 j 个方程开始重新对所有方程进行排序，并使得第 j 个方程在第 j 个位置上的系数绝对值最大。假定函数引用的数组的大小为 $N\times(N+1)$，并且相应的函数原型为：

```
void pivot_r(double a[][N+1], int j);
```

25. 修改问题 23 中的函数，使得行轴旋转在每个变量消元之前进行。使用问题 24 中定义的函数。

26. 列轴旋转与行轴旋转的方式类似，通过对列进行交换，使得绝对值最大的系数在要求的位置上。当列被交换时，需要对变元的顺序变化进行跟踪。编写一个函数完成列轴旋转，其参数中包括用于指明变元顺序变化的参数。假定相应的函数原型如下：

```
void pivot_c(double a[][N+1], int j, int reorder[N]);
```

27. 修改问题 23 中开发的函数，使得列轴旋转在变元消去之前进行。使用问题 26 中所开发的函数。

28. 修改问题 23 中开发的函数，使得行轴旋转和列轴旋转都在变元消去之前进行。使用问题 24 和 26 中所开发的函数。

行列式。下面的问题定义了方阵的代数余子式和子式，使用它们来计算行列式。

29. 矩阵 A 中的元素 $a(i,j)$ 的子式（minor）是除去包含元素 $a(i,j)$ 所在的行和列所形成的矩阵的行列式。因此，如果原矩阵有 4 行 4 列，那么它的子式是一个 3 行 3 列矩阵的行列式。编写一个函数，计算一个 4 行 4 列的方阵的子式。输入的参数为矩阵 A 和值 i 与 j。假定相应的函数原型为：

```
double minor(double a[][4], int i, int j);
```

30. 矩阵 A 的代数余子式（cofactor）$A(i,j)$ 是 $a(i,j)$ 的子式与因子 $(-1)^{i+j}$ 的乘积。编写一个函数计算一个 4 行 4 列的方阵的代数余子式。输入参数为矩阵 A 和值 i 与 j。你可能需要调用问题 29 中的函数。假定相应的函数原型为：

```
double cofactor(double a[][4], int i, int j);
```

方阵 A 的行列式可以通过下面的步骤计算：

(a) 选择任意列。

(b) 将所选的列中的每个元素与它的代数余子式相乘。

(c) 将 (b) 中得到的乘积相加。

31. 编写一个函数 det_c，使用上面的方法计算一个 4 行 4 列的方阵的行列式。你可能需要调用问题 30 中开发的函数。假定相应的函数原型为：

```
double det_c(double a[][4]);
```

32. 方阵 A 的行列式可以通过下面的步骤计算：
    (a) 选择任意行。
    (b) 将所选的行中的每个元素与它的代数余子式相乘。
    (c) 将 (b) 中得到的乘积相加。

33. 编写一个函数 det_r，使用上面的方法计算一个 4 行 4 列的方阵的行列式。你可能需要调用问题 30 中开发的函数。假定相应的函数原型为：

```
double det_r(double a[][4]);
```

34. 编写一个函数找出一个二维 vector 的转置。假定相应的函数原型如下：

```
void transpose(vector<vector<int> > &b, vector<
                                        vector<int> > &c);
```

35. 编写一个函数完成矩阵乘法。假定相应的函数原型如下：

```
void matrixMult(vector<vector<int> > &a,
                vector<vector<int> > &b,
                vector<vector<int> > &c);
```

第 9 章

Engineering Problem Solving with C++, 3e

指　　针

工程挑战：天气模式

正常情况下，沿赤道海洋表面温度是沿太平洋西部之较温暖，而沿太平洋东部则较冷。但是有一种反常的现象，即一条暖流导致太平洋东部（沿加利福尼亚的西海岸）的海洋温度增加了 18 华氏度。这种现象经常在圣诞节临近时发生，因此称作厄尔尼诺（在西班牙语中，厄尔尼诺是一个男孩）。而在太平洋西部则会发生相反的现象，即海水温度变低，这一现象称作拉尼娜（在西班牙语中，拉尼娜是一个女孩）。这个现象与暖流和东西气压变化间的南方涛动有关。ENSO（厄尔尼诺南方涛动）指数是一个从一系列变量中计算得到的度量值，这些变量包括气压、风和海洋温度。当 ENSO 指数是正数时，海洋温度表征为厄尔尼诺现象；当 ENSO 指数为负数时，海洋温度表征为拉尼娜现象。指数值越大，温度偏离正常值的范围就越大。本章我们开发了一个程序，读取一组 ENSO 指数，并确定厄尔尼诺现象出现的时间。

教学目标

本章我们所讨论的问题解决方案中包括：

- 地址与指针
- 指向数组的指针
- 动态内存分配
- 指向字符串的指针
- new 和 delete
- 链式数据结构
- C++ 标准模板库（STL）中的类
- 迭代器

9.1　地址与指针

C++ 程序在执行时，存储单元被分配给程序中使用的对象。每个存储单元都有一个唯一定义该单元的正整数地址。当对象被赋予一个值时，这个值就被存储在对应的存储单元中。对象的值可以被程序中的语句使用，也可以被程序中的语句修改。在程序每次执行时，所使用的对象的地址将被确定，并且每次执行时对象的地址可能都不同。

有时将存储单元比作一组邮政信箱。如果邮局有 100 个编号 1 ~ 100 的邮政信箱，那么邮政信箱的编号就对应着内存地址。每个邮政信箱都被赋予某个个体，拥有个体的名字；这个名字对应着赋给某个内存单元的标识符。信箱的内容对应于内存单元中的值；这个值可以被检查，也可以被改变。

邮政信箱编号	个体的名字	内容
78	John Ruiz	utility bill

内存地址	标识符	内容
0xbffff8d8	x	105

这种类比并非完全准确，因为两个不同的个体可能有相同的名字，但是标识符在严格意义上是不能相同的。此外，一个邮政信箱可能是空的，也可能包含许多信件，而内存单元总是只包含一个值。

9.1.1　地址操作符

在 C++ 中，一个对象的地址可以通过地址操作符（address operator）"&" 来引用，该操作符与按引用传递一起在第 6 章中介绍过。回想一下地址操作符被放在函数原型和函数头中某个形参的数据类型之后，当函数被调用时，形参就会接受对应参数的地址。为了说明使用地址操作符获取某个对象内存地址的用法，考虑下面的程序：

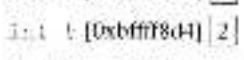
```
/*  ------------------------------------------------------------------------*/
/*  Program chapter9_1                                                      */
/*                                                                          */
/*  This program demonstrates the relationship                              */
/*  between objects and addresses                                           */

#include <iostream> //Required for cout
using namespace std;
int main()
{
//  Declare and initialize objects.
    int a(1), b(2);

//  Print the contents and addresses of a and b.
    cout << "a= " << a << ";  address of a = " << &a << endl;
    cout << "b= " << b << ";  address of b = " << &b << endl;
    return 0;
}
/*  ------------------------------------------------------------------------*/
```

程序的一次示例输出如下所示：

```
a = 1;  address of a = 0xbffff8d8
b = 2;  address of b = 0xbffff8d4
```

下面的内存快照给出了 cout 语句执行时两个内存单元的内容。内存快照同时还给出了每个变量的内存地址。

```
int a [0xbffff8d8] 1
int b [0xbffff8d4] 2
```



在这些示意图中我们通常不指出内存地址，因为这些地址是与系统相关的，并且程序每次执行时都会发生变化。

在下个例子中，对前一个程序进行了修改，取消了对对象 a 和 b 的初始化：

```
/*  ------------------------------------------------------------------------*/
/*  Program chapter9_2                                                      */
/*                                                                          */
```

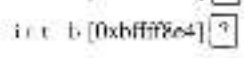
```
/*  This program demonstrates the relationship                    */
/*  between objects and addresses.                                */

#include <iostream>//Required for cout,
using namespace std;
int main()
{
//  Declare and initialize objects.
    int a, b;

//  Print the contents and addresses of a and b.
    cout << "a= " << a << ";  address of a = " << &a << endl;
    cout << "b= " << b << ";  address of b = " << &b << endl;
    return 0;
}
/*-----------------------------------------------------------------*/
```

该程序的一次示例输出如下所示：

```
a = 1073743608;  address of a = 0xbffff8e8
b = 1073784016;  address of b = 0xbffff8e4
```

在 cout 执行时的内存快照中，内存单元内容给出的是“?”，因为变量没有被初始化。但是变量已经被分配了内存，所以每个变量的内存地址都已经被确定。

```
int a [0xbffff8e8] [?]
int b [0xbffff8e4] [?]
```



我们看到，虽然没有在程序中进行赋值，但是变量中还是有值存在，我们不应当对这些值进行任何假设。这个例子说明了在程序其他语句中使用对象前对其进行初始化的重要性。

修改

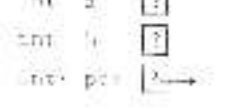
1. 在计算机上运行程序 chapter9_1 两次，你的计算机使用了相同的地址还是不同的地址？将你的结果与你的同学进行比较。
2. 运行本节给出的程序 chapter9_2，分配给 a 和 b 的内存单元的值是多少？程序在不同的执行过程中这些值发生变化了吗？

9.1.2 指针的分派

C++ 语言允许我们将内存单元的地址存放在一种称为指针（pointer）类型的对象中。在定义一个指针时，它将要指向的对象的类型也必须定义。指针所指向的对象的类型称为指针的基类型（base type）。指针的基类型决定了指针指向的对象将被如何解释。因此，一个被定义为指向一个整型对象的指针不能用来指向一个浮点对象。下面的语句声明了两个整型对象和一个指向整型对象的指针。注意其中使用星号来表明一个对象是指针，其中星号称作解引用（dereference）或间接引用（indirection）操作符：

```
int a, b, *ptr;
```

该语句说明内存地址应该被指派给三个对象——两个整型对象和一个指向整型对象的指针。语句中没有指明 a、b 和 ptr 的初始值。因此，声明之后的内存快照中所有对象的值都没有指定；示意图中使用箭头和“?”来表示 ptr 是一个指针，且 ptr 所指向的内容未确定。



```
int a [?]
int b [?]
int ptr [?]→
```

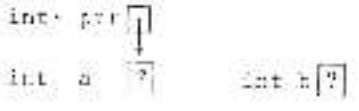
为了说明 ptr 应指向对象 a，我们可以使用赋值语句将 a 的地址存储在 ptr 中：

```
int a, b, *ptr;
ptr = &a;
```

我们也可以在声明语句中对 ptr 进行初始化：

```
int a, b, *ptr = &a;
```

在两种情况下，声明之后的内存快照如下：

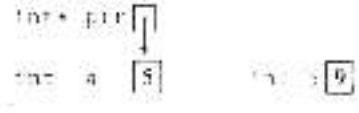


注意只要 ptr 指向的对象确定了，就没有必要指明 ptr 的内容。

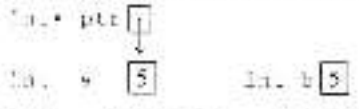
考虑下面的语句集：

```
// Declare and initialize objects.
int a(5), b(9), *ptr(&a);
...
// Assign the value pointed to by ptr to b.
b = *ptr;
```

最后一条语句读作“将地址 ptr 中包含的值赋给 b”，或者“将 ptr 所指向的值赋给 b”。在这条赋值语句执行前的内存快照如下：



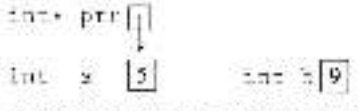
在赋值语句执行后的内存快照如下：



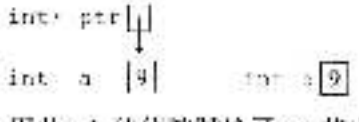
因此，b 被赋予了 ptr 指向的值。现在考虑下面的语句集：

```
// Declare and initialize objects.
int a=5, b=9, *ptr=&a;
...
// Assign the value of b to the object
// pointed to by ptr.
*ptr = b;
```

语句集中的赋值语句读作“b 的值被赋给 ptr 指向的对象”。赋值语句执行前的内存快照如下：



在赋值语句执行后的内存快照如下所示：



因此，b 的值被赋给了 ptr 指向的对象。

指针类型的声明：指针类型使用类型声明语句和指针操作符（*）进行声明。指针操作符可以使用地址操作符（&）进行初始化。

语法：声明指针

```
类型 * 标识符1 [,*标识符2,...];
```

示例：声明两个指向 int 的指针

```
int *iPtr, *qPtr;
```

示例：声明并初始化一个指向 int 的指针

```
int a=5;
int *aPtr = &a;
```

现在我们对程序 chapter9_1 进行扩展，以说明对象、地址和指针之间的关系。考虑下面的程序：

```
/*-------------------------------------------------------------*/
/*  Program chapter9_3                                         */
/*                                                             */
/*  This program demonstrates the relationship                 */
/*  between objects, addresses, and pointers.                  */

#include <iostream> //Required for cout
using namespace std;
int main()
{
// Declare and initialize objects
int a(1), b(2), *ptr(&a);

// Print address and contents of all objects.
    cout << "a = " << a << "; address of a = " << &a << endl;
    cout << "b = " << b << "; address of b = " << &b << endl;
    cout << "ptr = " << ptr << "; address of ptr = " << &ptr << endl;
    cout << "ptr points to the value " << *ptr << endl;
    return 0;
}
/*-------------------------------------------------------------*/
```

该程序的示例输出如下所示：

```
a = 1; address of a = 0xbffff8c8
b = 2; address of b = 0xbffff8c4
ptr = 0xbffff8c8; address of ptr = 0xbffff8c0
ptr points to the value 1
```

注意指向 a 的指针的值和 a 的地址是相同的。

练习

给出下面每组语句执行后的内存快照。

1.
```
int a(1), b(2), *ptr;
ptr = &a;
```

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 = 0;
a = *ptr;
```

9.1.3 指针的算术

可以对指针（或者地址）进行的操作限于以下几种：

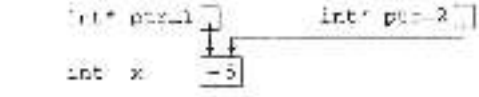
- 可以将一个指针赋给另一个类型相同的指针
- 可以将一个指针加上或减去一个整数值
- 可以将整数 0 赋给指针或将 0 与指针进行比较，等价地，也可以将指针与定义在 <iostream> 中的符号常量 NULL 进行前述操作

此外，对指向数组的指针进行减法或者比较操作也是访问数组元素的一种方式。

在某一时刻，一个指针只能指向一个地址，但是多个指针可以指向同一个地址，下一个例子将进行说明。示例中 ptr_1 和 ptr_2 在下面的语句被执行后指向同一个对象：

```
// Declare and initialize objects.
int x( 5 ), y(6), *ptr_1, *ptr_2;
...
// Assign both pointers to x.
ptr_1 = &x;
ptr_2 = ptr_1;
```

这些语句执行后的内存快照如下所示：



现在我们给出几条非法的语句，通过它们来说明使用指针时的一些常见错误：

```
&y = ptr_1;          // invalid statement: attempts
                     // to change the address of y

ptr_1 = y;           // invalid statement: attempts
                     // to change ptr_1 to a
                     // nonaddress value

*ptr_1 = ptr_2;      // invalid statement: attempts
                     // to assign an address to an
                     // integer object

ptr_1 = *ptr_2;      // invalid statement: attempts
                     // to assign a
                     // nonaddress value to ptr_1
```

作为一个有益的练习，我们推荐将这些非法语句的内存快照都画出来；在每条语句中我们都试图将一个对象的值存储到指针中或者试图将指针的值存储到对象中。为了避免这些错误，尽量使用比较明确的标识符作为指针的名字，以表明这些标识符与指针相关。

在定义简单对象时，我们不应当对对象内存单元之间的关系进行任何假设。例如，如果一条声明语句定义了两个整数 a 和 b，我们不应该假设它们的位置在内存中是相邻的；也不应当对内存单元的初始值进行任何假设。简单对象的内存分配是与系统相关的。但是，对于数组的内存分配则是保证在一组连续的内存单元中。因此，如果数组 x 包含 5 个整数，那么 x[1] 的内存单元则跟在 x[0] 的内存单元之后，x[2] 的内存单元跟在 x[1] 的内存单元之后，以此类推。因此，如果 ptr_x 是一个指向整数的指针，我们可以使用下面的语句将其初始化为指向整数 x[0]

```
ptr_x = &x[0];
```

语句

```
ptr_x = x;
```

与它等价，因为标识符 x 存储着数组 x 中第一个元素的地址。为了移动指针指向 x[1]，我们可以将 ptr_x 加 1，这将使其指向 x[0] 后面的一个元素，或者我们可以将 x[1] 的地址赋给 ptr_x。因此，若 ptr_x 当前指向 x[0]，则下面的任一语句都会使 ptr_x 指向 x[1]：

```
x[1]:

++ptr_x:            // increment ptr_x to point to the
                    // next value in memory

ptr_x++:            // increment ptr_x to point to the
                    // next value in memory

ptr_x = ptr_x + 1:  // increment ptr_x to point to
                    // the next value in memory

ptr_x += 1:         // increment ptr_x to point to
                    // the next value in memory

ptr_x = &x[1]:      // ptr_x is assigned the
                    // address of x[1]
```

类似地，语句

```
ptr_x + k:
```

引用了 ptr_x 指向的值之后的第 k 个值的地址。这些例子都是将整数与指针相加的例子。类似地，整数也可以与指针相减。9.2 节中将这里的讨论扩展到了一维数组中。

当指针与一个整数值相加或相减时，可以认为该整数是指针要移动值的个数。例如，语句

```
ptr++:
```

表示 ptr 应该被修改为指向内存中的下一个值，即当前 ptr 所指向值的下一个值。因为不同类型的值要求不同的内存量，实际加到 ptr 上的数值取决于指针的基类型。一个 double 类型所需的内存比整数要多，因此对于一个 double 类型指针而言，其地址增加量要比一个指向 int 的指针要多。例如，如果在你的系统上 int 的大小为 4 字节，double 为 8 字节，那么对于连续的整数其内存地址可能是 0xbffff8c4 和 0xbffff8c8，而对于连续的 double 类型而言则可能是 0xbffff8e4 和 0xbffff8fc。在这种情况下，对于一个 int 类型的指针的地址增长将会使指针值增加 4，而对于 double 类型的指针则会增加 8。幸运的是，在我们对指针进行加减时，编译器将为我们确定正确的地址增加值。

指针操作可能被包含在含有其他操作的语句中，因此确定正确的操作优先级是很重要的。地址操作符是一元操作符，因此它在二元操作符之前执行；如果语句中有一个以上的一元操作符，那么一元操作符总是从右向左执行的。表 9.1 中对这些优先级规则进行了总结。记住圆括号总是可以用来改变操作的优先级。

表 9.1 操作符优先级

优先级	操作符	结合性	优先级	操作符	结合性
1	() []	箭内优先	7	&&	从左向右
2	-+ -- + - ! (类型) & *	从右向左（一元）	8	|	从左向右
3	* / %	从左向右	9	?:	从右向左
4	- -	从左向右	10	= += -= *= /= %=	从右向左
5	< <= > >=	从左向右	11	,	从左向右
6	== !=	从左向右			

由指针导致的错误较难调试。更糟的是，指针错误常会在运行看似正常的情况下给出错误的结果。许多指针错误都是由于指针在使用前未初始化导致的。因此，在程序开头将所有的指针初始化是一个好的习惯。如果指针不能用内存单元地址初始化，则给它赋一个 NULL 的值。一个被赋予 NULL 的值不指向任何内存单元。你可以使用包含形如（NULL == ptr_1）的条件语句，在程序中的某处确定名为 ptr_1 的指针是否被赋予了一个内存单元的地址。

练习

对于下面的每个问题，给出程序语句执行后包括所有对象在内的内存快照。包含尽可能多的信息。使用问号来表示未被初始化的内存单元。

1.
```
double x(15.6), y(10.2), *ptr_1(&y), *ptr_2(&x);
*ptr_1 = *ptr_2 + x;
```

2.
```
int w(-3), x(2), *ptr_2(&x);
*ptr_2 = w;
```

3.
```
int x[5]={12,4,6,8,3};
int *ptr_1 NULL, *ptr_2 NULL, *ptr_3 NULL;
ptr_3 = &x[3];
ptr_1 = ptr_2 = ptr_3 + 2;
```

4.
```
int w[4], *first_ptr(NULL), *last_ptr(NULL);
first_ptr = w;
last_ptr =first_ptr + 5;
```

9.2　指向数组元素的指针

在第 7 章和第 8 章中，我们使用偏移量来确定特定的元素，对数组和数组操作进行了详细的讨论。指针也可以用于确定特定的数组元素。使用指针和地址对数组进行引用几乎总是比使用偏移量要快，因此，如果比较关注运行速度，一般考虑使用指针来引用数组。如同在本节讨论的，使用指针引用数组的值是基于数组值的内存分配总是连续的这一知识。

9.2.1　一维数组

考虑下面的声明，声明中使用一组浮点值定义并初始化了一个一维数组：

```
double x[6]={1.5, 2.2, 4.3, 7.5, 9.1, 10.5};
```

执行该语句后的内存快照如下：

double x[0]	1.5
double x[1]	2.2
double x[2]	4.3
double x[3]	7.5
double x[4]	9.1
double x[5]	10.5

使用标准的数组记号来引用数组 x 时，x[0] 引用了数组的第一个元素，x[1] 引用数组的第二个元素，x[k] 引用数组的第（k+1）个元素。类似的引用也可以使用指针完成。假设指针 ptr 已经被定义为指向一个 double 类型的指针，并使用下面的语句初始化

```
ptr = &x[0];
```

这时指针 ptr 存储了数组中第一个元素的地址。因此，引用 *ptr 就表示 x[0]，引用 *(ptr+1) 表示 x[1]，引用 *(ptr+k) 表示 x[k]。引用 *(ptr+k) 中 k 的值是距离数组中第一个

元素的偏移量。

下面的语句使用数组记号和 for 循环计算出数组 x 中数值的和：

```
// Declare and initialize objects.
double x[6], sum[0];
...
// Sum the values in the array x.
for (int k=0; k<5; ++k)
{
   sum += x[k];
}
```

下面是使用指针代替数组记号的等价语句：

```
// Declare and initialize objects.
double x[5], sum[0], *ptr=&x[0];
...
// Sum the values in the array x.
for (int k=0; k<5; ++k)
{
   sum += *(ptr+k);
}
```

注意在引用 *(ptr+k) 中需要使用圆括号来保证正确的计算顺序：首先将 k 加到 ptr 的地址值上，然后使用间接引用操作符来引用 ptr+k 指向的值。引用 *ptr+k 将作为 (*ptr)+k 进行计算，因为一元操作符的优先级高于二元操作符。

在第 7 章有关数组的讨论中，我们看到数组名就是第一个元素的地址。因此，数组名也可以作为一个指针来引用数组中的元素。例如，下面的两条语句时等价的：

```
ptr = &x[0];
ptr = x;
```

类似地，引用 *(ptr+k) 与 *(x+k) 也是等价的。因此，计算数组 x 的和的语句可以简化成下面的形式：

```
// Declare and initialize objects.
double x[5], sum[0];
...
// Sum the values in the array x.
for (int k=0; k<5; ++k)
{
  sum += *(x+k);
}
```

本例说明了使用数组名作为地址的用法。数组名在大多数语句中都可以用来替换指针，但是不能用在赋值语句的左侧，因为它的值不能被改变。

练习

假定数组 g 由下面的语句定义：

```
int g[] = {2, 4, 5, 8, 10, 32, 78};
int *ptr1(&g[1]), *ptr2(&g[5]);
```

给出包含数组值在内的内存快照，同时指出距离数组第一个值的偏移量。使用该信息给出下面引用的值。

1. *g
2. *(g+1)
3. *g.1
4. *(g+5)
5. *ptr1
6. *ptr2
7. *(ptr1 + 3)
8. *(ptr2 + 2)

9.2.2　字符串

回忆第 7 章中的内容，字符数组就是一个各元素存储为字符的数组。一个 C 风格的字符串就是一个以 null 字符为最后一个元素的字符数组。字符串被用在包括密码学和模式识别等在内的许多工程应用中。通过指向字符串的指针对字符串进行操作通常比较方便。第 6 章介绍了许多包含在头文件 cstring 中的字符函数，这些函数都要求使用指向字符串的指针作为参数，并且其中的许多函数返回指向字符串的指针。本节我们将看看使用指针引用 C 风格字符串的语法。

函数 strstr() 是包含在头文件 cstring 中的函数之一。函数 strstr(ps, pt) 接受一个指向字符串 s 的指针和一个指向字符串 t 的指针作为参数，返回一个指向包含字符串 s 的字符串 t 的指针。如果 t 中没有出现 s，则返回一个 NULL 指针。我们将在下一个例子中使用该函数。

假定我们想统计一个字符串在另一个字符串中出现的次数。例如，我们想统计字符串“bb”在字符串“abbcfgwdbibbw”中出现的次数。函数 strstr() 将返回一个指向字符串“bb”在字符串“abbcfgwdbibbw”中第一次出现的位置的指针，如果没有找到“bb”则返回 NULL 指针。在本例中，strstr() 函数将返回一个指向“bb”第一次出现的位置的指针，如下所示：

```
"abbcfgwdbibbw"
  ^
```

为了找到字符串“bb”下一次出现的位置，我们需要在“bb”第一次出现的位置之后的部分进行搜索。函数 strstr() 将在新的字符串中找到“bb”第一次出现的位置，并返回指向该位置的指针，如下所示：

```
"bcfgwdbibbw"
        ^
```

重复这一过程直到函数 strstr() 返回一个 NULL 为止，完成该过程的完整程序如下所示：

```
/*-----------------------------------------------------------------*/
/* Program chapter9_4                                              */
/*                                                                 */
/*  This program counts and prints the number of                   */
/*  times one string appears within another string.                */
/*
#include <iostream> //Required for cout
#include <cstring> //Required for strstr
using namespace std;

int main()
{
//  Declare and initialize objects.
int count(0);
char strg1[] ="abbcfgwdbibbw" : strg2[] = "bb";
char *ptr1(strg1), *ptr2(strg2);
//  Count  number of occurrences of strg2 in strg1.
//  While function strstr does not return NULL
//  Increment count and move ptr1 to next section
//  of strg1.
while ((ptr1 strstr(ptr1,ptr2)) != NULL)
{
  count(
  ptr1++;
```

```
// Print the number of occurrences.
cout << "Count: " << count << endl;

return 0;
}
/*                                                                    */
```

该程序的输出为：

```
count: 2
```

9.2.3 指针作为函数参数

当指针作为参数传递给函数时，指针可以通过值传递或引用传递。如果指针通过值传递，那么指针可以用于修改其所指向的对象，但是指针参数的值不能被函数修改。如果指针通过引用传递，那么指针所指向的对象可以被修改，指针参数的值也可以被修改。

考虑下面的程序，其中调用了一个函数将字符串转换成大写：

```
/*                                                                    */
/* Program chapter9_1                                                 */
/*                                                                    */
/* This program converts a string to all upper case.                  */
/*                                                                    */
#include <iostream> //required for cout
#include <cctype> //Required for toupper()
using namespace std;

//Function prototypes
void stringupper(char*);

int main()
{
// Declare and initialize objects.
char strg1[] ="abcdefghijklm";
char *ptr_strg=strg1;

// Output string before and after call to function.
cout << ptr_strg1 << endl;
stringupper(ptr_strg1);
cout << ptr_strg1 << endl;

return 0;
}
/*                                                                    */
/*                                                                    */
/*  This function converts each character in                          */
/*  the string pointed to by ptr_strg to upper case.                  */
/*                                                                    */
void stringupper(char* ptr_strg)
{
  // While not end of string (while character is not null).
  while(*ptr_strg)
  {
     // Convert character to upper case
     *ptr_strg =  toupper(*ptr_strg);

     // Move pointer to next character
```

```
    ptr_strg++;
  }
}
/*-------------------------------------------------------------------*/
```

在这个例子中，指针 ptr_str1 被传递给函数 stringupper()，ptr_str 所指向的字符串被转换为大写。在函数中增加形参 ptr_strg 以转换字符串，但并没有对主函数中的参数进行修改，因为这里使用的是值传递。程序的输出如下：

```
abbcfgwdbibbw
ABBCFGWDBIBBW
```

下面的程序对程序 chapter 9_5 中的函数进行了修改，将值传递的方式换成了引用传递：

```
/*-------------------------------------------------------------------*/
/* Program chapter9_6                                                */
/*                                                                   */
/*  This program converts a string to all upper case.                */
/*                                                                   */
#include <ostream> //Required for cout
#include <ctype> //Required for toupper
using namespace std;

//Function prototypes
void stringupper(char*&);

int main()

//  Declare and initialize objects.
char strg1[] ="abbcfgwdbibbw";
char *ptr_strg1 = strg1;

// Input string before and after call to function.
cout << ptr_strg1 << endl;
stringupper(ptr_strg1);
cout << ptr_strg1 << endl;
return 0;
}
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/*                                                                   */
/*  This function converts each character in the string              */
/*  pointed to by ptr_strg to upper case.                            */
/*                                                                   */
void stringupper(char* &ptr_strg)
{
  // While not end of string,
  while(*ptr_strg)
  {
     // Convert character to upper case
     *ptr_strg = toupper(*ptr_strg);

     // Move pointer to next character
     ptr_strg++;
  }
}
/*-------------------------------------------------------------------*/
```

注意函数原型和函数头中的操作符顺序。当通过引用方式传递指针时，地址操作符必须跟在解引用操作符之后。

在这个版本的程序中，和程序 chapter 9_5 一样也是将字符串转换成大写。但在这个版本中，函数对形参的修改使主函数中的参数也被修改了。因为这里采用了引用传递的方式。在调用了函数 stringupper() 之后，指针 ptr_strg1 指向了字符串末尾的 null 字符。因此，大写字符串不会被打印。程序的一次示例输出如下：

```
abbcfgwcbitbw
```

在两个例子中，函数 stringupper() 都可以修改形参所指向的字符串对象。

有时需要将指针传递给函数作为参数，同时还要避免函数不经意地修改指针所指向的对象。这时我们可以在函数原型和函数头中使用 const 限定符，以避免指针所指向的对象被修改。在下一个例子中我们将进行说明。函数 stringupper() 的定义没有包含在这个例子中：

```
/*-------------------------------------------------------------------------*/
/* Program chapter9_7                                                      */
/*                                                                         */
/*  This program counts the number of times a specified                    */
/*  letter appears in an upper case string.                                */
/*                                                                         */
#include <iostream> //Required for cout
#include <cctype> //Required for toupper()
using namespace std;

//Function prototypes
void stringupper(char*);
int countchar(const char*, char);

int main()
{
// Declare and initialize objects.
char strg[] ="abbcfgwcbitbw";
char *ptr_strg1 = strg, ch='b';

// Convert string to upper case.
stringupper(ptr_strg1);

cout << "The letter " << ch << " appears "
     << countchar(ptr_strg1, ch) << " times in the string "
     << ptr_strg1 << endl;

return 0;
}
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*  This function counts the number of times the character ch      */
/*  appears in the string pointed to by ptr_strg.                          */
/*                                                                         */
int countchar(const char* ptr_strg, char ch)
{
   // Declare and initialize local objects.
   int cnt=0;

   // While not end of string.
   while(*ptr_strg)
   {
   // Look for ch and increment cnt.
      if( *ptr_strg == ch)
         cnt++;
```

```
    // Never pointer to next character
    ptr_strg++;
    }
    return cnt;
}
/*      ------------------------------------------------------------*/
```

程序生成的输出如下所示：

```
The letter B appears 5 times in the string ABBCFGWDBIBBW
```

函数 countchar() 不能修改 ptr_strg 指向的对象。例如，如果我们在 if 语句中将相等操作符替换成赋值操作符，即

```
if ( *ptr_strg = ch):
```

编译器会将其标识为错误。

const 限定符也可以用在定义指针的类型声明语句中，如下面的语句：

```
const char *cptr = "abbcfgwdbibbw";
```

在前面的语句中，cptr 被定义成指向常量的指针（pointer to a constant）。需要注意的是，cptr 本身不是常量，但是 cptr 指向的内容被看做是常量。因此，cptr 可以被重新赋值，但是我们不能使用 cptr 来修改对象。下面的代码段说明了 const 限定符的用法：

```
// Declare and initialize objects.
int x(5), y(10), *ptr1(&x);
const int *cptr(&y);

// Print values.
cout << *ptr1 <<',' << *cptr << endl;
cout << x << ',' << y << endl;

// Increment x.
(*ptr1)++;

// Print values.
cout << *ptr1 <<',' << *cptr << endl;
cout << x << ',' << y << endl;

// Reassign both pointers.
ptr1 = &y;
cptr = &x;

// Increment y.
(*ptr1)++;

// Print values.
cout << *ptr1 <<',' << *cptr << endl;
cout << x << ',' << y << endl;
```

该代码段生成的输出如下所示：

```
5,10
5,10
6,10
6,10
11,5
6,11
```

常量指针（constant pointer）可以在类型声明语句中声明，如下面的语句：

```
char *const cptr = "A standard message.";
```

使用这种语法，cptr 将不能被重新赋值，但是当 cptr 所指向的对象可以被修改。

练习

假定有一些对象由下面的语句定义：

```
int i(5), *ii(0);
int *ipi(&i);
const int *cpi(&i);
int *const ipc(&i);
```

确定下面的语句是否合法：

```
1. cpi = ipc;          2. ipc = cpi;          3. *cpi = *ipi;
4. *cpi = *ipi;        5. *ipi = *cpi;        6. ipi = cpi;
```

9.3 解决应用问题：厄尔尼诺南方涛动数据

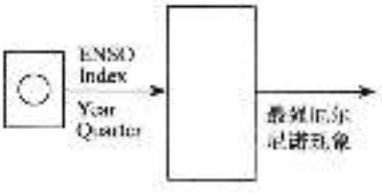
在本章开头的讨论中，我们讨论了海洋表面温度。正常情况下，沿赤道海洋表面温度是沿太平洋西部比较温暖，而沿太平洋东部则较冷。但是有一种反常的现象，即一条暖流导致太平洋东部（沿加利福尼亚的西海岸）的海洋温度增加了 18 华氏度。这种现象经常在圣诞节邻近时发生。因此被称作厄尔尼诺（在西班牙语中，厄尔尼诺是一个男孩）。而在太平洋西部则会发生相反的现象，即海水温度变低。这一现象被称作拉尼娜（在西班牙语中，拉尼娜是一个女孩）。ENSO（厄尔尼诺南方涛动）指数是一个从一系列变量中计算得到的度量值，这些变量包括气压、风和海洋温度。当 ENSO 指数是正数时，海洋温度表征为厄尔尼诺现象；当 ENSO 指数为负数时，海洋温度表征为拉尼娜现象。指数值越大，温度偏离正常值的范围就越大。编写一个程序，从包含有一定时间段内年、季度和 ENSO 指数的数据文件中读取数据。该程序要确定并打印出最强厄尔尼诺现象的年和季度。

1. 问题描述

确定最强厄尔尼诺现象的年和季度。

2. 输入/输出描述

下面的 I/O 示意图给出了程序的输入和输出，输入是数据文件，输出为年和季度。



3. 用例

假定数据文件中包含下面的数据：

Year	Quarter	ENSO Index	Year	Quarter	ENSO Index
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			

对应的输出应当如下：

```
Maximum El Nino Conditions in Data File
Year: 1998, Quarter: 1
```

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。

分解提纲

1）将ENOS数据读入数组，确定最大的正指数；

2）打印出对应于最大指数的年和季度。

细化的伪代码

```
main: if file cannot be opened
          print error message

    else
          read data and determine maximum intensity
          print the year and quarter that go with maximum intensity
```

伪代码中的步骤已经足够详细，可以转换成C++代码

```
/*-------------------------------------------------------------------*/
/* Program chapter 9_2                                               */
/*                                                                   */
/* This program reads a data file of ENSO index values and           */
/* determines the maximum El Nino condition in the file.             */

#include<iostream> //Required for cout
#include<fstream> //Required for ifstream

const int MAX_SIZE = 1000;

using namespace std;

int main()
{
  //Declare variables
  int k=0, year[MAX_SIZE],qtr[MAX_SIZE], maxK=0;
  double index[MAX_SIZE];
  ifstream fin("ENSO1.txt");
  if(fin.fail())
  {
    cerr <<"Could not open file ENSO1.txt" <<endl;
    exit(1);
  }
  fin >> *year, *qtr, *index;
  while(fin)
  {
    if(*(index+k) > *(index+maxK))
    {
      maxK = k;
    }
    k++;

     fin >> *(year+k) >> *(qtr+k) >> *(index+k);
 }//end while

  /* Print data for maximum El Nino condition.                        */
  cout << "Maximum El Nino conditions in Data File \n";
```

```
    cout << "Year:" << *(yearMaxK) <<" Quarter:"
         << *(qtrMaxK) << endl;
    return 0;
}

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

5. 测试

使用用例中的数据得到的程序输出如下所示：

```
Maximum El Nino conditions in Data File
Year: 1990 Quarter: 1
```

修改

1. 修改程序，找到并打印出最大的拉尼娜现象。
2. 修改程序，找到最接近于0的现象。这些将是最接近于正常情况的现象。
3. 修改程序，打印出所有厄尔尼诺现象的年和季度。
4. 修改程序，统计厄尔尼诺现象的年数和拉尼娜现象的年数，打印出这两个值。
5. 修改程序，计算并打印出厄尔尼诺现象年份的平均强度。

9.4 动态内存分配

动态内存分配允许程序在执行时为对象分配内存，而不是在程序编译时确定内存需求。这在程序执行期间使用一个大小不能确定的数组时显得很重要；没有动态内存分配，程序将不得不指定可能的最大数组大小。对于内存有限的系统而言，如果程序中所有用到的数组都指定最大的大小，则很有可能没有足够的内存来运行程序。动态内存是从一个称作程序的堆（heap）的区域中分配可用的内存。动态内存通过new操作符从堆上进行分配，通过delete操作符返回给堆。

9.4.1 new操作符

动态内存分配使用关键字new和其后所跟的对象类型标识符来完成。new操作符返回新的对象在内存中的地址。为了给一个int类型的对象分配内存，我们可以使用下面两组语句集中的任一种：

```
// Example 1
// Dynamically allocate memory for one integer object.
// No initial value is given to the object.
int *ptr;
ptr  = new int;

// Example 2

// Dynamically allocate memory for one integer object.
// Give an initial value of -1 to the object.
int *ptr;
ptr  = new int(-1);
```

在前面的两个例子中，new操作符都返回一个赋给指针的值。如果堆上有可用内存，指针将包含所分配内存的地址。在第一个例子中，没有为新分配的内存指定初始值，如图9.1所示。在第二个例子中，为新分配的内存指定了初始值-1，如图9.2所示。

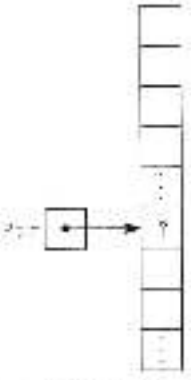


图 9.1 示例 1 中堆的示意图

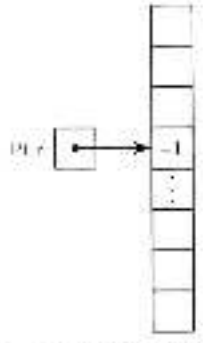


图 9.2 示例 2 中堆的示意图

因为堆是有限的，所以分配内存的请求不能得到满足的可能总是存在的。如果堆上没有足够的内存来满足内存请求，一般来说，new 操作符将抛出一个异常（throw an exception），名为 bad_alloc，该异常将终止程序的执行。C++ 中支持一种检测程序异常并从潜在的错误恢复的机制，这种机制称作异常处理（exception handling）。本书中不讨论异常处理。

9.4.2 动态分配数组

动态内存分配在需要大量数组的工程问题中十分有用。加入我们想要动态分配一块内存，用于存储一个包含有 npts 个元素的 double 数组。（在本例中，我们为 npts 指定一个较小的值用来说明，但是在更一般的情况下 npts 都是一个很大的值，该值由程序中的其他语句计算得到或者从键盘或数据文件中读取。）下面的语句集说明了分配的要求：

```
// Declare objects.
int npts = 10;
double *dptr;
...
// Dynamically allocate memory.
dptr = new double[npts];
```

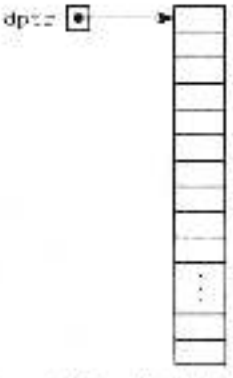


图 9.3 在堆上分配的内存块

如图 9.3 所示，在堆上将分配一块连续的内存用以存储 10 个 double 类型的对象。指针 dptr 被赋予数组中第一个元素的内存地址。可以使用指针记号来引用数组，如 *(dptr+2)；也可以使用标准的数组记号来引用数组，如 dptr[2]。

当使用 new 来为对象动态分配内存时，所指定的对象的数据类型必须是内建类型或任何自定义类类型。

9.4.3 delete 操作符

当程序结束了对使用 new 动态分配得到内存的使用后，可以使用 delete 将内存返回到堆中，或者叫做取消分配（de-allocated）。下面的程序说明了操作符 new 和 delete 的用法：

```
/*                                                              */
/*  Program Chapter9_6                                          */
/*  This program illustrates the use of operators new and delete. */

#include<iostream>  //Required for cout
```

```
using namespace std;

int main()
{
// Declare objects.
int *ptr, npts[10];
double *darr_ptr;

// Dynamically allocate one integer object.
ptr = new int( 1);

// Dynamically allocate array of type double.
darr_ptr = new double npts ;

// Assign what is pointed to by ptr to all elements of dynamic array.
for(int i=0; i<npts; ++i)
{
     darr_ptr[i] = *ptr;
}

// Print all values in dynamic array.
for(int i=0; i<npts ; ++i)

{
     cout << darr_ptr[i] << " ";
}
cout << endl;
// Return memory to free store.
delete ptr;
delete [] darr_ptr;
return 0;

/*                                                                    */
```

程序的输出如下所示：

```
1 1 1 1 1 1 1 1 1 1
```

当取消对一个数组的分配时，必须在 delete 后加上空的中括号。如果我们忽略了中括号，程序可能不会被正确地执行。

9.5 解决应用问题：地震监测

地震检波器这种专用传感器用于收集地壳运动信息。这些地震检波器可以用在被动的环境中，在这种环境下它们可以记录包括地震和潮汐运动在内的地壳运动。通过对几个地震检波器收集的地震产生的地表运动数据进行分析，可以帮助确定地震的震中和强度。地震强度通常使用里氏级数来计量，该级数为 1 ~ 10 级，以美国地震学家 C.F.Richter 的名字命名。

编写一个程序从一个名为 seismic.dat 的数据文件中读取一组地震检波器数据。文件的第一行包含两个值：文件中包含的地震检波器数据的数目以及两次连续测量值之间的时间间隔（以秒为单位）。数组的内存单元是基于数据文件中地震检波器数据记录的数量动态分配的。时间间隔是一个浮点值。我们假定所有的测量值之间的时间间隔都相同。在读取并存储测量数据后，程序应当使用能量比标记出可能的地震，也称作地震事件。在给定的时间点上，这个比例是一个短时间内的能量测量值与一个长时间内能量测量值的商。如果这个比例高于给定的阈值，那么在这个时间点上可能发生地震。给定测量数据中的某个点，短时间内的能量测量值是使用给定点加上该点之前的一小部分点的平均能量或值的平方的均值。长时间的能

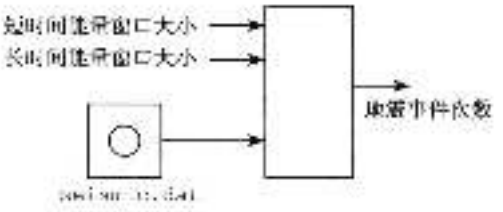
量测量值是使用给定点加上该点之前的一大部分点的平均能量或值的平方的均值。(用于计算的点的集合有时也称作数据窗口。)为了避免检测到常量数据中的事件(因为如果数据值都是相同的值,那么短时间内的能量测量值会与长时间的能量测量值相等),所以给定的阈值一般都大于1。假定用于计算短时间能量测量值和长时间能量测量值的测量值的数目由键盘读入。将阈值设定为1.5。

1. 问题描述

使用数据文件中的一组地震检波器测量值确定可能的地震事件的位置。

2. 输入/输出描述

程序的输入是名为 seismic.dat 的数据文件和用于计算短时间能量和长时间能量的测量值的数目。输出是给出关于潜在的地震事件次数的报告。



3. 用例

假定数据文件中包含下面的数据,其中包含了测量点的数量(11)和测量点之间的时间间隔(0.01),11个值对应的值的序列为 x_0, x_1, …, x_{10}:

```
11 0.01
1   2   1   1   1   2   4   2   1   1   1
```

如果短时间的能量测量值使用两个样本,长时间的能量测量值使用5个测量值,那么我们可以计算出能量比,从窗口中最右边的点开始:

```
1 2 1 1 1 5 4 2 1 1 1

short window

long window

Point x4:    Short-time power = (1 + 1)/2 = 1

Long-time power= (1 + 2 + 1 + 1 + 4)/5 = 1.8
Ratio = 1/1.8 = 0.63

1 2 1 1 1 5 4 2 1 1 1

short window

long window
Point x5:    Short-time power = (25 + 1)/2 = 13

Long-time power = (25 + 1 + 1 + 1 + 4)/5 = 6.4

Ratio = 13/6.4 = 2.03

1 2 1 1 1 5 4 2 1 1 1
```

```
short window

long window

Point x6:    Short-time power = (16 + 25)/2 = 20.5

Long-time power = (16 + 25 + 1 + 1 + 1)/5 = 8.8

Ratio = 20.5/8.8 = 2.33

1 2 1 1 1 5 4 2 1 1 1

short window

long window

Point x7:    Short time power = (4 + 16)/2 = 10

Long-time power = (4 + 16 + 25 + 1 + 1)/5 = 9.4

Ratio = 10/9.4 = 1.06

1 2 1 1 1 5 4 2 1 1 1
short window

long window

Point x8:    Short-time power = (1 + 4)/2 = 2.5

Long time power = (1 + 4 + 16 + 25 + 1)/5 = 9.4

Ratio = 2.5/9.4 = 0.27

1 2 1 1 1 5 4 2 1 1 1

short window

long window

Point x9:    Short-time power = (1 + 1)/2 = 1

Long time power = (1 + 1 + 4 + 16 + 25 )/5 = 9.4

Ratio = 1/9.4 = 0.11

1 2 1 1 1 5 4 2 1 1 1

short window

long window

Point x10:    Short-time power = (1 + 1)/2 = 1

Long-time power = (1 + 1 + 1 + 4 + 16 )/5 = 4.6

Ratio = 1/4.6 = 0.22
```

通过使用前面计算出的比例，可以确定可能的地震事件发生在点 x_5 和 x_{6c} 因为每个点之间的时间间隔为 0.01 秒，所以对应的地震事件时间为 0.05 和 0.06 秒。(我们假定文

件中第一个点发生在0.0秒。）

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。

分解提纲

1）读取文件头并分配内存；

2）从数据文件读取地震数据，从键盘读取计算能量的测量值数目；

3）计算出能量比，打印出可能的地震事件时间。

步骤3中包括计算能量比并将其与阈值比较用以确定是否可能发生地震。因为对于每个可能的事件位置，需要计算两个能量测量值，我们将能量计算作为一个函数。下面给出主函数和能量函数细化后的伪代码：

细化的伪代码

```
main:  set threshold to 1.5
       read npts and time interval
       allocate memory for sensor array

       read the values into sensor array
       read short-window, long-window from keyboard
       set k to long window - 1
       while k<= npts-1
           set short-power to power(sensor,short-window,k)
           set long-power to power(sensor,long-window,k)
           set ratio to short power/long power
           if ratio > threshold
                print k*time-interval
           increment k by 1
power(x,length,n):
     set xsquare to zero
     set k to 0
     while k<=n-1
         add x[length-k]*x[length-k] to xsquare
     return xsquare/length
```

现在我们准备将伪代码转换成C++：

```
/*-------------------------------------------------------*/
/*  Program chapter9_10                                  */
/*                                                       */
/*  This program reads a seismic data file and then      */
/*  determines the times of possible seismic events.     */
/*  Dynamic memory allocation is used.                   */

#include <fstream>//Required for ifstream
#include <string>//Required for string
#include <cmath>//Required for pow()
using namespace std;

//  Set threshold.
const double THRESHOLD = 1.5;

//  Function prototypes.
double power_w(double arr[], int length, int n);
int main()
{
   //  Declare objects.
```

```
    int k, npts, short_window, long_window;
    double time_incr, *sensor, short_power, long_power,
           ratio;
    string filename;
    ifstream fin;

    //  Prompt user for file name and open file for input
    cout << "Enter name of input file\n";
    cin >> filename;
    fin.open(filename.c_str());
    if(fin.fail())
    {
       cerr << "error opening input file" << endl;
    }
    else
    {
    //  Read data header and allocate memory.
       fin >> npts >> time_incr;
       sensor = new double[npts];

    //  Program continues if no exception is thrown.
       cout << "Memory allocated." << endl;

    //  Read data into an array.
       for (k=0; k<npts; ++k)
           fin >> sensor[k];
    //  Read window sizes from the keyboard.
       cout << "Enter number of points for short-window: \n";
       cin >> short_window;
       cout << "Enter number of points for long-window: \n";
       cin >> long_window;

    //  Compute power ratios and search for events.
       for (k=long_window ; k<npts; ++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)
                cout << "Possible event at " << time_incr*k
                                             << " seconds \n";
       }
    //  Return memory to free store, close file, and exit program.
       delete [] sensor;
       fin.close();
    }
return 0;
}
//-------------------------------------------------------*/
//-------------------------------------------------------*/
/*  This function computes the average power in a      */
/*  specified window of a double array.                */
double power_w(double arr[], int length,  int n)
{
    //  Declare and initialize objects.
    double xsquare(0);

    //  Compute sum of values squared in the array x.
    for (int k=0; k<n; ++k)
```

```
    {
       xsquare += pow(arr[length-1],2);
    }

    /*  Return the average squared value.  */
    return xsquare/n;
}
/*-------------------------------------------------------------*/
```

5. 测试

使用用例中的数据得到的程序输出如下所示：

```
Memory allocated.
Enter number of points for short window:
2
Enter number of points for long window:
5
Possible event at 0.05 seconds.
Possible event at 0.06 seconds.
```

修改

修改事件检测程序，使其包含下面的新功能：

1. 允许用户输入阈值。检查该值，确定其是一个大于 1 的正数。
2. 打印出程序检测到的事件数目。（假定在一个连续时间段内的事件都是同一事件的一部分。所以在用例中只检测到一个事件。）
3. 使用 vector 类替换数组，去除对 new 和 delete 的需求。

9.6 使用 new 和 delete 的常见错误

一只已被动态分配的内存不再需要，就应当将其返回到堆中，让它为其他的动态分配请求服务。对于指向内存空间的所有指针都需要小心地追踪。在对一个指针使用 delete 操作符后，其指向的内存返回到堆中，此时指针将指向非法的内存空间或者仍指向已被删除的内存空间的地址，这取决于编译器的实现。在这些情况下，指针都不应当被引用，直到它被赋予一个新的，合法的地址为止。

在处理指针和动态内存分配时非常容易出错，并且很难找到这些错误。下面是一些较为常见的错误：

- 在使用 delete 操作符将指针所指向的动态分配内存返回到堆中之后引用该指针。
- 在存储空间不再被使用后未能将其返回到堆中，这通常称作内存泄漏（memory leak）。
- 对一个没有通过 new 操作符进行动态内存分配的指针使用 delete 操作符。
- 使用 delete 操作符释放动态分配数组时忽略了其后的中括号。

为了避免有关使用动态分配数组的诸多错误，我们推荐使用 vector 类。

练习

假定指针 iptr，jptr 和 arr_ptr 由下面的语句定义：

```
int *iptr, *jptr, *arr_ptr;
iptr = new int(10);
arr_ptr = new int[5];
```

画出内存分配示意图，并给出下面每组语句的输出。我们推荐你为每个例子编写程序，以确定在你的系统上生成的输出。

```
1. jptr = ipt;
   cout << *ptr << ' ' << *jptr << endl;
   cout << ipt << ' ' << jptr << endl;
   delete ipt;
   cout << ipt << ' ' << jptr << endl;

2. cout << arr.ptr << endl;
   for(int i=0; i<4; ++i)
     arr.ptr[i] = i;
   for(int i=0; i<4; ++i)
     cout << *(arr.ptr)++ << ' ';
   cout << endl << arr.ptr << endl;
   for(int i=0; i<4; ++i)
     cout << arr.ptr[i] << ' ';

3. for(int i=0; i<4; ++i)
     arr.ptr[i] = i;
   iptr = &arr.ptr[2];
   cout << arr.ptr << ' ' << iptr << endl;
   cout << *arr.ptr << ' ' << *iptr << endl;
   delete [] arr.ptr;
   cout << arr.ptr << ' ' << iptr << ' ' << *iptr << endl;
```

9.7 链式数据结构

当我们使用诸如数组和 vector 之类的数据结构时，我们都是在处理一块连续的内存块。访问数组或 vector 中的单个元素很方便，因为每次访问时距离起始地址的偏移量都是确定的。但是，要插入或删除一个元素并不容易，除非那个元素是最后一个元素。为了在任意位置插入或删除一个元素，需要将所插入或删除的元素后面的所有元素复制到一个新的内存位置。这样的效率很低，尤其是处理的数据量很大时。

链式数据结构（linked data structure）如链表（link list）、栈（stack）和队列（queue），被设计用于高效的插入和删除。在执行程序时按照需要分配和删除存储空间，指针用于链接元素，因为元素不在一块连续的内存块中。使用指针链接元素可以支持元素的高效插入和删除；链接需要重排，但是不需要对元素进行复制。但是访问链式结构中的元素的效率要低于访问数组中的元素，因为对于每次访问我们都需要遍历（traverse）链个结构。为了遍历一个链式数据结构，我们必须从结构的第一个元素开始，并沿着链接逐次访问元素，直到到达要访问的元素为止。

9.7.1 链表

链表是一种由指针将一组元素链接起来的数据结构。其中的元素由数据和指向下一个元素的指针组成。我们通常假定所存储的元素具有一定的顺序，比如升序，这样我们可能希望从有序列表中删除元素或插入新元素。

图 9.4 给出了带有 4 个元素的链表，其中包含了有序的信息 10、14、21、35。有一个单独的指针被标记为 head，指向了链表的第一个元素。

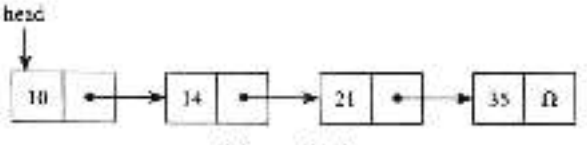


图 9.4 链表

为了访问链表中的元素，我们使用 head 指针来引用第一个元素中的信息（在本例中元素包含的信息为 10）。因为第一个元素包含了指向下一个元素（该元素包含了值 14）的指针，我们可以使用该信息移动到第二个元素。类似地，我们使用第二个元素中的指针移动到第三个元素。链表中最后一个元素将包含一个值为 NULL 的指针，标记我们已经到达了表中最后一个元素。在我们的图中使用符号 Ω 来表示值 NULL。

为了在链表中插入一个值，我们必须遍历链表以找到要插入的位置。从链表的头开始，使用当前元素所包含的指针，逐个移到下一个元素，直到找到我们需要插入的位置为止。插入的位置可以是下面的四种位置之一：第一个元素之前、两个元素之间、最后一个元素之后，或者在空表中插入。图 9.5 ~ 图 9.8 说明了每一种情况，其中假定从图 9.4 中的链表结构开始完成前三种情况。

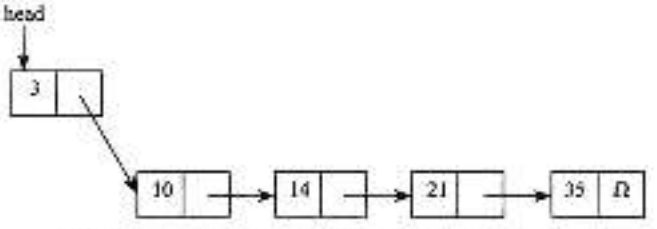


图 9.5　在第一个元素前插入（需要更新 head 指针）

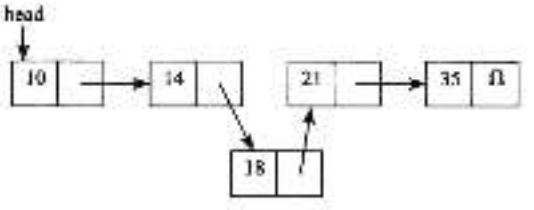


图 9.6　在两个元素间插入

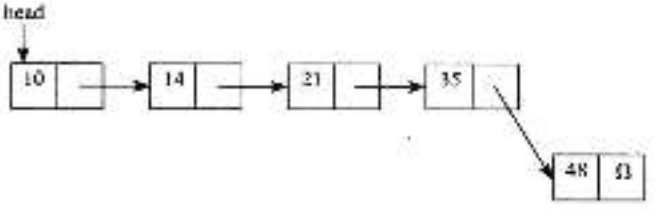


图 9.7　在最后一个元素后插入

为了从链表中删除一个元素，我们必须遍历链表以找到要删除的元素。从链表的头开始，使用当前元素所包含的指针，逐个移到下一个元素。如果找到了要删除的元素，它可能是第一个元素，两个元素之间的元素，或者最后一个元素。图 9.9 ~ 图 9.11 列出了这三种情况。对于每种情况的说明，都是从图 9.4 中的链表开始的。

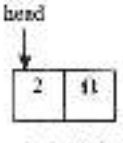


图 9.8　向空链表中插入（需要更新 head 指针）

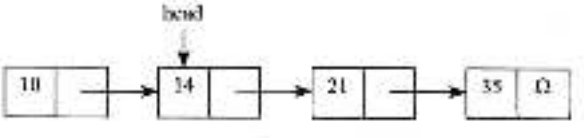


图 9.9　删除第一个元素（需要更新 head 指针）

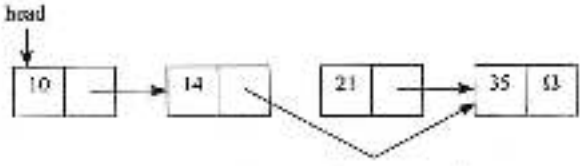


图 9.10 删除两个元素之间的元素

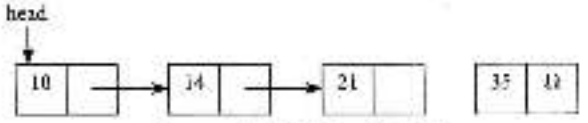


图 9.11 删除最后一个元素

9.7.2 栈

栈是一种后进先出（Last-In-First-Out，LIFO）的数据结构。只能在栈顶插入（push）和删除（pop）元素。因此，从栈中删除的元素总是最后增加的元素。栈是编译器设计的基础，它说明了程序中的函数调用流。例如，每当一个函数被调用时，返回地址就被添加到栈中。每条 return 语句都会引用上次添加到栈中的地址。图 9.12 说明了栈的工作方式。

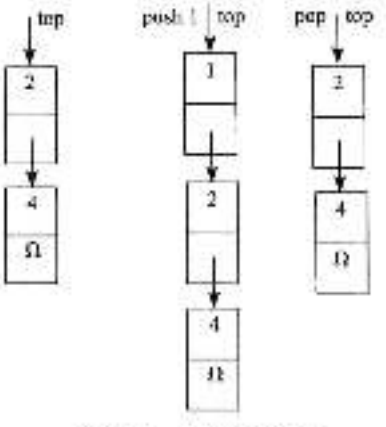


图 9.12 栈的工作方式

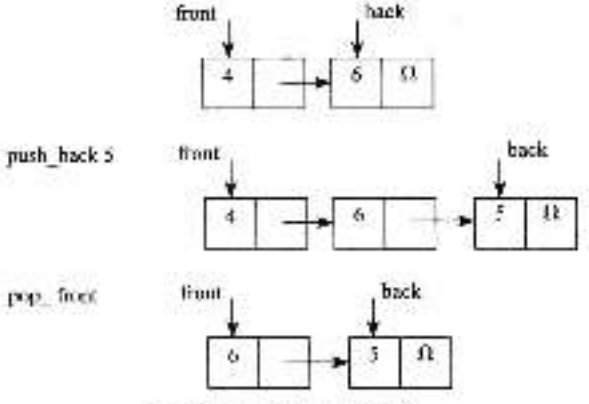


图 9.13 队列的工作方式

9.7.3 队列

队列是一种先进先出（First-In-First-Out，FIFO）的数据结构。只能从队列的后端添加（push）元素，删除（pop）元素只能在队列的前端进行。因此从队列中删除的元素总是第一个添加到队列中的元素。在计算机工程和计算机科学应用中，队列是很有用的数据结构。例如，操作系统使用队列来处理任务请求。任务请求，如请求打印机，都在一个队列中等待处理。在队列前端的任务请求将被首先处理。图 9.13 说明了队列的工作方式。

练习

1. 写一个 stack 类实现，给定的类声明如下：

```
class stack
{
int* a;
```

```
int sizeOfStack;
public:
stack();
void push(int);                    //add value to top of stack
int pop();                         //remove top element
int top();                         //return value of top element
                                   //element is not removed
int isempty();

};
```

2. 修改你的 stack 类，使其变成类型为 double 类型的栈。
3. 修改你的 stack 类，使其变成类型为 point 类型的栈。

9.8　C++ 标准模板库

编写自定义的函数从链式结构中插入和删除元素，该链式结构需要关心内存管理和指针的重新赋值，在第 10 章中完成自定义的二叉树实现时将会谈到。但是，作为实现自己数据结构库的一种选择，你可以选择在 C++ STL 中定义的容器类。容器类，是像 vector 类一样设计用于保存对象的类。在 SGI 网站上（http://www.sgi.com/tech/stl/）有完整的和最新的、有关 STL 的资源。我们的第一个例子将说明 list 类的用法。

9.8.1　list 类

使用 list 类时必须包含编译器指令 #include<list>。当定义一个新的 list 时，list 中对象的数据类型必须指定。下面的类型声明语句定义了一个名为 alist 的空的整数列表：

```
list<int> alist;
```

为了系统地从一个列表中删除或插入元素，我们必须能够遍历列表。list 类提供了一个特殊类型的指针，称为 iterator，以支持对列表中的元素进行顺序访问。一个 iterator 是一个指向列表中元素的指针。我们可以定义名为 iter 的 iterator，通过下面的语句来访问整数列表中的元素：

```
list<int>::iterator iter;
```

list 类包括大量的成员函数来支持列表结构，包括从列表中插入元素和删除元素的函数。表 9.2 中列出了 list 类中若干常用的成员函数。

表 9.2　list 类中的成员函数

begin()	返回指向列表中第一个元素的 iterator
end()	返回一个指向列表中最后一个元素之后的 iterator
empty()	如果列表为空返回 true，否则返回 false。
insert (iterator, value)	在 iterator 指定位置插入值
remove (value)	从列表中移除所有具有指定值的实例
sort()	按照升序排列元素

下面的程序说明了其中一些函数的用法：

```
/*                                                              */
/* Program Chapter9_10                                          */
/*                                                              */
/* This programs creates a list of data entered from
   standard input.                                              */
```

```
/* The list is sorted and printed to standard output.                    */

#include<iostream>//Required for cout, cin
#include<list>//Required for list, begin(), end(), insert(), sort()
using namespace std;

int main()
{

  // Declare objects.
  list<int> alist;
  list<int>::iterator iter;
  int ivalue;

  // Set iter to beginning of alist.
  iter = alist.begin();

  cout << "enter integer values, 's' to stop\n";

  // While valid data, read value and insert into list.
  while(cin >> ivalue)
  {
    alist.insert(iter, ivalue);
    ++iter;
  }

  // Sort the list in ascending order.
  alist.sort();

  // Print the list to standard output.
  cout << "Sorted list: \n";
  for(iter=alist.begin(); iter!=alist.end(); ++iter)
  {
    cout << *iter << endl;
  }
  return 0;
}
```

如果数据

```
35 18 21 14 10 2 s
```

从键盘输入，程序将得到下面的输出：

```
Enter integer values, 's' to stop
Sorted list:
2
10
14
18
21
35
```

语句

```
++iter;
```

并不完成标准的指针算术操作，而只是将 iter 重新指向 list 中的下一个元素。

9.8.2 stack 类

C++ STL 中包含了 stack 类，它提供了栈基本对象的实现。使用 stack 类时必须包括编译器指令 #include<stack>。

当定义 stack 时，必须指明 stack 中对象的数据类型。下面的类型声明语句定义了一个空的整数栈：

```
stack<int> mystack;
```

表 9.3 中列出了 stack 类的成员函数。

表 9.3 stack 类中的成员函数

empty()	如果栈为空返回 true，否则返回 false
pop()	从栈顶移除元素，不返回值
push (value)	将值 value 加入栈顶
size()	返回栈中的元素数目
top()	返回栈中的第一个元素，但并不从栈中移除这个元素

下面的程序说明了其中一些函数的用法：

```
/*------------------------------------------------------------------*/
/* Program chapter9_11                                              */
/*                                                                  */
/* This program creates a stack of data entered from
   standard input.                                                  */
/* The stack is printed to standard output.                         */

#include<iostream>//Required for cout, cin
#include<stack>//Required for stack, push(), top(), empty()
using namespace std;
int main()
{

  // Declare objects.
  stack<int> astack;
  int ivalue;

  cout << "Enter integer values, 'e' to stop\n";

  // While valid data, read value and add to stack.
  while(cin >> ivalue)
  {
     astack.push(ivalue);
  }

  // Print values to standard output.
  cout << "Elements from the stack: \n";
  while(!astack.empty())
  {
  // Access the top element.
     cout << astack.top() << endl;

  // Remove top element from the stack.
     astack.pop();
  }
  return 0;
}
```

如果数据

```
25 18 21 16 e
```

从键盘输入，那么程序将得到下面的输出：

```
Enter integer values, 's' to stop
Elements from the stack:
15
21
18
35
```

注意元素的顺序已经反转了。

9.8.3 queue 类

C++ STL 中包含了 queue 类，它提供了队列基于对象的实现。要使用 queue 类必须包含编译器指令 #include<queue>。当定义 queue 时，必须指明 queue 中对象的数据类型。下面的类型声明语句定义了一个空的整数队列：

```
queue<int> squeue;
```

表 9.4 中列出了 queue 类的成员函数。

表 9.4 queue 类中的成员函数

empty()	如果队列为空返回 true，否则返回 false
pop()	从队列前端移除元素，不返回值
push (value)	将值 value 加入队列后端
size()	返回队列中的元素数目
top()	返回队列前端的元素值，但并不从队列中移除这个元素

下面的程序说明了其中一些函数的用法：

```
/*------------------------------------------------------------------*/
/* Program chapter9_12                                              */
/*                                                                  */
/* This programs creates a queue of data entered from
   standard input.                                                  */
/* The queue is printed to standard output.                         */
#include<iostream>//Required for cout, cin
#include<queue>//Required for queue, push(), empty(), top()
using namespace std;
int main()
{

  // Declare objects.
  queue<int> squeue;
  int ivalue;

  cout << "Enter integer values, 's' to stop\n";

  // While valid data, read value and add to back of queue.
  while(cin >> ivalue)
  {
    squeue.push(ivalue);
  }

  // Print values to standard output.
  cout << "Elements in the queue: \n";
 while(!squeue.empty())
 {
 // Access element at the front of the queue.
    cout << squeue.top() << endl;
```

```
    // Remove element from the front of the queue.
      aqueue.pop();
    }
    return 0;
}
```

如果数据

```
33 19 21 14 s
```

从键盘输入，那么程序将得到下面的输出：

```
Enter integer values, 's' to stop
Elements in the queue:
33
19
21
14
```

9.9 解决应用问题：文本文件的索引

文本文件的索引就是文本文件中按字母顺序排列的所有唯一单词的列表。这里给出了前一个句子的索引：

```
a
an
alphabetical
concordance
file
in
is
list
of
text
the
unique
words
```

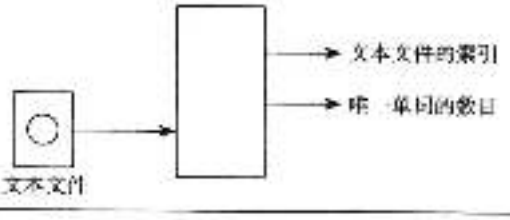
编写一个程序，提示用户输入作为输入文件的文件名以及输出文件的文件名。构建输入文件的索引，并将索引打印到输出文件中。

1. 问题描述

构建一个文本文件的索引。将文本文件的索引和唯一单词的数目写入到输出文件中。

2. 输入/输出描述

下面的 I/O 示意图给出了程序的输入和输出。输入是文本文件，输出是文本文件的索引和文件中唯一单词的数目。我们将使用 list 类来定义一个列表，列表中的元素类型为 string。



3. 用例

假定我们的文本文件只包含下面的文本：

```
A concordance of a text file is an alphabetical list of
the unique words in the text file.
```

我们的程序将生成下面的输出文件：

```
There are 13 distinct words in the text file:
a
an
alphabetical
concordance
file
in
is
list
of
text
the
unique
words
```

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。

分解提纲

1）打开输入输出文件；

2）从输入文件读取一个单词；

3）如果单词不在列表中则将其插入表中；

4）将唯一单词的列表按照字母顺序排列；

5）将列表的大小和内容写入输出文件。

在步骤2中包含一个每次从文本文件中读取一个字符的循环，直到读到一个非字母的字符时为止。非字母的字符标志着单词的结束。除了用作单词间分隔符的字符外，所有的非字母字符都被忽略。所有的字母字符都被转换成小写。我们将使用一个函数来完成这项工作。步骤3要求我们将不在列表中的单词插入列表中。我们将使用一个用到 list 类的成员函数的函数来完成。步骤4涉及列表的升序排列。我们将使用通用函数 sort() 完成。因为列表可能很长，步骤5中在打印列表时将在每行打印3个单词。我们将使用一个函数来完成该项工作。细化后的伪代码如下：

细化的伪代码

```
main:  open input file
         get_word(istream,string)
         while(string size is not 0)
           insert_word(list,string)
           get_word(string)
         sort(list)
         print size of list
         display_list(ostream,list)
get_word(istream,string):
       clear string
       read a character
       while(not end of file and character is not alpha)
          read next character
```

```
        while(not end of file and character is an alpha)
            append lower case character to string
            input next character
insert_word(list, string)
        if(string is not in list)
            insert string
display_list(ostream,list)

        set counter to 0
        while(not end of list)
          print list element
          increment counter
          if(counter mod 3 is zero)
            print newline
```

伪代码中的步骤已经足够详细，可以转换成C++代码：

```
/*-------------------------------------------------------------------*/
/* Program chapter9_13                                               */
/* This program builds a concordance of a text file.                 */
/*                                                                   */
#include <fstream>//Required for ifstream, ofstream
#include <string>//Required for string
#include <cctype>//Required for isalpha(), tolower()
#include <iomanip>//Required for setw()
#include <list>//Required for list, sort(), begin(), end()
#include <algorithm>//Required for find()
using namespace std;

// Function prototypes.
void get_wordlist(ifstream& in_stream, string& w);
void insert_word(string word, list<string> &wordlist);
void display_list(ostream& out_stream, list <string> wordlist);

int main()
{
// Declare objects.
ifstream in_stream;
ofstream out_stream;

string infile, outfile;//filenames
string word;        // string to hold current word

// Prompt for filenames and open files
cout << "Enter the input file name ";
cin >> infile;
cout << "Enter the output file name ";
cin >> outfile;

in_stream.open(infile.c_str());
if(in_stream.fail())
  cout << "Fail to open file " << infile << endl;
else
{
  out_stream.open(outfile.c_str());
  list <string> wordlist;
  list <string>::iterator iter;
  get_word(in_stream,word);  // get a word

  // While non-empty word was returned
  while(word.size())
  {
     insert_word(word, wordlist);
```

```
    get_word(in_stream,word);  // get a word
  }
  wordlist.sort();
  out_stream << "There were " << wordlist.size()
             << " distinct words. \n";
  out_stream << "\nHere is the ordered list of words\n";
  display_list(out_stream,wordlist);

}//end else
return 0;
}
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* This function will insert word into wordlist                      */
/* if word is not found in wordlist.                                 */
/*                                                                   */
void insert_word(string word, list<string> &wordlist)
{
  list<string>::iterator iter;

  iter = find(wordlist.begin(), wordlist.end(), word);

  if( iter == wordlist.end() )
  {
  // Word is not in list.  insert word.
    wordlist.insert(iter, word);
  }
}
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* This function returns the next word from the input stream.        */
/* All non-alpha characters are treated as delimiters.               */
/* The function will ignore all leading non-alpha characters.        */
/* then read and store the following alpha characters                */
/* until it reaches the next non alpha character.                    */
void get_word(istream& in_stream, string& w)
{
  char ch;
  w = ""; //clear word

  in_stream.get(ch);
  while( !isalpha(ch)&& !in_stream.eof() )// skip non-alpha
  {
    in_stream.get(ch);
  }

  while( isalpha(ch) && !in_stream.eof() ) // read and store alpha
  {
    ch = tolower(ch);
    w += ch;
    in_stream.get(ch);
  }
}
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/* This function outputs the list of words to an output stream.      */
/* Three words per column are printer                                */
void display_list(ostream& out_stream, list<string> wordlist)
{
  int columns(3), counter(0);
```

```
    list<string>::iterator iter;
    out_stream << setiosflags(ios::left);

    // Position iter at beginning of list.
    iter = wordlist.begin();
    while(iter != wordlist.end())
    {
       out_stream << setw(20) << (*iter).c_str();
       iter++;
       counter++;
       if(counter%columns == 0)
       {
          out_stream << endl;
       }
    }
}
```

5. 测试

如果我们使用示例中的文本，则写入数据文件的信息将如下所示：

```
There are 13 distinct words in the .ext file:
a                   an                  alphabetical
concordance         file                in
is                  list                of
text                the                 unique
words
```

本章小结

本章我们讨论了存储在对象中的值、指派给对象的标识符以及用于存储对象值的内存单元地址之间的关系。可以定义一个新的指针数据类型来存储另一个对象的地址。同时还给出了使用指针引用数组元素和字符串的例子。定义了动态内存分配的过程。讨论了操作符 new 和 delete，并重新将 vector 类作为动态内存分配的一种选择作了介绍。讨论了包括链表、栈和队列等数据结构，并给出了使用 list 类、stack 类和 queue 类的例子。

关键术语

address（地址）
address operator（地址操作符）
container（容器）
deallocation（取消分配）
dereference（解引用）
dynamic data structures（动态数据结构）
dynamic memory allocation（动态内存分配）
indirection（间接引用）
linked data structures（链式数据结构）
list（链表）
pointer（指针）
queue（队列）
stack（栈）

C++ 语句总结

指针声明

一般形式：

```
数据类型 *标识符 [, *标识符];
```

示例：

```
int *ptr_1;
double a, *ptr_2=&a;
```

动态内存分配

一般形式：

```
指针对象 = new 数据类型;
```

示例：

```
ptr = new double;
```

取消动态内存分配

一般形式：

```
delete 指针对象;
```

动态数组内存分配

```
arr_ptr = new double[100];
```

取消动态数组内存分配

一般形式：

```
delete 指针对象[];
```

注意事项

1. 使用比较明确的标识符作为指针的名字，以表明这些标识符与指针对象相关。
2. 如果指针没有使用一个内存单元地址进行初始化，则为它赋值为 NULL，以表示它还没有被分配地址。

调试要点

1. 在其他语句中使用对象时，确保该对象在程序中进行了初始化。
2. 在使用指针引用某个值之前确定指针对象已经被初始化了。
3. 对应于函数中指针类型参数的实参必须是一个地址或者指针。

习题

判断题

1. 地址操作符和间接引用操作符都是一元操作符。
2. 指针提供了一种间接访问某个特定项目的值的方式。
3. 对象必须总是在指针指向它之前定义和初始化。
4. 分配给动态内存空间的内存地址在程序编译时确定。

多选题

5. 存储空间中的内存单元（　　）。
   (a) 在对象声明时保留　　(b) 在对象在程序中使用时保留
   (c) 可以同时存储几个不同的值　　(d) 一旦被赋值后就不能重新使用了
6. 指针对象（　　）。
   (a) 包含了存储在内存单元里的数据
   (b) 包含了一个内存单元的地址
   (c) 可以用在输入语句中，但不能用于输出语句
   (d) 在输入和输出语句中都可以被修改成不同的值

7. 你会用（　　）将被指针 a 所引用的对象的值赋给对象 name。

(a) a = &name;　　(b) name = &a;

(c) a = *name;　　(d) name = *a;

8. 假定 a 和 b 都是指向整数的指针，其中 a 指向对象 name。那么下面语句的作用是（　　）。

```
b = a;
```

(a) 将 name 的值复制到 b 中　　(b) 将 a 中存储的地址复制到 b 中

(c) 将 b 中存储的地址复制到 a 中　　(d) 指针 a 现在指向了一个不同的对象

内存快照问题

9. 给出下面的一组语句执行后对应的内存快照：

```
double name, x[20.5];
double *a = &x;
name = *a;
```

假定 name 的地址为 10，x 的地址为 14（即 name 存储在内存单元 10 的位置，x 存储在内存单元 14 的位置）。问题 10 ~ 13 中都用到了下面的语句：

```
int i1, i2;
int *p1, *p2;
i1 = 5;
p1 = &i1;
i2 = *p1/2 + 10;
p2 = p1;
```

10. i1 的值是多少？

11. i2 的值是多少？

12. *p1 的值是多少？

13. *p2 的值是多少？

vector 函数。这些问题要求开发用于操作 vector 中数值的函数。

14. 编写一个函数，创建一个大小为 n 的 vector，其中的每个元素都赋值为 v。假定函数原型为

```
void assign_nivector <double> &x, int n, double v);
```

15. 编写一个函数，创建一个大小为 n 的 vector，其中的每个元素都是 a 到 b 之间的随机数。假定函数原型为

```
void assign_random(vector <int> &x, int n, int a, int b);
```

16. 编写一个函数，计算 vector 中数值的和。假定函数原型为

```
int v_sum(vector<int> x);
```

17. 编写一个函数，将 vector 中的值都替换成原值的绝对值。假定函数原型为

```
void v_abs(vector<int>&x);
```

字符函数：模式识别。在许多工程领域的解决方案中都需要在某个信号信息中搜索特定的模式。问题 18 ~ 25 开发了一组与此目的相关的函数。

18. 编写一个函数，接收一个指向字符串的指针和一个字符作为参数。函数应当返回字符在字符串中出现的次数。假定函数原型为

```
int charcnt(char *ptr, char c);
```

19. 编写一个函数，接收一个指向字符串的指针作为参数，返回字符串中重复字符的数目。例如，在字符串中“Mississippi”中有三个重复的字符。不要统计字符串中重复的空格。如果字符出现了两次以上，只统计作为 1 次重复；因此“hisssss”只有一个重复的字符。假定函数原型为

```
int repeat(char *ptr);
```

20. 重写问题 19 中的函数，使其将每对字符都统计为一次重复。因此，字符串“hissss”有四个重复的字符。假定函数原型为

```
int repeat2(char *ptr);
```

21. 编写一个函数，接收两个指向字符串的指针作为参数，返回第二个字符串在第一个字符串中出现的次数。不允许出现重叠。因此，字符串“110101”中只出现了一次“101”。假定函数原型为

```
int overlap(char *ptr1, char *ptr2);
```

22. 重写问题 21 中的函数，使其统计出现次数时允许重叠。因此，字符串“110101”中出现了 2 次“101”。假定函数原型为

```
int overlap(char *ptr1, char *ptr2);
```

23. 重写问题 19 中的函数，使用 string 类替代 C 风格字符串。假定函数原型为

```
int repeat(string);
```

24. 重写问题 20 中的函数，使用 string 类替代 C 风格字符串。假定函数原型为

```
int repeat2(string);
```

25. 重写问题 21 中的函数，使用 string 类替代 C 风格字符串。假定函数原型为

```
int pattern(string, string);
```

26. 修改程序 chapter 9_13，将所有单词都插入列表中，然后使用成员函数 unique() 删除重复的元素。
27. 修改程序 chapter 9_13，对文本文件中每个唯一的单词出现的次数进行统计。
28. 编写一个函数，将数组 x 的前 n 个元素赋值为 v。使用原型：

```
void assignV(double *x, int n, double v);
```

29. 编写一个函数，将数组 x 的前 n 个元素都赋值为随机数。使用原型：

```
void assignRandom(double *x, int n);
```

高级主题

工程挑战：人工智能

人工智能（Artificial Intelligence，AI）是一个有趣的科学领域，它关注创建具有“思考”能力的智能机器。有关机器人、认知科学、语言、计算和感官知觉的研究促进了智能机器的创造。如今，已经存在一些软件，可以与人类客户进行对话，击败世界级的国际象棋选手，帮助诊断疾病。虚拟宠物可以“学习”它们的环境并回应它们的主人。今天已经存在的智能软件只是将要出现的软件的一小部分。

教学目标

本章我们所讨论的问题解决方案中包括：

- 函数模板
- 操作符重载
- 图像处理示例
- 迭代的成员函数
- 类模板
- 类继承
- 可重复的囚徒困境游戏的一种实现

10.1 泛型编程

泛型编程是一种支持一种算法或类型的实现的编程形式，其中使用一组参数而非特定的类型。C++ 编程中通过使用模板（template）来支持泛型编程。模板是一种通用算法或通用类型的形式化定义，它可以由编译器使用特定的类型来实现或实例化。定义一种概念的能力并针对不同的类型自动实现这种能力，在问题解决方案的开发和测试中可以帮助我们节省时间。

在前面章节我们所开发的问题解决方案中，已经用到了 C++ 标准库中的预定义模板，包括 vector<type> 和 random_shuffle()。本章我们将介绍自定义模板的用法，通过编写一组函数模板来完成对多种数据集的统计测量，同时编写了一个类模板来实现二叉树的概念。

使用模板定义算法和类型的能力是 C++ 编程语言的一个强大特性，但是也具有一些局限性。不是所有的编译器、链接器和调试器对模板都能很好地支持。因此，使用模板的代码可能更难编译和调试，代码的可移植性也减弱了。

为了最大限度地减少花费在调试模板上的时间，一个好的办法是首先为某个特定类型开发函数或类的一种可工作的版本。一旦一种可工作的版本完全通过了测试和调试，那么就可以定义一个模板，并对模板进行测试，与工作版本的结果和性能进行比较。为了可移植性的考虑，以及消除某些潜在的链接器错误，在问题解决方案中，我们将使用 #include 语句包含定义了我们的模板的源文件，因此消除了链接到对象文件的需求。

函数模板

函数模板使用函数定义作为参数，而非一个数据结构。当一个函数模板使用某个特定类型作为参数时，编译器将会生成一个模板的实例，将其中的参数用特定类型的函数参数替换。因此，函数模板编写一次后，就可以按照需要由编译器生成多个版本的模板实例。

第 7 章我们编写了一系列的函数来对一组数据进行统计测量。每个函数都假定数据在一个 double 类型的数组中。本节我们将开发一组函数模板来完成统计测量，然后使用这些模板对第 7 章开发的函数进行测试。我们从函数 minval() 开始。函数 minval() 的定义摘自第 7 章，重复如下所示：

```
/*-------------------------------------------------------*/
/* This function returns the minimum                     */
/* value in an array x with n elements.                  */
double minval(const double x[], int n)
{
// Declare objects.
double min_x;
// Determine minimum value in the array.
min_x = x[0];
for (int k=1; k<n; ++k)
{
   if (x[k] < min_x)
      min_x = x[k];
}
// Return maximum value.
return min_x;

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

函数 minval() 返回在 double 类型数组中找到的最小值。找出数组中最小值的算法独立于数组的数据类型，因此 minval() 是函数模板的一个好的候选者。

函数模板的定义以下面表达式的形式开始：

```
template<typename identifier>
```

表达式中的标识符提供了一个参数化类型的名字，该标识符用于在函数定义中替换某种特定的类型，名为 minVal() 的函数模板定义如下所示：

```
/*-------------------------------------------------------*/
/* Function returns the minimum value in an              */
/* array of type Dtype and size n.                       */

template <typename Dtype>
Dtype minVal(const Dtype x[], int n)
{

// Declare objects.
   Dtype minX;
// Determine minimum value in the array.
   minX = x[0];
   for (int k=1; k<n; ++k)
   {
      if (x[k] < minX)
         minX = x[k];
   }
// Return minimum value.
   return minX;
}
/*-------------------------------------------------------*/
```

注意在定义函数 minval() 和定义函数模板 minVal() 的代码中唯一的差别就是将 minval() 中的数据类型 double 替换成参数 Dtype。该函数模板的原型如下：

```
template <typename Dtype>
Dtype minVal(const Dtype x[], int n);
```

函数模板：函数模板就是一个参数化的函数定义。

语法

```
template<typename 标识符 1, 标识符 2,…1>
返回类型 函数名(参数列表)
{
    //语句块
}
```

示例

```
template<typename Dtype>
void swapTwo(Dtype& a, Dtype& b)
{
  Dtype temp = a;
  a = b;
  b = temp;
}
```

原型

```
template<typename Dtype>
void swapTwo(Dtype& a, Dtype& b);
```

在函数模板定义和函数模板原型之前，必须有关键词 template 及其后所跟的由 <> 所包含的参数列表。在参数列表中，关键词 typename 位于列表中每个标识符之前，表明标识符代表一种数据类型。模板参数列表中必须至少包含一个参数。如果使用了多个参数，则它们必须使用逗号分隔，如：

```
template <typename T1, typename T2, typename T3>
```

下面的程序调用了 4 次函数模板 minVal()，分别用来找出类型分别为 int、char、double 和 string 的数组中的最小值。

```
/*                                                                    */
/* Program chapter10_1                                                */
/* This program demonstrates the use of the function                  */
/* template minVal.                                                   */

#include<iostream> //Required for cout.
#include<string> //Required for string.
using namespace std;

//Function Prototype
template <typename Dtype>
Dtype minVal(const Dtype x[], int n);

int main()
{
 //Declare objects.
 const int SIZE = 10;
```

```
    char ch[SIZE] = {'h','e','l','l','o','w','o','r','l','d'};
    int iDat[SIZE] = {5,2,7,8,2,5,9,8,1,9};
    double dDat[SIZE] = {-2.1,4.3,0.0,9.3,0.4,-4.2};
    string sDat[SIZE] = {"this","short","the","list","of","strings"};

    //Print smallest value in each array.
    cout << "smallest char in ch is "
         << minVal(ch,SIZE) << endl;        // Char
    cout << "smallest integer in iDat is "
         << minVal(iDat,SIZE) << endl;      // int
    cout << "smallest double in dDat is "
         << minVal(dDat,6) << endl;         // double
    cout << "smallest string in sDat is: "
         << minVal(sDat,6) << endl;         // string

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

程序一次运行的输出如下所示：

```
smallest char in ch is: d
smallest integer in iDat is: 1
smallest double in dDat is: -4.2
smallest string in sDat is: list
```

函数模板 minVal() 使用两个操作符，“<”和“=”，来找出数组 x 中的最小值。因此，minVal() 可以被任何定义了这两个操作符的数据类型调用。操作符“=”默认为所有类型都有定义，但操作符“<”并非都有定义。例如，如果我们希望调用 minVal() 来对类型为 Card 的数组进行操作，编译器将会给出一长串的编译错误，因为我们没有在 Card 类的定义中重载这个操作符。

如果我们可以定义怎样表示一张牌比另一张牌小，则可以在 Card 类中添加一个布尔方法来重载“<”操作符。当为一个新的类型重载操作符“<”时，应该保证对所有类型的对象都满足如果 c1 < c2 且 c2 < c3，则 c1 < c3。这对所有的关系操作符都正确。重载操作符将在后面的小节中讨论。

修改

问题 1～4 与第 7 章中定义的函数有关。

1. 为下面的函数编写一个函数模板：

```
double mean(const double x[], int n);
```

使用至少两种数据类型测试你的模板。

2. 为下面的函数编写一个函数模板：

```
double median(const double x[], int n);
```

使用至少两种数据类型测试你的模板。

3. 为下面的函数编写一个函数模板：

```
double variance(const double x[], int n);
```

使用至少两种数据类型测试你的模板。

4. 为下面的函数编写一个函数模板：

```
double std_dev(const double x[], int n);
```

使用至少两种数据类型测试你的模板。

10.2 数据抽象

在实际的应用中，常常会碰到一些无法使用内建或预定义数据类型的情况。实际的应用解决方案常需要多个程序员来处理解决方案的不同方面。在这些情况下，就需要一种面向对象的方法来设计和实现解决方案。定义一种新的类型来表示某个概念，以确保所有的程序员对于这个概念都使用相同的物理定义，并且遵循相同的操作规则，这样减少了潜在的错误，提高了生产率。自定义类型也可以被用来定义更复杂的概念。

C++程序语言通过使用自定义类型或者叫做数据抽象（data abstraction），来支持面向对象的编程。C++支持使用类来定义新的类型，这些类使用起来与内建类型一样简单，并且如内建类型一样受到几乎同样多的编译器的支持。一个设计良好的类型提供一个好的公共接口，并通过封装隐藏类型的实现。封装要求直接访问类型中数据成员的权限被限制在类方法中。回忆自定义类 Point 中使用关键字 private 来限制对数据成员的访问权限。封装和好的公共接口有利于类型的维护和扩展，而不需要对使用这些类型的应用作出修改。

C++中支持的操作符重载允许自定义类型使用起来如同预定义类型一样简单。我们首先介绍第3章中提到的操作符重载，其中我们定义了操作符 Point::operator-() 用于计算一个平面上两点之间的距离。本节我们将看到有关操作符重载的更多细节，并使用关键字 friend 来重载输入和输出操作符“<<”和“>>”。

10.2.1 操作符重载

操作符重载允许自定义类型在某些限制条件下，重新定义已存在的操作符的行为。C++中所有预定义的操作符，除了其中的4个例外，其他的操作符都可以被重载。不能定义新的操作符，如定义 ** 作为指数操作。表10.1中列出了不能被重载的4个操作符。

表10.1 不能被重载的操作符

操作符	描述	操作符	描述
::	域解析操作符	.*	成员指针操作符
.	点操作符	?:	条件操作符

C++中的操作符有定义它们用途的语法。像 ++x 这样的一元操作符要求一个操作符，像 x + y 这样的二元操作符要求两个操作数。当 C++ 中重载操作符时，必须遵守定义操作符的语法。因此，在重载一个二元操作符时，你必须提供两个操作数，当重载一个一元操作符时，必须提供一个操作数。为了说明操作符重载，我们将设计一个新的类型用来实现像素的概念。

10.2.2 像素类

数字图像可以被描述成图像元素（picture element）的集合。图像元素是一个小的矩形区域，也称作像素（pixel）。在彩色图像中，每个像素代表着一个红、绿、蓝的三元组（red, green and blue triple）。一般而言，三元组中每个值的范围都是0～255，这里0表示在像素中完全不呈现出某种原色，255表示呈现出原色的最大量。因此，表示白色的像素值为（255, 255, 255），表示黑色的像素值为（0, 0, 0）。

数字图像可以通过操作表示图像的像素来进行修改。例如，我们可以通过对每个像素乘上一个大于1的正数，使每个像素更接近白色，从而使图像整体变亮（brighten）。也可

以通过对每个像素乘上一个小于1的正数，使每个像素更接近黑色，从而使图像整体变暗(darken)。

当我们将像素乘以一个值时，我们希望将像素中的三个颜色值——红色值、绿色值和蓝色值——都同时乘上相同的值，我们希望使用乘法操作符"*"在一次乘法中完成这些工作。因此，我们在定义像素类时重载乘法操作符。

另一种对数字图像进行的修改称作平滑化(smoothing)。要使图像平滑，我们将每个像素使用其周围像素的平均值替代。我们选择包含的周围像素的数目称作邻域(neighborhood)。邻域的大小决定了平滑化的程度：邻域越大，平滑度越好。例如，我们可以通过替换除了图像边界上的像素外的其他所有像素来进行图像平滑，替换的像素是被替换像素周围4个像素——上下左右像素——的平均值。为了计算平均值，我们需要计算像素的值和并除以像素的数目。当我们将两个像素加和时，我们希望将红色值、绿色值、蓝色值分别加和。当我们将一个像素除上一个值，我们希望将红色值、绿色值和蓝色值都除上相同的值。因此，我们将重载加法操作符和除法操作符。

我们已经标识出了像素的属性为红、绿、蓝三元组，同时确定了修改像素值需要定义的三种算术操作(加法、乘法和除法)。Pixel类的声明在下面给出。我们现在将实现Pixel类，使用下面所给出的类声明。

```
/*-------------------------------------------------------------*/
/* Pixel class declaration.                                    */
/* File name: Pixel.h                                          */
/* This class implements the concept of a pixel                */
#include <iostream> //Required for istream, ostream
using namespace std;
class Pixel
{
  public:
  //Constructors
  Pixel();                          //Default
  Pixel(unsigned );                 //Gray scale
  Pixel(unsigned,unsigned,unsigned); //Full color range

  //Overloaded operators.
  //Addition.
  Pixel operator+(const Pixel& p) const;

  //Multiplication of a Pixel by a floating point value.
  Pixel operator*(double v) const;

  //Division of a Pixel by an integer value.
  Pixel operator/(unsigned v) const;

  //Input operator.
  friend istream& operator >>(istream& in, Pixel& p);

  //Output operator.
  friend ostream& operator <<(ostream& out, const Pixel& p);

  private:
  unsigned int red, green, blue;
};
/*-------------------------------------------------------------*/
```

类声明中包含了一组构造函数的原型；算术操作符+，-和*，以及操作符<<和>>。在输入、输出操作符的原型中使用了关键字friend(友元)。友元函数不是类的成员函数，但是关键字friend为函数提供了访问友元类private和protected数据成员的权限。这违背了封

装的准则，但是使得自定义类型可以以内建类型同样的方式被使用。操作符的重载和友元函数将在本节后面进行讨论。

10.2.3 算术操作符

我们的 Pixel 类声明中包括了下面的原型：

```
Pixel operator+(Pixel p) const;
Pixel operator*(double v) const;
Pixel operator/(unsigned v) const;
```

每个原型的返回类型都为 Pixel，这说明每个方法都将返回一个 Pixel 类型的值。同时每个方法都有一个形参。在每个方法中，形参是值传递的参数，这防止了参数被修改。关键字 const 防止了调用对象被修改。

+ 操作符。方法 Pixel operator+（Pixel p）const 有一个 Pixel 类型的形参。因为该方法是 Pixel 类的成员，它将被一个 Pixel 类型的对象调用。调用对象将为“+”操作符提供第一个操作数。参数将提供第二个操作数。重载的加法操作符的实现如下。

```
/*                                                                  */
/* Addition (+) operator.                                           */
/* File name: Pixel.cpp                                             */
#include "Pixel.h"

Pixel Pixel::operator+(Pixel p) const
{
   Pixel temp;

   temp.red = red + p.red;
   temp.green = green + p.green;
   temp.blue = blue + p.blue;
   return temp;
}
/*                                                                  */
```

我们可以以下面给出的形式来调用 + 操作符：

```
Pixel p1, p2(100,200,100), p3(10,20,30);
p1 = p2 + p3;  //function call
cout << p1;
...
```

在语句

```
p1 = p2 + p3;
```

中，p2 是调用对象，p3 是参数。当“+”操作符被调用时，形参 p 接受参数 p3 的值。该方法引用了调用对象的私有数据成员（red、green 和 blue）和形参的私有数据成员（p.red、p.green、p.blue）。函数还引用了局部对象 temp 的私有数据成员（temp.red、temp.green 和 temp.blue）。在函数定义中的下面 3 个赋值语句

```
temp.red = red + p.red;
temp.green = green + p.green;
temp.blue = blue + p.blue;
```

将函数参数与调用对象相加的结果赋给了局部对象 temp。temp 的值被返回，并赋给了 Pixel p1。当 p1 被打印在屏幕上，生成的输出为

```
110 220 130
```

图 10.1 中给出了这些语句的程序跟踪。

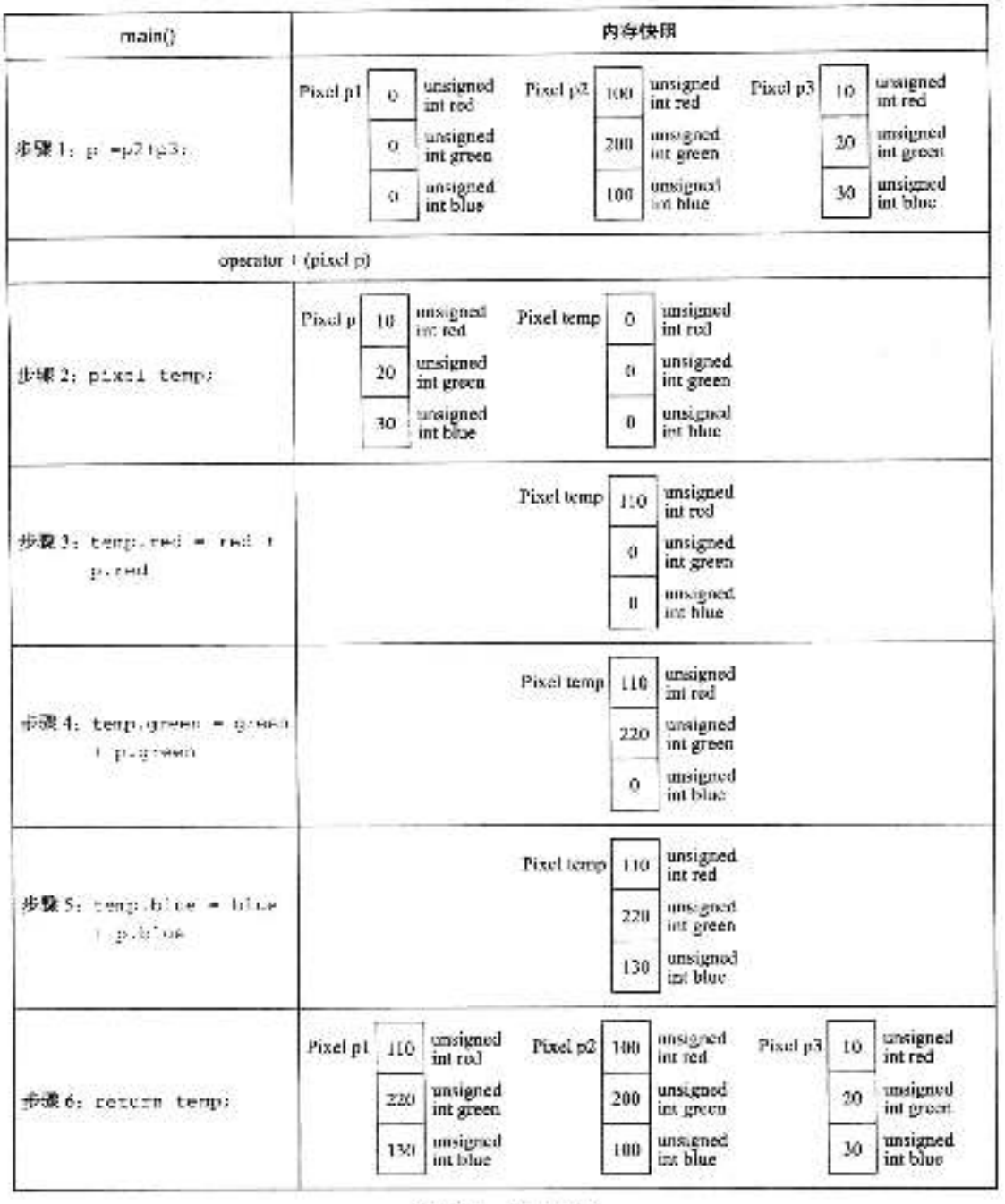


图 10.1 程序跟踪

操作符。函数 Pixel operator（double v）const 有一个 double 类型的形参，调用对象为 * 操作符提供第一个操作数，函数参数提供第二个操作数。下面是函数定义。

```
/*                                                                    */
/* Multiplication (*) operator.                                       */
/* File name: Pixel.cpp                                               */
```

```
Pixel Pixel:: operator*(double v) const
{
   Pixel temp;
   temp.red = red*v;
   temp.green = green*v;
   temp.blue = blue*v;
   return temp;
}
/*-------------------------------------------------------------*/
```

我们可以像下面这样调用 operator* 函数：

```
Pixel p1, p2(100,200,100);
double factor(1.2);
p1 = p2*factor;
cout << p1;
```

在本例中，p2 是调用对象，factor 是函数参数。当函数 operator* 被调用时，形参 v 接收 factor 的值。调用对象的每个数据成员与浮点值 v 相乘的结果被赋给局部对象 temp。temp 的值被返回并赋给 Pixel p1。当 p1 被打印到屏幕上时，生成的输出为

```
120 240 120
```

图 10.2 中给出了程序跟踪，用来说明信息的传递。

main()	内存快照
步骤1：Pixel p1 p2 (100,200,100);	Pixel p1: 0 unsigned int red; 0 unsigned int green; 0 unsigned int blue Pixel p2: 100 unsigned int red; 200 unsigned int green; 100 unsigned int blue
步骤2：double factor (1.2)	double factor 1.2
步骤3：p1 = p2*factor;	
operator * (double v)	
步骤4：Pixel temp;	Pixel temp: 0 unsigned int red; 0 unsigned int green; 0 unsigned int blue double v 1.2
步骤5：temp.red = red*v; 步骤6：temp.green = green*v; 步骤7：temp.blue = blue*v;	Pixel temp: 120 unsigned int red; 240 unsigned int green; 120 unsigned int blue
步骤8：return temp;	
	Pixel p1: 120 unsigned int red; 240 unsigned int green; 120 unsigned int blue Pixel p2: 100 unsigned int red; 200 unsigned int green; 100 unsigned int blue

图 10.2 程序跟踪

注意，该函数没有定义两个像素之间的乘法，而定义了一个像素和一个 double 类型之间的乘法。如果我们希望定义两个像素之间的乘法，则可以在 Pixel 类的定义中通过下面的原型来包含该函数：

```
Pixel operator*(Pixel p) const;
```

/ 操作符。函数 Pixel operator/(int v) const 有一个 int 类型的形参。调用对象将为 "/" 操作符提供第一个操作数，函数参数将提供第二个参数。函数定义如下：

```
/*-------------------------------------------------------------------*/
/* Division (/) operator.                                            */
/* File name: Pixel.cpp                                              */

Pixel Pixel::operator/(int v) const
{
   Pixel temp;
   temp.red = red/v;
   temp.green = green/v;
   temp.blue = blue/v;
   return temp;
}
/*-------------------------------------------------------------------*/
```

我们可以按照下面的方式调用该函数：

```
Pixel p1, p2(100,200,100);
p1 = p2/2;
cout << p1;
```

在本例中，p2 是调用对象，整数 2 是函数的参数。当函数被调用时，形参 v 接收参数的值。调用对象与整数值除得的商被赋给局部对象 temp。temp 的值被返回，并赋给 Pixel p1。当 p1 被打印到屏幕上，生成的输出为

```
50 100 50
```

注意，该函数没有定义两个像素的除法，而是定义了像素与一个 int 类型的除法。如果我们希望定义两个像素的除法，我们可以在 Pixel 类的定义中通过下面的原型来包含该函数：

```
Pixel operator/(Pixel p) const;
```

上面定义的重载操作符函数都是 Pixel 类的成员函数。因此，每个函数都必须被一个 Pixel 对象调用。这限制了这些操作符的使用，因为第一个操作数必须是一个 Pixel 对象。考虑下面使用 "+" 操作符的代码段：

```
Pixel p1, p2(100,200,100), p3(10,20,30);
p1 = p2 + p3; // Valid.
p1 = p2 + 100; // Valid.
p1 = 100 + p2; // Invalid!
```

第一条赋值语句

```
p1 = p2 + p3; // Valid.
```

是合法的，因为 "+" 操作符的两个操作数都是 Pixel 对象。第二条赋值语句

```
p1 = p2 + 100; // Valid.
```

也是合法的，因为第一个操作数即调用对象是一个 Pixel 对象。第二个操作数是一个整数，

不是一个 Pixel 对象。但是，因为我们的 Pixel 类有一个带整数参数的构造函数，这个构造函数可以将整数 100 提升成值为（100,100,100）的 Pixel 对象，所以函数调用成功。第三条语句

```
p1 = 100 - p2; // Invalid!
```

非法，因为第一个操作数即调用对象是一个整数对象，而不是一个 Pixel 对象。这条语句在编译时将出错。这种限制可以通过将操作符定义为友元函数（而非成员函数）得到消除。

练习

假定下面的原型已经被添加到 Pixel 类声明中了。为下面的每个方法写出函数定义。

```
1. pixel operator/(pixel p1); //divide a pixel by a pixel
2. bool operator==(pixel p1); //return true if p1 is
                              //equivalent to calling object
```

10.2.4 友元函数

友元函数不是成员函数，但是友元函数对类所包含的私有数据成员具有访问权限。为了将函数声明为友元函数，在类声明中的函数原型必须以关键字 friend 开始。因为友元函数不是成员函数，所以函数不受 public 或 private 关键字的影响。在 Pixel 类中，我们将友元函数包含在 public 节中，与其他的操作符函数放在一起。

如果我们希望将“+”操作符定义为友元函数，而不是成员函数，它的原型应该是：

```
friend Pixel operator+(Pixel p1, Pixel p2);
```

注意函数原型需要两个形参，因为友元函数不是成员函数，没有调用对象。相反，我们必须在函数原型和函数头中将两个操作数都作为形参。下面的代码段说明了函数的用法：

```
Pixel p1, p2(100,200,150), p3(10,20,30);
p1 = p2 + p3; // Valid.
p1 = p2 + 100; // Valid.
p1 = 100 + p2; // Valid!
```

第三条赋值语句

```
p1 = 100 + p2; // Valid!
```

现在是合法的语句。整数 100 为“+”函数提供了第一个参数，p2 提供了第二个参数。因为函数需要 Pixel 对象作为参数，这里使用了构造器将整数 100 提升为 Pixel 对象（100, 100, 100），所以函数调用成功。函数定义如下：

```
/*-------------------------------------------------------------------*/
/* Addition (+) operator as friend function.                         */
/* File name: Pixel.cpp                                              */

Pixel operator +(Pixel p1, Pixel p2)
{
   Pixel temp;
   temp.red = p1.red + p2.red;
   temp.green = p1.green + p2.green;
   temp.blue = p1.blue + p2.blue;
   return temp;
}
/*-------------------------------------------------------------------*/
```

注意关键字 friend 没有包含在函数定义中。友元状态只能在类定义中被保证。将该函数定义与成员函数定义进行比较。因为该函数不是成员函数，没有调用对象，所以在成员函数中对 red、green 和 blue 的引用，在友元函数中都被 p1.red、p1.green 和 p1.blue 所替代。

练习

假定下面的函数原型都已经被添加到了 Pixel 类的声明中。写出下面每个函数的函数定义。

```
1. friend Pixel operator-(Pixel p1, Pixel p2);
2. friend Pixel operator/(Pixel p1, Pixel p2);
3. friend Pixel operator !(Pixel p1, Pixel p2);
```

<< 操作符。我们的 Pixel 类中包含了一个用于重载 "<<" 操作符的函数。函数原型如下：

```
friend ostream& operator<<(ostream& out, const Pixel& p);
```

该函数被定义为带两个形参的友元函数，并返回一个 ostream 引用。返回 ostream 引用可以支持链式操作。因此，我们可以在一条输出语句中输出多个表达式，如下面语句所示：

```
cout << p1 << ',' << p2 << endl;
```

"<<" 操作符不能被定义成 Pixel 类的成员函数，因为函数的第一个参数必须是一个 ostream 的引用。但是，函数需要访问 Pixel 参数的私有数据成员，所以我们将它声明为 Pixel 类的一个友元函数。考虑到效率，我们选择使第二个形参成为一个 const 的引用传递，而非值传递。这允许通过引用传递，但阻止对引用所指向对象的修改。函数定义如下：

```
/*-------------------------------------------------------------*/
/* Output << operator.                                         */
/* File name: Pixel.cpp                                        */
ostream& operator<<(ostream& out, const Pixel& p)
{
   out << p.red << ' ';
   out << p.green << ' ';
   out << p.blue;
   return out;
}
/*-------------------------------------------------------------*/
```

函数必须返回一个 ostream 的引用，因此语句

```
return out;
```

是必需的。

我们的函数输出 Pixel p 中 red、green 和 blue 的值，用空白分隔。我们可以像下面这样使用函数：

```
Pixel p1, p2(100,200,100);
cout << "P1: " << p1 << "  P2: " << p2 << endl;
...
```

打印到屏幕的输出为

```
P1: 0 0 0  P2: 100 200 100
```

我们可以使用 "<<" 函数将输出写入文件中。回忆 ofstream 类是派生自 ostream 类的。因此，所有 ostream 的操作都可以应用到 ofstream 对象上。下面的代码段使用我们的 "<<" 函数将输出写入文件 pixel.dat 中：

```
...
pixel p1, p2(255);
ofstream outfile("pixel.dat");
outfile << "P1: " << p1 << "  P2: " << p2 << endl;
...
```

写入到文件 pixel.dat 中的输出是：

```
P1: 0 0 0  P2: 255 255 255
```

注意我们的 "<<" 函数没有在 Pixel 的尾后重输出一个新行，这允许我们将多个 Pixel 的值输出到同一行上，同时保持了我们与内建数据类型所定义的输出操作符在性能上的一致性。

>> 操作符。我们的 Pixel 类声明了重载 ">>" 操作符的函数。函数原型如下：

```
friend istream& operator>>(istream& in, Pixel& p);
```

该函数被定义为带有两个参数的友元函数，并返回一个 istream 的引用。">>" 操作符不能定义成 Pixel 类的成员函数，因为第一个参数必须是一个 istream 引用。但是，函数需要访问 Pixel 参数的私有数据成员，所以我们要将函数声明成 Pixel 类的友元函数。

因为函数被定义成输入一个 Pixel 值，所以函数必须修改形参 p 引用的对象。因此，形参必须是一个引用传递。函数定义如下：

```
/*
/*-------------------------------------------------------------------*/
/* Input (>>) operator.                                              */
/* File name: Pixel.cpp                                              */

istream& operator>>(istream& in, Pixel& p)
{
   in >> p.red >> p.green >> p.blue;
   return in;
}
/*-------------------------------------------------------------------*/
```

该函数必须返回一个 istream 引用。

重载 ">>" 操作符与重载 "<<" 操作符类似，除了在输入时 ">>" 可能遇到错误的可能更大。函数试图输入三个整数值，用来赋给 p 的 red、green 和 blue 的数据成员。该函数在输入流中查找用空白分隔的三个整数。任何输入流中非预期的字符，如逗号或冒号，都将使 istream 出现 fail 状态。

如果我们的 Pixel 类需要特殊的输出的格式，则 ">>" 函数将需要检查需要的格式。如果没有遇到需要的格式，函数需要将 istream 置为 fail 状态。为了说明，假定输入的 Pixel 的格式要求整数值使用冒号分开，如

```
100:150:100
```

我们将修改前面的函数来检查这种格式。如果格式不正确，就将 istream 置为错误状态：

```
/*-------------------------------------------------------------------*/
/* Input (>>) operator.                                              */
/* Expected Pixel format is 100:150:100                              */
/* File name: Pixel.cpp                                              */

istream& operator>>(istream& in, Pixel& p)
{
// Declare local objects.
   char ch;
```

```
// Input integer value for red data member.
   in >> p.red;

// Input colon.  If colon not encountered,
// set error state and return.
   in >> ch;
   if(ch != ':')
   {
      in.setstate(ios::failbit);
      return in;
   }

// Input integer value for green data member.
   in >> p.green;

// Input colon.  If colon not encountered,
// set error state and return.
   in >> ch;
   if(ch != ':')
   {
      in.setstate(ios::failbit);
      return in;
   }

// Input integer value for blue data member.
   in >> p.blue;

   return in;
}
/*-------------------------------------------------------------------*/
```

在函数定义中，函数 setstate() 用于将 istream 置为 fail 状态。函数 setstate() 是 istream 类的一个成员函数。在语句

```
in.setstate(ios::failbit);
```

中，istream 对象 in 调用了 setstate 函数，将 failbit 置为真。

使用 Pixel 类的另一个潜在错误就是将一个大于 255 的值赋给一个或多个数据成员。后面的小节将讨论处理潜在的溢出错误。

10.2.5 验证对象

我们将一个像素定义为红、绿、蓝的三元组，其中每个值的范围都是 0 ～ 255。Pixel 类包括三个 unsigned int 类型的数据成员（red、green 和 blue）。数据类型 unsigned int 防止出现负值，但是它不能防止一个值超过最大值 255。考虑下面使用 Pixel 类的例子：

```
/*-------------------------------------------------------------------*/
/* Program chapter10_2                                               */
/* Driver program for testing Pixel class                            */
#include "Pixel.h"  //Required for Pixel
#include <iostream> //Required for cout
using namespace std;
int main()
{
  //Test constructors
  Pixel defaultP;
  Pixel grayP(100);
  Pixel redP(255,0,0);
```

```
    //Test output operator
    cout << "Default pixel: " << defaultP << endl;
    cout << "Gray pixel: " << grayP << endl;
    cout << "Red pixel: " << redP << endl;

    //Test arithmetic operators
    //Addition
    defaultP = grayP + redP;
    cout << "After addition, defaultP: " << defaultP << endl;
    return 0;
}
```

程序的一次示例运行如下：

```
Inghers-MacBook-Pro:Programs jeinghers$ g++ main.cpp Pixel.cpp
Inghers-MacBook-Pro:Programs jeinghers$ ./a.out
Default pixel: 0 0 0
Gray pixel: 100 100 100
Red pixel: 255 0 0
After addition, defaultP: 255 100 100
```

我们看到 defaultP 的 red 成员的值已经超过了 255 的最大值。如果一个 Pixel 对象的任何一个数据成员超过最大值，那么在使用 Pixel 类修改图像时都会出现不可预料的结果。为了处理这种情况，我们仿照前面小节中讨论过的 istream 类的模型。Pixel 类将维护一个最大值的溢出标志，当检测到溢出时，将对溢出标志中的相应位进行设置。使用 Pixel 类的应用程序应当负责检查溢出标志的状态。类中将提供公用的方法来访问溢出标志，同时提供一个私有的验证方法来维护标志的状态。Pixel 类中所有的修改方法都将调用验证方法。扩展后的 Pixel 类声明如下：

```
/*-------------------------------------------------------------------*/
/* Pixel class declaration.                                          */
/* File name: pixel.h                                                */
/* This class implements the concept of a pixel                      */
#ifndef PIXEL_H
#define PIXEL_H
#include <iostream> //Required for ostream, istream
using namespace std;
class Pixel
{
  public:
  static const unsigned int MAXVAL = 255;

  //Constructors
  Pixel(); //Default
  Pixel(unsigned ); //Gray scale
  Pixel(unsigned,unsigned,unsigned); //Full color range

  //Overloaded operators:
  Pixel operator+(const Pixel& p) const;
  Pixel operator*(double v) const;
  Pixel operator/(unsigned v) const;

  //IO Operators:
  friend istream& operator >>(istream& in, Pixel& p);
  friend ostream& operator <<(ostream& out, const Pixel& p);

  bool overflow() const; //check overflow state
  void reset(); //reset overflow state
```

```
private:
unsigned int red, green, blue;
unsigned short overflowFlag;

static const unsigned short RMASK = 4;
static const unsigned short GMASK = 2;
static const unsigned short BMASK = 1;
static const unsigned short CHECK = 7;

void validate(); //set overflow bits
};
```

注意我们向Pixel类中添加了5个静态常量。公共常量MAXVAL被设置为255，它是公共接口的一部分。其他4个私有常量被多个类方法引用，分别被赋值为1、2、4和7。回忆二进制里1，以8位表示为00000001；二进制的2表示为00000010；二进制的4表示为00000100；二进制的7表示为00000111。这些常量被用作位掩码（bit mask），用来设置和检查overflowFlag。位掩码是一个整数，通过使用按位操作符来操作每一位。扩展后的Pixel类的实现如下：

```
#include "Pixel.h"
/*-------------------------------------------------------------*/
/* Pixel class implementation.                                 */
/* File name: Pixel.cpp                                        */
/* Addition (+) operator.                                      */
Pixel Pixel::operator+(const Pixel& p) const
{
  Pixel temp;
  temp.red = red + p.red;
  temp.green = green + p.green;
  temp.blue = blue + p.blue;
  temp.validate();
  return temp;
}
/*-------------------------------------------------------------*/
/* Multiplication (*) operator.                                */
Pixel Pixel::operator*(double v) const
{
  Pixel temp;
  temp.red = red*v;
  temp.green = green*v;
  temp.blue = blue*v;
  temp.validate();
  return temp;
}
/*-------------------------------------------------------------*/
/* Division (/) operator.                                      */
Pixel Pixel::operator/(unsigned int v) const
{
  Pixel temp;
  temp.red = red/v;
  temp.green = green/v;
  temp.blue = blue/v;
  temp.validate();
  return temp;
}

/*-------------------------------------------------------------*/
ostream& operator<<(ostream& out, const Pixel& p)
```

```
}
  out << p.red << ' ';
  out << p.green << ' ';

  out << p.blue;
  return out;
}
//-----------------------------------------------------------------*/

istream& operator>>(istream& in, Pixel& p)
{
  in >> p.red >> p.green >> p.blue;
  p.validate();
  return in;
}
//-----------------------------------------------------------------*/
Pixel::Pixel()
{
  //Black
  red=green=blue=0;
  return;
}
//-----------------------------------------------------------------*/
Pixel::Pixel(unsigned int value)
{
  //Grey scale
  red=green=blue=value;
  validate();
  return;
}
//-----------------------------------------------------------------*/
Pixel::Pixel(unsigned int r,unsigned int g,unsigned int b)
{
  //Full color range
  red=r;
  green=g;
  blue=b;
  validate();
  return;
}
//-----------------------------------------------------------------*/
bool Pixel::overflow() const
{
  return(overflowFlag&CHECK);
}
//-----------------------------------------------------------------*/
void Pixel::reset()
{
  if(red > MAXVAL) red = MAXVAL;
  if(green > MAXVAL) green = MAXVAL;
  if(blue > MAXVAL) blue = MAXVAL;
  overflowFlag = 0;
}
//-----------------------------------------------------------------*/
void Pixel::validate()
{
  if(red > MAXVAL) overflowFlag = overflowFlag|RMASK;
  if(green > MAXVAL) overflowFlag = overflowFlag|GMASK;
  if(blue > MAXVAL) overflowFlag = overflowFlag|BMASK;
}
```

注意私有的修改方法 void Pixel::validate() 使用了按位操作符“|”来设置 overflowFlag 的状态。方法 validate() 必须被 Pixel 类的所有修改方法调用，用于保证 overflowFlag 的正确状态。公共访问方法 bool Pixel::overflow() 使用按位操作符“&”返回 overflowFlag 的状态。我们将看看这两个方法的细节，并讨论按位操作符。

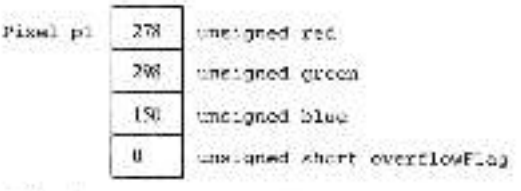
10.2.6 按位操作符

按位操作符对操作数的每位进行操作。C++ 支持 3 种二元按位操作符：按位或（|），按位与（&）和按位异或（^），还支持一元按位非操作符（~）。这些操作符的操作数被限定为整型类型，如 int 或 char。表 10.2 给出了按位操作符的真值表。

表 10.2 C++ 按位操作符的真值表

A	B	~A	A|B	A^B	A&B
0001	0001	1110	0001	0000	0001
0010	0010	1101	0010	0000	0010
0011	0100	1100	0111	0111	0000
0100	0111	1011	0111	0011	0100

为了说明按位操作符的用法，假定某个 Pixel 对象已经定义，并具有如下的值：



语句“p1.validate();”会形成下面的程序跟踪和内存快照：

Validate()	内存快照
步骤 1：if(red>MAXVAL) overflowFlag = overflowFlag |RMASK;	Pixel p1 278 unsigned red 298 unsigned green 150 unsigned blue 4 unsigned short overflowFlag
步骤 2：if(green>MAXVAL) overflowFlag = overflowFlag|GMASK;	Pixel p1 278 unsigned red 298 unsigned green 150 unsigned blue 6 unsigned short overflowFlag
步骤 3：if(blue>MAXVAL) overflowFlag = overflowFlag|BMASK;	

当步骤 1 中按位或的操作数为 overflowFlag（00000000）和 RMASK（00000100）时，结果为 4，如下所示：

```
00000000   overflowFlag
00000100   RMASK

00000100   按位或的结果
```

操作的结果被赋给 overflowFlag，替代了前一个值。当步骤 2 中按位或的操作数为 overflowFlag（00000100）和 RMASK（00000010）时，结果为 6，如下所示：

```
00000100  overflowFlag
00000010  CMASK

00000110  按位或的结果
```

这里 overflowFlag 的值表明在数据成员 red 和 green 中出现了溢出，在数据成员 blue 中没有溢出。因此，语句 p1.overflow() 将返回一个布尔值——真，如下所示：

```
00000110  overflowFlag
00000111  CHECK

00000110  按位与的结果
```

我们现在将编写一个小的驱动程序来测试 Pixel 类的新方法：

```
/*-------------------------------------------------------------*/
/* Program Chapter10_3                                         */
/* Driver program to test overflow state of Pixel              */

#include "Pixel.h"
#include <iostream>
using namespace std;
int main()
{
  //Declare objects
  Pixel defaultP;
  Pixel grayP(100);
  Pixel redP(255,255,0);

  //Create overflow in red
  defaultP = grayP + redP;
  cout << defaultP << endl;
  if(defaultP.overflow() )
  {
     defaultP.reset();
  }
  cout << defaultP << endl;

  return 0;
}
```

一次示例运行的输出如下所示：

```
Ingbers-MacBook-Pro:Programs jaingber$ g++ main.cpp Pixel.cpp
Ingbers-MacBook-Pro:Programs jaingber$ ./a.out
355 100 100
255 100 100
```

我们看到表达式 defaultP.overflow() 返回值为真，说明检测到了溢出，所以语句 "defaultP.reset();" 被执行，以将 defaultP 的值重置为合法状态。

修改

1. 将下面的方法添加到 Pixel 类中：

```
bool checkRed() const;
bool checkGreen() const;
bool checkBlue() const;
```

每个方法都应当检查调用对象的 overflowFlag 标志，如果在特定的数据成员中发生了溢出则返回 true，否则返回 false。

2. 将下面的方法添加到 Pixel 类中：

```
void resetRed();
void resetGreen();
void resetBlue();
```

每个方法都应当将调用对象的特定数据成员置为 MAXVAL，并且重置 overflowFlag 中的相应位。

3. 修改输出函数

```
friend ostream& operator<<(ostream& out, const pixel& p)
```

使输出的 pixel 值用冒号而不是空白分隔 red、green 和 blue 的值。

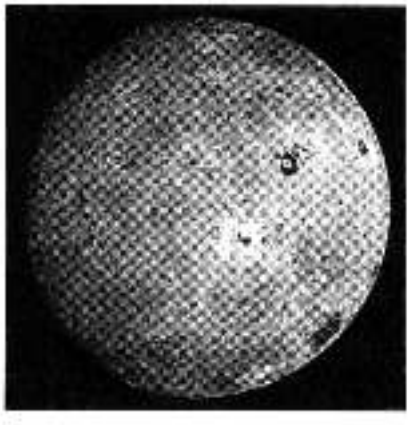
4. 修改输入函数

```
/*-------------------------------------------------------------*/
/* Input >> operator. Expected pixel format is 100:150:100*/

friend istream& operator>>(istream& in, pixel& p)
```

通过警告用户来处理不正确的格式，并要求重新输入。（提示：参考第 5 章讨论的错误处理。）

10.3 解决应用问题：彩色图像处理

下面是一幅由伽利略飞船拍摄的木星卫星——木卫一的图像：



这幅图像是从 nasa.gov 主页上搜索木卫一图像时下载的。

使用 unix 工具 xv 将该图像转换成 ASCII 文件格式。下面是来自 ASCII 文件中代表像素的抽样，称为位图（bitmap）：

```
 49  58 100 147 150 204 148 160 208 147 159 207 146 158 210
149 161 213 146 159 212 145 158 211 144 158 210 143 158 213
142 159 211 143 159 211 142 160 210 142 160 210 142 160 208
142 161 208 136 156 208 135 158 208 136 156 207 138 155 207
139 153 207 140 153 206 142 151 206 142 151 206 140 155 210
149 156 211 148 157 212 148 159 213 147 156 212 142 159 211
142 158 210 141 156 211 142 156 212 142 152 213 141 151 212
140 150 211 139 149 210 138 149 210 140 150 211 140 150 211
142 152 213 142 152 213 143 153 214 144 154 215 144 154 215
143 153 214 142 152 213 142 152 211 145 154 211 144 153 208
```

```
143 152 207 143 152 207 144 155 208 143 154 209 147 156 211
148 157 212 146 155 210 147 156 211 148 157 212 148 157 212
149 158 213 150 159 214 151 160 215 151 161 214 152 162 215
152 162 213 152 162 213 152 162 213 152 162 213 152 162 213
```

位图中第一个像素由三元组 43 56 100 表示。位图中的每行有 5 个像素值。完整的图像存储使用 ppm 文件格式（ppm file format）存储为一个 ASCII 文件。一个 ppm 文件必须包含头部，用于识别或转换图像。头部中包含一个“魔数”（对于 ppm 文件为 P3）、可能的注释（注释行第一列以 # 开头）、图像的宽度和高度以及最大的色彩值。该图像的头部为

```
P3
# CREATOR: XV Version 3.10a  Rev: 12/29/94
256 256
255
```

一旦图像被修改，可以使用 xv 工具查看 ASCII 文件，或者将它转换回一个可在其他平台查看的 jpg 或 gif 文件。“被平滑过的”木星一图像如下所示：



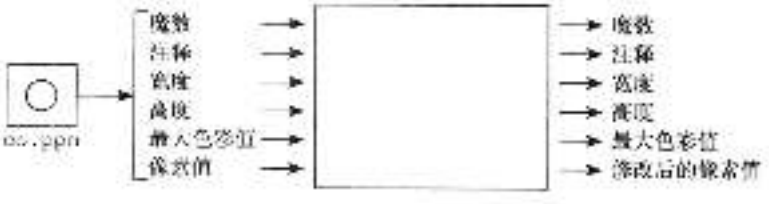
编写一个程序从一个 ASCII ppm 文件中输入一幅图像，对这幅图像进行平滑操作。将被平滑过的图像写入一个新的 ppm 文件中。

1. 问题描述

修改一幅彩色数字图像，对其进行平滑化。

2. 输入 / 输出描述

程序的输入是图像文件 lo.ppm 中的数据，输出是修改过的图像。我们必须首先读取头部信息，以确定图像的大小。然后使用头部中的信息分配内存并输入像素值。



3. 用例

为了完成对图像的平滑化处理，我们将对当前像素和其 4 个邻接像素取平均值；4 个邻接像素分别是上、下、左、右的像素。我们将把原始的像素值用计算出的平滑过的像素值来替换。对于我们的用例，我们将确定木星一图像中某个像素被平滑后的值。

原始图像：

```
143 159 211 142 160 210 142 160 210
136 158 208 138 158 208 136 156 207
140 153 206 142 151 206 142 151 206
```

当前像素的 red 值为 135，green 值为 158，blue 值为 208。该像素的平滑值计算如下：

```
red value -> (135 + 136 + 142 +136 +142)/5 = 138
green value -> (158 +158 + 160 + 156 +153)/5 = 156
blue value -> (208 + 208 + 210 + 207 + 206)/5 = 207
```

修改后的图像：

```
143 159 211 142  60 210 142 160 210
136 158 208 138 156 207 136 156 207
140 153 206 142 151 206 142 151 206
```

对图像内部的每个像素重复该过程。边界上的像素由于缺失了 4 个邻接像素中的 1 个（角上的像素缺失 2 个），因此在应用中将不会被修改。

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。

分解提纲

1）读取头部信息；

2）将头部信息写入新文件中；

3）读取像素值；

4）对每个内部像素进行平滑化；

5）将平滑过的像素值写入新文件中。

步骤 1 和 2 涉及从数据文件中读取头部信息并通过将信息写入新文件进行保存。我们将编写一个函数来完成这项任务。步骤 3 要求将像素值读入一个二维数组。数组的大小取决于头部中的信息，所以我们将使用 vector 类来定义一个 Pixel 类型的二维数组。步骤 4 设计为对图像的平滑化进行修改。我们将再编写一个函数来完成该项任务。为了简化图像中像素元素的存储和修改，我们将使用在 10.2 节中开发的 Pixel 类。细化后的伪代码如下所示：

细化的伪代码

```
main():
    open data files
    read_header(fin, fout, width, height, max_color)
    read image
    smooth(image,width, height)
    write image
read_header(fin, fout, width, height, max_color):
        read magic_num
        while(next_ch == '#')
            read comment_line
            write comment_line to new file
        read width, height, max_color
        write width, height, max_color
smooth(image, width, height):
    set i to 0
    set j to 0
while(i<height)
  while(j<width)
    image[i][j] = (image[i][j] + image[i][j-1] + image[i-1][j]
                  + image[i][j+1] + image[i+1][j])/5
```

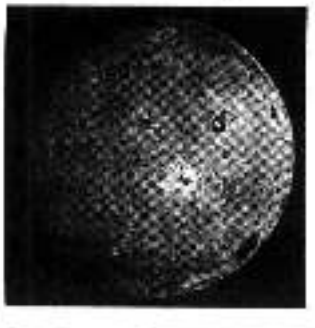
现在我们将伪代码转换成 C++ 代码：

```
/*                                                                    */
/* Program chapter16_4                                                */
/*                                                                    */
/* This program reads an ASCII digital image and performs a           */
/* smoothing operation on the on the image. The smoothed image        */
/* is written to a new file.                                          */

#include "Pixel.h"
#include <fstream>//Required for ifstream, ofstream
#include <string>//Required for string
#include <vector>//Required for vector
using namespace std;

//Function prototypes.
void read_header(ifstream& fin, ofstream& fout,
                 int& width, int& height, int& max);
void smooth(vector <vector<Pixel> >& , int w, int h);

int main()
{
   // Declare objects.
   int height, width, max, i, j;
   string filename;
   ifstream fin;
   ofstream fout;

   // Prompt user for file name and open file for input.
   cout << "enter name of input file ";
   cin >> filename;
   fin.open(filename.c_str());
   if(fin.fail())
   {
      cerr << "Error opening input file\n";
   }
   else
   {
   // Open new file for output.
      filename = "smoothed "+filename;
      fout.open(filename.c_str());

   // Read and write header information.
      read_header(fin,fout, width, height, max);
   // Declare image array.
      vector< vector<Pixel> > image(height, width);

   // Read the image.
      for(i=0; i<height; i++)
       for(j=0; j<width; j++)
       {
         fin >> image[i][j];
       }
   // Smooth the image.
       smooth(image, width, height);

   // Write modified image to new file.
      for(i=0; i<height; i++)
        for(j=0; j<width; j++)
        {
           fout << image[i][j] << ' ';
```

```
            if((i+1)%5 == 0) fout<<endl;
        }
    }

    // Exit program.
    return 0;
}
void read_header(istream& fin, ostream& fout, int& width,
                 int& height, int& max)
{
  char header[100];
  char ch;
// Get magic number.
  fin.getline(header, 100);
// Write magic number.
  fout << header << endl;
  cout << header << endl;

// Get all comment lines and write to new file.
   fin >> ch;
   while(ch == '#')
   {
    fin.getline(header, 100);
    fout <<ch<< header << endl;
    cout << ch <<header << endl;
    fin >>ch;
   }
   fin.putback(ch);
// Input width and height of image.
   fin >> width >> height;
   cout << width <<" " << height << endl;
   fout << width << " " << height << endl;
// Input maximum color value.
  fin >> max;
  cout << max << endl;
  fout << max << endl;
  return;
}
void smooth(vector< vector<Pixel> > &image, int w, int h)
{
   for(int i=1; i<w-1; i++)
    for(int j=1; j<w-1; j++)
      image[i][j] = (image[i][j] + image[i+1][j] + image[i-1][j]
                     + image[i][j-1] + image[i][j+1])/5;
}
```

5. 测试

下面是平滑后的图像及原图像，后者用于说明效果：





修改

1. 修改程序 chapter10_4 中的 smooth 函数，使其包含 8 个邻接像素，而不再是 4 个。
2. 修改程序 chapter10_4 中的 smooth 函数，使其包含对边界、角上像素的平滑运算。注意不要包含图像边界外的像素引用。

10.4 递归

递归是解决某类特定类型问题的有力工具。这一类型问题的解决方案可以被定义成一个与之相似但更小的问题，这个更小的问题仍可以定义成一个与之相似但更小的问题。一直重复这种将问题定义成更小问题的过程，直到某个更小的问题有一个确定解为止，这样就可以将该确定解用于确定整个解决方案。

支持递归的程序语言允许函数调用自身。调用自身的函数称为递归函数（recursive function）。为了以递归方式解决问题，函数可以按照这种方式编写：函数每次调用自身，这样将问题简化成一个类似但更小的问题。函数将一直调用它自身，直到最终返回问题小部分的唯一解为止。唯一解被传回调用链，从而确定全部解决方案。

每次一个递归函数调用自身时，当前函数的实例信息就被存储在一块被称作栈（stack）的内存单元中。回忆第 9 章的内容，栈是一种先进后出的数据结构。每次连续地对递归函数调用都将信息压入栈中。每个连续的 return 将信息从栈中弹出。由于栈的大小有限，递归函数可调用自身的次数是一个与系统相关的限制值，但这些限制通常不会带来问题，如果你的递归函数编写正确的话。

一个递归函数需要两个部分：

1）一部分定义终止条件或返回点，这里将返回问题的一个更小版本的唯一解。

2）一个递归部分，该部分将问题缩减成与原来相似但更小版本的问题。

为了说明递归，我们将从开发一个用于计算阶乘函数的递归算法开始。

10.4.1 阶乘函数

回忆 $n!$（读作 n 的阶乘）的定义如下：

$$n! = (n)(n-1)(n-2)\cdots(1)$$

这里 n 是一个非负整数，且定义 0!=1。下面给出了该问题的递归定义：

$$f(n) = n! = \begin{cases} 1 & n=0 \\ n * f(n-1) & n>1 \end{cases}$$

本例的递归定义中给出了终止条件（$f(0)=1$），以及递归部分的约简语句（$f(n)=n*f(n-1)$）。约简语句将求 $f(n)$ 的问题简化成计算 $f(n-1)$ 与 n 的乘积问题。

为了说明，我们将手工计算 5!，如下所示：

$$5! = 5 * 4!$$
$$4! = 4 * 3!$$
$$3! = 3 * 2!$$
$$2! = 2 * 1!$$
$$1! = 1 * 0!$$
$$0! = 1$$

因此，我们已经将一个阶乘定义成了关于更小的阶乘的乘积。更小的阶乘可以一直被重

定义，直到 0! 为止。在最后一个等式中，我们将唯一解使用 0! 替换，然后开始回溯等式列表，将阶乘替换成值：

$$1! = 1 * 1$$
$$2! = 2 * 1! = 2$$
$$3! = 3 * 2! = 6$$
$$4! = 4 * 3! = 24$$
$$5! = 5 * 4! = 120$$

现在我们就得到了一个计算阶乘的递归算法。

我们给出一个使用递归函数计算阶乘的程序。因为阶乘值增长很快，所以我们使用长整数来存储阶乘值。注意在 main() 函数中的函数调用看起来与调用一个非递归函数是一样的。

```
/*-------------------------------------------------------------*/
/*  Program Chapter10_9                                        */
/*                                                             */
/*  This program calls a recursive function to                 */
/*  compute a factorial.                                       */

#include <iostream> //Required for cout
using namespace std;

// Function prototypes.
long factorialR(int n);

int main()
{
   //  Declare objects
   int n;

   //  Get user input.
   cout << "Enter positive integer: \n";
   cin >> n;

   //  Compute and print factorials.
   cout << "Recursive: " << n << "! = " << factorialR(n) << endl;

   //  Exit program.
   return 0;
}
/*-------------------------------------------------------------*/
/*  This function computes a factorial recursively.            */

long factorialR(int n)
{
   /*  Recursive reference until n is equal to 0.             */
   if (n == 0)  //Solution is known
   {
      return 1;  //Return unique solution.
   }
   return n*factorialR(n - 1);  //Reduce the  problem
}
/*-------------------------------------------------------------*/
```

终止条件 n == 0 使递归示例不会成为无限循环。递归部分调用自身时，参数每次递减 1，直到调用参数为 0 为止。

对于较大的 n 值，$n!$ 的值甚至会超出 long 的范围。在这种情况下，计算式应当使用 double 类型。在本章结尾我们将讨论一种有趣的近似阶乘计算方法。

图 10.3 给出了程序 chapter10_5 的程序跟踪。

main()	内存快照
步骤 1: int n; 步骤 2: cin >>n; 步骤 3: cout << ... << factorialR(n);	int n ? int n 4
factorialR(n)	
步骤 4: return n*factorialR(3)	int n 4 stack 4*factorialR(3)
factorialR(3)	
步骤 5: return 3*factorial_R(2)	int n 3 stack 3*factorialR(2)
factorialR(2)	
步骤 6: return 2*factorialR(1)	int n 2 stack 2*factorialR(1)
factorialR(1)	
步骤 7: return n*factorialR(0)	int n 1 stack 1*factorialR(0)
factorialR(0)	
步骤 8: return 1	factorialR(0) returns 1 factorialR(1) returns 1*1 factorialR(2) returns 2*1 factorialR(3) returns 3*2 factorialR(4) returns 4*6

图 10.3 程序跟踪

10.4.2 斐波纳契序列

斐波纳契序列是一个由 f0，f1，f2，f3，…组成的数字序列，序列中的前两个值（f0 和 f1）都等于 1，随后的每个数都是前面两个数的和。因此，斐波纳契序列前面的若干值为

1 1 2 3 5 8 13 21 34…

该序列在 1202 年首次提出，它的应用从生物学到电子工程都有涉及，例如，斐波纳契序列常用于计算野兔的种群增长。

计算斐波纳契序列的第 k 个值的函数是递归函数的一个好候选者，因为序列的每个新值都计算自前面两个值，如下面的递归定义所示：

$$f(n)=\begin{cases}1 & n=0,\ n=1\\ f(n-1)+f(n-2) & n>1\end{cases}$$

下面我们给出计算斐波纳契序列中第 k 个值的一个递归函数。

```
/* ------------------------------------------------------------ */
/*                                                              */
/*  This function computes the kth Fibonacci                    */
/*  number using a recursive algorithm.                         */

int fibonacciR(int k)
{
   // Declare objects.
   int term = 1;
```

```
    //  Compute k-th Fibonacci number recursively
    if (k <=1)
    {
      return term;  //Unique solution.
    }
    return  fibonacci8(k-1) + fibonacci8(k-2);  //Reduce.
}
/*------------------------------------------------------------------*/
```

在递归函数中，条件 $k \leq 1$ 使函数不会陷入无限循环。

修改

1. 使用程序 chapter10_5 计算 1!，2! 等阶乘的值，直到到达长整数的上限为止。在你的系统上当 k! 的值超过上限时会出现什么错误消息？
2. 修改程序 chapter10_5，使其用 double 类型而不是整数来计算阶乘值。解释为什么在使用 double 值计算 k! 时，精度的位数决定了可以被正确计算的阶乘最大值。在你的系统上，使用 double 值可计算出的 k! 的最大值为多少？
3. 编写一个 main() 函数测试斐波纳契函数。在你的系统上，使用整数可以正确计算出的最大斐波纳契值是多少？

10.4.3 BinaryTree 类

本节我们将实现一个新的自定义类型，用来表示二叉树。我们的实现中将包含类组合的使用，以及若干递归方法的定义。在 10.5 节中，我们将使用二叉树类来开发一个类模板。

二叉树是一种动态的链式数据结构，代表了从单一根（root）扩展出来的节点集合。二叉树的根是第一个节点。二叉树上的每个节点（node）都由一个数据值和至多两个指向其他节点的链接组成。指向其他节点的链接称作左孩子（left child）和右孩子（right child）。左孩子是左子树的根，右孩子是右子树的根。如果一个节点的左孩子和右孩子都为空，那么这个节点称作叶子节点（leaf node）。图 10.4 中给出了一个带有 5 个节点的二叉树示意图。

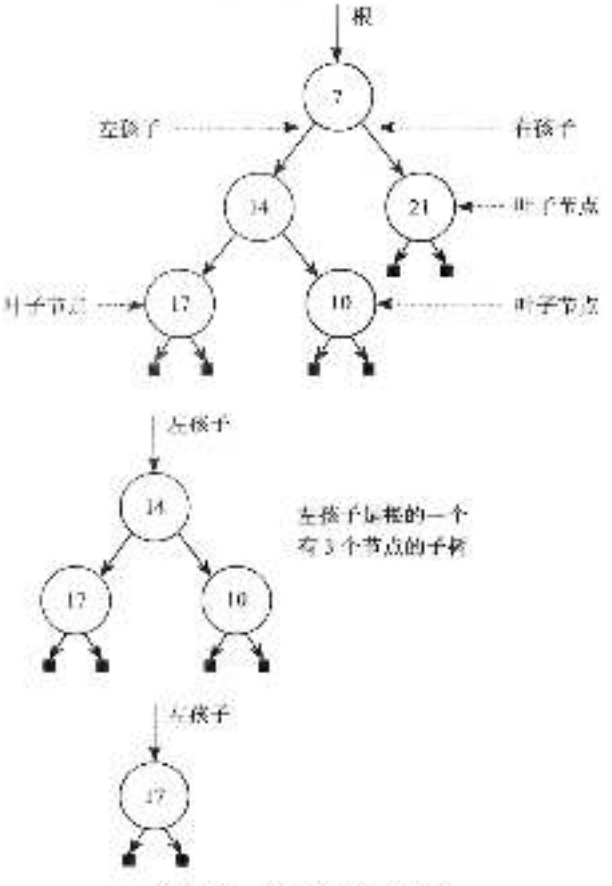


图 10.4 5 个节点的二叉树

图 10.4 中，树的根是第一个节点，值为 7。第一个节点有一个左孩子，是一个值为 14 的节点，右孩子是一个值为 21 的节点。值为 14 的节点有一个左孩子，后者是一个叶子节点；右孩子也是一个叶子节点。注意每个节点的左孩子和右孩子都是一个更小的子树的根。

当设计一个新的类型时，我们需要考虑类型的物理描述和类型的公共接口所支持的操作。二叉树的物理描述就是指，这里的根是指向一个节点的指针。二叉树的基本操作包括插入节点，打印节点值和清空树（删除所有节点）。图 10.5 给出了一个最小化二叉树的类型。

注意在图 10.5 中使用了名为 Node 的数据类型和重载方法 print()、insert() 和 clear()。二叉树的结构本身就是递归的，因为二叉树上的每个节点都是第一个节点的一棵更小的子树。因此，二叉树的基本操作的实现使用了递归。私有方法都是递归方法，它们被非递归的公共方法调用。我们将在设计一个实现节点的新类型之后来讨论这里的更多细节。

Node 类。因为没有内建类型用于表示一个节点，我们将定义一个新的类型来实现节点。二叉树上的节点是一个定义为有数据值、左孩子和右孩子的类型。这三个属性都是可修改的，所以我们将提供包含 3 个存取方法和 3 个修改方法的公共接口。图 10.6 中给出了 Node 类的类图。

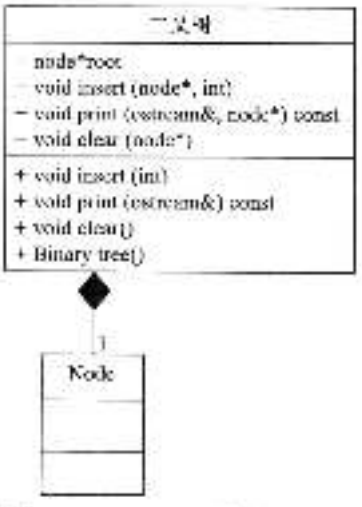


图 10.5 BinaryTree 类的 UML 图

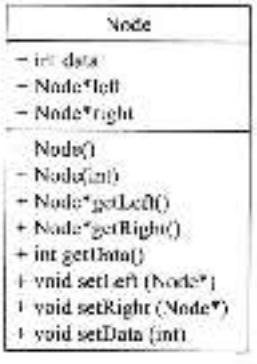


图 10.6 Node 类的 UML 图

图 10.6 中的类图为 Node 的数据值指定了整数类型。注意，对二叉树进行的操作独立于每个节点所存储的数据值的类型，因此，Node 是类模板的一个好候选者，这里数据值的类型是参数化类型。在实现和测试了包含特定数据类型的 Node 类之后，我们将设计一个用于一般节点类型的类模板。

下面给出了图 10.6 中描述的 Node 类的实现。

注意预处理指令 #ifndef、#define 和 #endif 的使用，这些指令应当用于所有的头文件中，以免头文件被多次包含。例如，如果在编译时头文件 Node.h 被一个以上文件包含，编译器将给出类似下面的错误消息。

```
error: redefinition of 'class Node'
```

产生这个错误消息是因为编译器尝试多次定义 Node 类。预处理指令 #ifndef 读作“如果未定义”，它后面跟一个标识符，该标识符一般是全部大写并引用头文件的名字。头文件第一次被包含时，#ifndef 返回 true，因为此时大写的标识符还没有被定义。因为 #ifndef 返回 true，所以在 #ifndef 和 #endif 之间的内容都会被处理，其中包括预处理指令 #define NODE_H，该指令定义了 NODE_H。在头文件下一次被包含时，指令 #ifndef NODE_H 返

回 false；因此在 #ifndef 和 #endif 之间的内容都会被忽略，这样就避免了 Node 类被多次包含。

```
/*                                                              */
/* Class declaration for node:                                  */
/* Filename node.h                                              */

#ifndef NODE_H
#define NODE_H
class Node
{
   //Private attributes
   private:
        int data;
        Node *left;
        Node *right;

   //Public interface.
   public:
        Node();  //Default constructor
        Node(int); //Parameterized constructor

        //Accessors
        Node* getLeft() const;
        Node* getRight() const;
        int getData() const;

        //Mutators
        void setLeft(Node*);
        void setRight(Node*);
        void setData(int v);

};
#endif
/*--------------------------------------------------------------*/
/*--------------------------------------------------------------*/
/* Class implementation for node                                */
/* Filename: node.cpp                                           */

#include "node.h" //Required for Node.
using namespace std;

//Constructors.
Node::Node() : data(0),left(0),right(0)
{
}
Node::Node(int v):data(v), left(0), right(0)
{
}
//Accessors
Node* Node::getLeft() const
{
  return left;
}
Node* Node::getRight() const

  return right;
}
int Node::getData() const
{
```

```
  return data;
}

//Mutators
void Node::setLeft(Node* l)
{
  left=l;
}
void Node::setRight(Node* r)
{
  right=r;
}
void Node::setData(int v)
{
  data=v;
}
//-----------------------------------------------------------/
```

为了测试 Node 类，我们将编写一个驱动程序，该程序至少一次调用所有的方法，并打印出每次方法调用后属性的状态。

```
//-----------------------------------------------------------/
/* Program chapter13_6                                       */
/* This program tests the node class                         */
/* Filename: nodeTest.cpp                                    */

#include<iostream> //Required for cout.
#include "node.h" //Required for Node.
using namespace std;

int main()
{
  Node n1,n3; //Test default constructor.
  Node n2(4); //Test parameterized constructor.

  //Test accessor methods.
  cout << "Value of n1 after default construction: "
       << endl << n1.getData() << "," << n1.getLeft() << ","
       << n1.getRight() << endl;
  cout << "Value of n2 after parameterized construction: "
       << endl << n2.getData() << "," << n2.getLeft() << ","
       << n2.getRight() << endl;

  //Test mutator methods.
  n1.setData(3);
  n1.setLeft(&n2);
  n1.setRight(&n3);
  cout << "Value of n1 after modification: " << endl
       << n1.getData() << "," << n1.getLeft() << ","
       << n1.getRight() << endl;
  cout << "Value of n2 after modification: " << endl
       << n2.getData() << "," << n2.getLeft() << ","
       << n2.getRight() << endl;
  cout << "Value of n3 after modification: " << endl
       << n3.getData() << "," << n3.getLeft() << ","
       << n3.getRight() << endl;
  return 0;
}
//-----------------------------------------------------------/
```

测试程序某次运行生成的输出如下：

```
Value of n1 after default construction:
0,0,0
Value of n2 after parameterized construction:
1,0,0
Value of n1 after modification:
13,0xbffffa[illegible],0xbffffae4
Value of n2 after modification:
4,0,0
Value of n1 after modification:
0,0,0
```

练习

使用本节设计的 Node 类回答下面的问题。考虑下面给出的程序：

```
#include<iostream>
#include "node.h"
using namespace std;

int main()
{
  Node n1, n2(1);
  cout << n1.getRight() << endl; //Line 1
  cout << n2.getLeft() << endl; //Line 2
  cout << n2.getData() << endl; //Line 3
  n1.setLeft(&n2);
  cout << n1.getLeft() << ',' << n1.getRight()
          << ',' << n1.getData() << endl; //Line 4
  return 0;
}
```

1. 给出行 1 的输出。
2. 给出行 2 的输出。
3. 给出行 3 的输出。
4. 给出行 4 的输出。

递归成员函数，为了继续如图 10.5 所描述的来开发 BinaryTree 类，我们必须考虑如何安排树上的节点，以及如何遍历二叉树。二叉树可能是有序的，也可能是无序的。无序二叉树假定节点上的数据值是无序的。有序二叉树假定节点上的数据值是有序的。

二叉搜索树（binary search tree）是一个有序二叉树的例子。二叉搜索树是这样的二叉树：

1）每个节点都有一个值；

2）节点的左子树所包含的值小于节点的值；

3）节点的右子树所包含的值大于等于节点的值。

图 10.7 中展示了一棵 6 个节点的二叉搜索树。

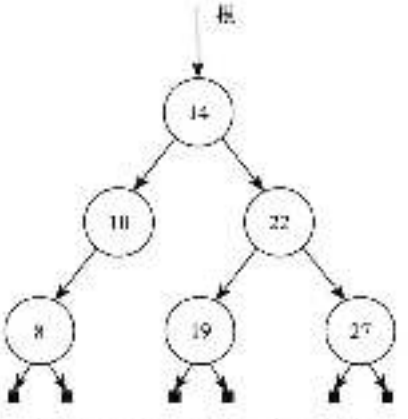


图 10.7 6 个节点的二叉搜索树

二叉搜索树是用于存储大量有序数据时的常用数据结构，它进行遍历和修改的速度都很快。注意图 10.7 中的示意图，从树的根到其他每个节点的路径都是唯一的。还可以看到，每个节点的左孩子和右孩子都是根的一棵子树。这两个特征允许在树遍历（tree traverse）时使用递归算法。

遍历树就是以系统的方式访问树上每个节点的过程，遍历算法可以按照节点被访问的顺序来分类。例如，如果我们希望 print() 方法打印出图 10.7 中存储在二叉搜索树中的有序数据，我们将会对树进行中序（in order）遍历，当访问一个节点时打印出它的值。对树进行中序遍历的算法如下所示：

```
inOrder traverse

if(root == null)
  return
inOrder traverse left child
visit node
inOrder traverse right child
```

算法从二叉树的根开始，然后在随后每个节点的左孩子上递归调用自身，直到遇到空值为止。当最后找到一个左孩子为空的节点时，该节点就被访问了，算法随后在该节点的右孩子递归调用自身。条件 root == null 是终止条件，每次递归调用都将问题简化成对子树的遍历。对图 10.7 中的树进行中序遍历，打印出访问的每个节点，得到的输出结果将是：8 10 14 19 22 27。

将一个新的节点插入到正确的位置，同时需要保持二叉搜索树的顺序，完成该过程的算法也是从树的根开始。如果树为空，那么新节点将作为树的第一个节点。如果树非空，新节点的值就需要与第一个节点的值比较。如果新节点的值小于根节点，算法就会在左孩子上调用自身，否则就在右孩子上调用自身。该递归算法描述如下：

```
insert( node* root, int value):

  if(root == null)
    root = new node(value)
    return
  else
    if( value < root->value)
      insert(root->left child,value)
    else
      insert(root->right child,value)
```

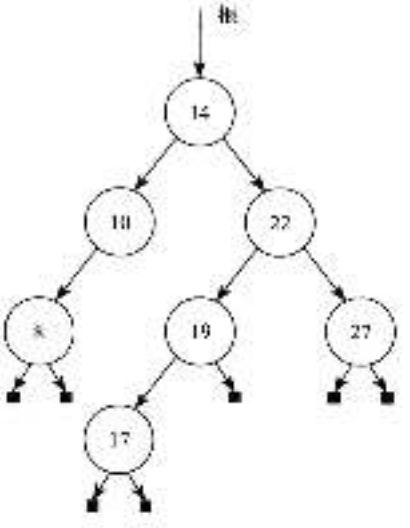


图 10.8 插入新节点后的二叉搜索树

如果值 17 被插入图 10.7 的二叉搜索树中，得到的结果如图 10.8 所示。

清空一棵树的算法描述如下。

```
clear(node* root):

if(root == null)
  return
else
  clear(root->left)
  clear(root->right)
  delete root
```

现在我们可以像图 10.5 所描述的那样完成 BinaryTree 类的定义。

```
/*-------------------------------------------------------+/
/* Binary tree declaration                               +/
/* filename: binaryTree.h                                +/

#include "node.h" //Required for node.
#include <ostream> //Required for ostream.
using namespace std;

class BinaryTree {
public:
  //Default constructor.
  BinaryTree():
```

```
  //public, non recursive print and insert
  void print(ostream& out) const;
  void insert(int value);
  void clear();
private:
  //private recursive overloaded print and insert
  void print(ostream& out, node* rt) const;
  void insert(node* rt, int value);
  void clear(node* rt);

  //private attribute
  node* root;
};
/*-------------------------------------------------------*/
```

注意在 BinaryTree 类的定义中，方法 print()、insert() 和 clear() 被重载了。这些方法的 public 版本是非递归的。这些方法的 private 版本是递归的，每个方法都有一个 Node* 类型的形参。

这些方法的 public 版本必须被一个 BinaryTree 对象调用，因此树的根由调用对象提供。这些方法的 private 版本在调用对象的左子树和右子树上递归调用自身，按此遍历树。因此形参 rt 的值对于每个随后的调用都不一样，必须作为参数进行传递，以支持这些方法的递归性质。一个最小化的 BinaryTree 类的定义如下：

```
/*-------------------------------------------------------*/
/* binary tree implementation                            */
/* filename: binaryTree.cpp                              */
/*-------------------------------------------------------*/
#include "bTree.h" //Required for BinaryTree.
#include <iostream> //Required for ostream.
using namespace std;

//Constructor
  BinaryTree::BinaryTree():root(0)
  {
  }

//print: Public method
  void BinaryTree::print(ostream& os) const
  {
    if(root == NULL)
    {
     cout << "tree is empty ";
     return;
    }
    else
     print(os, root); //call private print()
  }

//print: private, recursive method
  void BinaryTree::print(ostream& os, node* theRoot) const
  {
   if(theRoot    NULL)
     return;
   else
   {
     print(os, theRoot->getLeft());
     os << theRoot->getData() << " ";
     print(os,theRoot->getRight());
   }
```

```
//insert: public method
void BinaryTree::insert(int value)
{
  if(root == NULL)
    root = new node(value);
  else
    insert(root,value); //call private insert
}
//insert: private method
void BinaryTree::insert(node* root, int val){
  if(val < root->getData())
  {
    //Traverse the left subtree
    if(root->getLeft() == NULL)
    {
      //insert new node here.
      root->setLeft(new node(val));
    }
    else
    {
      //recursive call to traverse left subtree
      insert(root->getLeft(),val);
    }
  }
  else
  {
    //Traverse the right subtree
    if(root->getRight() == NULL)
    {
      //insert new node here
      root->setRight(new node(val));
    }
    else
    {
      //recursive call to traverse right subtree
      insert(root->getRight(),val);
    }
  }
}//end insert
//clear: Public method
void BinaryTree::clear()
{
  if(root == NULL)
    return;
  else
  {
    clear(root); //call private clear()
    root = 0; //tree is empty
  }
}//end clear

//clear: private, recursive method
void BinaryTree::clear(node* theRoot)
{
  if(theRoot == NULL)
    return;
  else
  {
    clear(theRoot->getLeft());
```

```
        clear(theNode->getRight());
        delete theNode;
    }
    //end clear

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

下面的程序测试了 BinaryTree 类。

```
//-------------------------------------------------------------*/
/* Filename: testBTree                                         */

#include<iostream> //Required for cout.
#include "binaryTree.h" //Required for BinaryTree.
using namespace std;

int main()
{
  BinaryTree bt;        //Test default constructor
  bt.insert(2);         //Test insert on empty tree.
  bt.insert(10);        //Test insert to right subtree.
  bt.insert(-2);        //Test insert to left subtree.
  bt.print(cout);       //Test print method.
  bt.clear();           //Test clear method.
  bt.print(cout);
  return 0;
}
```

测试程序一次示例运行的输出如下：

```
-2 2 10
tree is empty
```

既然一个可工作的二叉树版本已经编写完成并经过测试，那么我们将写一个类模板来表示一个一般的二叉树，并将对它的测试版本与工作版本进行比较。

10.5 类模板

类模板的声明必须以下面形式的表达式开头：

```
template < typename 形参名 >
```

下面的类模板声明表示了二叉树中的节点：

```
//-------------------------------------------------------------*/
/* class declaration for node template:                        */
/* filename nodeTanplate.h                                     */

#ifndef TNODE_H
#define TNODE_H
template<typename T>
class Node
{
   //Private attributes
   private:
        T data;
        Node *left;
        Node *right;

   //Public interface.
   public:
        Node(); //Default constructor
```

```
        Node(T); //Parameterized constructor

        //Accessors
        Node* getLeft() const;
        Node* getRight() const;
        T getData() const;

        //Mutators
        void setLeft(Node*);
        void setRight(Node*);
        void setData(T v);
};
#endif
/*-------------------------------------------------------------*/
```

注意标识符 T 替代了类模板声明中的 int。该类模板的实现如下：

```
/*-------------------------------------------------------------*/
/* Class implementation for Node template                      */
/* filename: nodeTemplate.cpp                                  */

#include "nodeTemplate.h" //Required for Node.
using namespace std;

//Constructors:
 template<typename T>
Node<T>::Node() : data(0),left(0),right(0)
{
}
template<typename T>
Node<T>::Node(T v):data(v), left(0), right(0)
{
}
//Accessors
template<typename T>
Node<T>* Node<T>::getLeft() const
{
  return left;
}
template<typename T>
Node<T>* Node<T>::getRight() const
{
  return right;
}
template<typename T>
T Node<T>::getData() const
{
  return data;
}

//Mutators
template<typename T>
void Node<T>::setLeft(Node<T>* l)
{
  left=l;
}
template<typename T>
void Node<T>::setRight(Node<T>* r)
{
  right=r;
}
template<typename T>
void Node<T>::setData(T v)
```

```
  data=v;
}
/*                                                                    */
```

注意类实现中要求每个方法都定义成模板方法。

BinaryTree 的类模板定义如下：

```
/*-------------------------------------------------------------------*/
/* Binary tree declaration                                           */
/* filename: binaryTreeTemplate.h                                    */
#include "nodeTemplate.h" //Required for node.
#include "nodeTemplate.cpp" //Required for node implementation.
#include <iostream> //Required for ostream.
using namespace std;

template<typename T>
class BinaryTree {
public:
  //Default constructor.
  BinaryTree();

  //public, non recursive print and insert
  void print(ostream& out) const;
  void insert(T value);
  void clear();

private:
  //private recursive overloaded print and insert
  void print(ostream& out, Node<T>* rt) const;
  void insert(Node<T>* rt, T value);
  void clear(Node<T>* rt);

  //private attribute
  Node<T>* root;
};
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* Binary tree implementation                                        */
/* filename: binaryTreeTemplate.cpp                                  */

#include "binaryTreeTemplate.h" //Required for BinaryTree.
#include <iostream> //Required for ostream.
using namespace std;

//Constructor
  template<typename T>
  BinaryTree<T>::BinaryTree():root(0)
  {
  }

//clear: Public method
  template<typename T>
  void BinaryTree<T>::clear()
  {
    if(root == NULL)
    {
     return;
    }
    else
    {
```

```
    clear(root); //call private clear()
    root = 0; //tree is empty
  }
}//end clear

//clear: private, recursive method
template<typename T>
void BinaryTree<T>::clear(Node<T>* theRoot)
{
 if(theRoot == NULL)
   return;
 else
 {
   clear(theRoot->getLeft());
   clear(theRoot->getRight());
   delete theRoot;
 }
}//end clear

//print: public method
template<typename T>
void BinaryTree<T>::print(ostream& os) const
{
  if(root == NULL)
  {
   cout << "tree is empty";
   return;
  }
  else
   print(os, root); //call private print()
}//end print

//print: private, recursive method
template<typename T>
void BinaryTree<T>::print(ostream& os, Node<T>* theRoot) const
{
 if(theRoot == NULL)
   return;
 else
 {
   print(os, theRoot->getLeft());
   os << theRoot->getData() << " ";
   print(os,theRoot->getRight());
 }
}//end print

//insert: public method
template<typename T>
void BinaryTree<T>::insert(T value)
{
  if(root == NULL)
    root = new Node<T>(value);
  else
    insert(root,value); //call private insert
}//end insert

//insert: private method
template<typename T>
void BinaryTree<T>::insert(Node<T>* root, T val)
{
  if(val < root->getData())
  {
```

```
    //Traverse the left subtree
    if(root->getLeft() == NULL)
    {
       //insert new node here.
       root->setLeft(new Node<T>(val));
    }
    else
    {
       //recursive call to traverse left subtree
       insert(root->getLeft(),val);
    }
  }
  else
    //Traverse the right subtree
    if(root->getRight() == NULL)
    {
      //insert new node here
      root->setRight(new Node<T>(val));
    }
    else
    {
      //recursive call to traverse right subtree
      insert(root->getRight(),val);
      insert(root->getRight(),val);
    }
  }
}//end insert
/*                                                                    */
```

通过使用自定义的类模板进行泛型编程是 C++ 程序语言支持的一个强大特性，但是所要求的语法有些繁琐。类模板应当总是从一个特定的类实现的工作版本演变而来。

下面的程序测试了我们的类模板：

```
/*                                                                    */
/* Filename: testBTree                                                */

#include<iostream> //Required for cout.
#include "BinaryTreeTemplate.cpp" //Required for BinaryTree.
using namespace std;

int main()
  BinaryTree<int> bt;  //Test default constructor
  bt.insert(2);        //Test insert on empty tree.
  bt.insert(10);       //Test insert to right subtree.
  bt.insert(-2);       //Test insert to left subtree.
  bt.print(cout);      //Test print method.
  bt.clear();          // Test clear method.
  bt.print(cout);
  return 0;
}
```

测试程序的一次示例运行的输出如下：

```
-2 2 10
tree is empty
```

输出结果与特定定义的测试输出匹配。

修改

1. 添加一个名为 printPreOrder（ostream&）的 public 方法和一个名为 printPreOrder（ostream&, Node*）的 private 递归方法，通过它们使用前序遍历的方式打印出一棵树。递归的前序遍历算法如下：

```
preOrder traverse

if(root == null)
  return
visit node
preOrder traverse left child
preOrder traverse right child
```

图 10.8 中树的前序遍历得到的输出为：14 10 8 22 19 17 27。

2. 添加一个名为 printPostOrder（ostream&）的 public 方法和一个名为 printPostOrder（ostream&, Node*）的 private 方法，通过它们使用后序遍历的方式打印出一棵树。递归的后序遍历算法如下：

```
postOrder traverse

if(root == null)
  return
postOrder traverse left child
postOrder traverse right child
visit node
```

图 10.8 中树的后序遍历得到的输出为：8 10 17 19 27 22 14。

10.6 继承

本节我们将介绍 C++ 程序语言支持多态（polymorphism）所使用的继承和 virtual 方法等特性。多态是一种将多个含义赋予同一个名字的能力。这是面向对象编程中的一个重要概念，因为它允许将许多不同类型的对象赋给一个单一类型的对象。因此，一个单一类型名可以有一种以上的含义。

继承提供了一种从已存在的类的类型中定义新类的类型的机制。已存在的类型称作基类型（base type），新类型称作派生类型（derived type）。派生类型派生自基类型，并继承了基类型的属性。一个派生类型的对象与其基类型的对象兼容，因此一个基类型的对象可以被赋予许多不同派生类型对象的值。

除了对多态的支持，继承通过对类的类型属性和方法的继承，使得代码的重用最大化。为了说明 C++ 中继承的实现，我们将开发图 10.9 中所描述的 Rectangle 类和 Square 类。注意 UML 使用开放式箭头来说明继承。继承在箭头的方向建立一种 is-a 关系。重要的是要记住一个正方形是矩形，但一个矩形不一定是正方形。

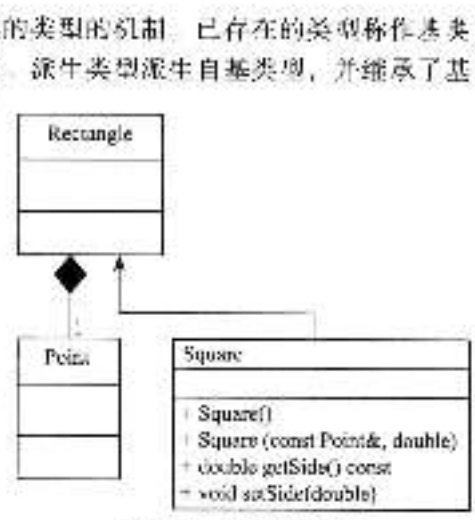


图 10.9 UML 类关构

10.6.1 Rectangle 类

一个矩形可以通过一个原点、宽度和高度来描述。因此，类声明必须要求有三个私有属性，以及支持这些属性的修改和存取方法。我们还包含了一个打印方法、一个用于在平面上移动矩形的方法，以及一个计算矩形面积的方法。计算矩形面积的方法不应当修改调用对象的属性，所以我们将在定义面积方法时使用 const 限定符。Rectangle 的类声明如下所示：

```
/*-------------------------------------------------------------------*/
/* Class declaration for Rectangle.                                  */
/* filename: rectangle.h                                             */
#include "Point.h" //Required for Point
class Rectangle
{
private:
// Declaration of data members;
double width, height;
Point origin;

public:
// Public interface
// Default constructor
Rectangle();
// Parameterized constructor
Rectangle(double w, double h, double x, double y);
// Accessor methods.
double getWidth() const;
double getHeight() const;
Point getOrigin() const;

//Mutator methods.
void setWidth(double w);
void setHeight(double h);
void setOrigin(Point p);


//Additional operations
void move(double dx, double dy);
double area() const;
void print(ostream&) const;

};
/*-------------------------------------------------------------------*/

/*                                                                   */
/* Class Implementation for Rectangle.                               */
/* filename: Rectangle.cpp                                           */
#include "Rectangle.h" //Required for Rectangle

Rectangle::Rectangle():origin(0,0)
{
   cout << "Constructing Rectangle() ..." << this << endl;
   width=height=1;
}
Rectangle::Rectangle(double w, double h,
                     double x, double y):origin(x,y)
{
   cout << "Constructing Rectangle(parameter list)..."
        << this <<endl;

  width=w;
  height=h;
}
void Rectangle::print(ostream& out) const
{
  out << "Width: " << getWidth() << " Height: " << getHeight();
  out << "\nOrigin =: (" << this->getOrigin().getX()
      << "," << this->getOrigin().getY() << ")";
}
double Rectangle::getWidth() const
```

```
  return width;
}
double Rectangle::getHeight() const
{
  return height;
}
Point Rectangle::getOrigin() const
{
  return origin;
}
double Rectangle::area() const
{
  return width*height;
}
void Rectangle::setWidth(double w)
{
  width = w;
}
void Rectangle::setHeight(double h)
{
  height = h;
}
void Rectangle::setOrigin(Point p)
{
  origin = p;
}
void Rectangle::move(double dx, double dy)
{
  origin.setX( origin.getX() + dx);
  origin.setY( origin.getY() + dy);
}
/*  ----------  ----------  ----------  */
```

注意 Rectangle 类的构造函数定义中的语法。如前文所述构造函数用于数据成员的初始化。在使用类的组合时，如果某个数据成员是一个类对象，那么使用初始化列表对类属性进行初始化比在构造函数内对属性赋值的效率要更高。初始化列表和方法头部之间要使用冒号分隔。在两个构造函数中，对于属性 origin 的初始化都会调用 Point 类的带参数的构造函数。Rectangle 类中所使用的初始化列表不是必需的，但是它提高了效率。当类中拥有非静态的 const 属性或者拥有的属性中有引用时，必须使用初始化列表。

在 Rectangle 构造函数中，我们在 cout 语句中使用了 C++ 关键字 this 打印出被构造的对象地址。关键字 this 表示指向一个类对象的指针。在成员函数中，this 指针总是被定义用来存储调用对象的地址。构造函数的调用对象就是正在被构造的对象。注意在 Rectangle::print() 中 this 指针总是被用来显式地调用 getOrigin() 方法。

构造函数：构造函数是用于构造类实例的方法。构造函数是一个与类名相同，无返回值的类成员。

语法

```
类名::类名([参数列表]) [:初始化列表]
{
   //语句块
}
```

示例

```
Point::Point (double x, double y)
{
   xCoord = x;
   yCoord = y;
}
Rectangle::Rectangle():origin(1.0,1.0)
{
   width = height = 1.0;
}
Valid References:
Point p (1.5, 2.7);
Rectangle r;
```

10.6.2 Square 类

正方形是四边相等的矩形。因为正方形是矩形，所以 Rectangle 的公共接口都应该应用于所有的 Rectangle 对象，即使有些矩形可能是正方形。如果我们希望定义新的类来实现正方形的概念，则可以使用继承从 Rectangle 类继承属性和方法，并添加新的 public 方法支持正方形的特性。Square 类的声明如下：

```
#ifndef SQUARE_H
#define SQUARE_H
/*-------------------------------------*/
/* Class Declaration for Square        */
/* Filename Square.h                   */

#include "Rectangle.h"
using namespace std;

class Square : public Rectangle
{
  public:
  //Constructors
  Square();
  Square(const Point&, double s);
  double getSide() const;
  void setSide(double);
};
/*-------------------------------------*/
#endif
```

注意类声明的第一行：

```
class Square : public Rectangle
```

: public 说明 Square 将继承所有 Rectangle 的 public 和 protected 成员。但是，派生类不会从基类继承构造函数，因此必须为每个派生类定义新的构造函数。

派生类所定义的构造函数将总是调用基类的构造函数。在派生类构造函数开始执行时，基类的默认构造函数将被隐式调用，或者显式调用基类中带参数的构造函数。当派生类的构造函数显式调用基类中带参数的构造函数时，该显式调用必须是默认构造函数中的第一条执行语句。

Square 类的实现如下：

```
/*                                                                    */
/* Class implementation for Square                                    */
/* filename: Square.cpp                                               */

#include "Square.h"

//Constructors
Square::Square()

  //Rectangle constructor called implicitly.
  cout << "Construction Square()... " << this
       << endl;

Square::Square(const Point& p, double s):Rectangle(s,s,
                                                   p.getX(),
                                                   p.getY())
{
  //Parameterized constructor explicitly called in parameter list.
  cout << "Constructing Square( Point, double)... " << this
       << endl;
}
double Square::getSide() const
{
   return getWidth();
}
void Square::setSide(double s)
{
   setWidth(s);
   setHeight(s);
}
/*                                                                    */
```

我们首先看看 Square 的默认构造函数。因为 Square 类没有其他属性需要初始化，所以在其默认构造函数中只需要调用 Rectangle 类的默认构造函数。对于默认构造函数的调用是隐式的。因此定义 Square 类的默认构造函数的语句块中只有一条打印语句在运行时追踪对象的构造。打印语句输出 this 的值。this 是 C++ 中的关键字，它被定义用作保存调用对象的地址。因此，打印语句将打印出被构造对象的地址。为了说明，在 Rectangle 类中也添加了类似的打印语句。

在 Square 类带参数的构造函数的初始化列表中显式调用了 Rectangle 类中带参数的构造函数。在 Square 类和 Rectangle 类的带参数的构造函数中，我们都加入了打印语句，用来追踪对象的构造。

方法 getSide() 和 setSide() 使用了 Rectangle 类的公共接口来访问 Rectangle 的私有属性。我们本来可以通过 Square 类继承 Rectangle 类的属性，以使得这些属性成为 protected 而非 private 属性，但这是一个不明智的设计决定。如果一个派生类通过名字来引用基类的属性，那么派生类的实现就独立于基类的实现了。派生类应该使用基类的公共接口来避免高度独立于基类的实现。

下面给出了一个用于测试 Square 类的程序：

```
/*                                                                    */
/* Program chapter10_7                                                */
/*  filename: testSquare.cpp                                          */
/* This program tests the Square class.                               */
#include<iostream>
```

```
#include "Square.h"
using namespace s d;

int main()
{
   //Test construction.
   Point p1(5,4);
   Square s1, s2(p1, 4);

   /* Test print()                                  */
   /* Inherited from Rectangle.                     */
   cout << "Square s2 has: " << endl;
   s2.print(cout);
   cout << endl;

   //Test getSide().
   cout << "Length of side of s2 is "
        << s2.getSide() << endl;

   //Test area method from Rectangle
   cout << "Area of s2 is "
        << s2.area() << endl;

   cout << "**********************\n";
   //Test setSide()
   s1.setSide(3,2);

   //Test print()
   cout << "Square s1 has: " << endl
   s1.print(cout);
   cout << endl;
   return 0;
}
/*****************************************/
```

程序 chapter10_7 的一次示例运行的输出如下：

```
Constructing Rectangle() ...0xbffffb68
Construction Square()... 0xbffffb68
Constructing Rectangle(parameter list)...0xbffffb88
Constructing Square(Point, double)... 0xbffffb88
Square s2 has:
Width: 4 Height: 4
Origin at: (5,4)
Length of side of s2 is 4
Area of s2 is 16
**********************
Square s1 has:
Width: 3.2 Height: 3.2
Origin at: (3,0)
```

程序 chapter10_7 的输出追踪了 Square s1 和 s2 的构造。s1 由默认构造函数定义，地址为 0xbffffb68。s2 由带参数的构造函数定义，地址为 0xbffffb88。从输出我们看到 Rectangle 在 Square 之前被构造。这是可以理解的，因为 Square 派生自 Rectangle。

10.6.3 Cube 类

立方体是正方形的三维表示。这种关系暗示着实现立方体概念的新类型可以从 Square 类派生。在派生类中将添加用于计算立方体体积的方法，我们将重载“<<”来打印出一个

立方体而不是矩形的值。在图 10.10 中给出了 Cube 的类图。

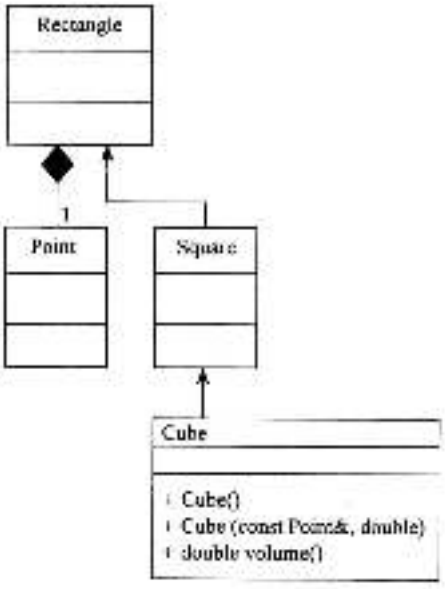


图 10.10 Cube 的 UML 示意图

Cube 的定义如下：

```
#ifndef CUBE_H
#define CUBE_H
/*---------------------------------------------------------*/
/* Class declaration for Cube                              */
/* filename: Cube.h                                        */

#include "Square.h"//Required for Square
#include "Point.h" //Required for Point
#include <iostream> //Required for ostream

class Cube : public Square
{
   public:
   Cube();
   Cube(const Point& p, double);
   double volume();
   void print(ostream&) const;
};
#endif
/*---------------------------------------------------------*/
/* Class implementation for Cube                           */
/* filename: cube.cpp                                      */

#include "Cube.h" //Required for Cube
#include <cmath> //Required for pow()

   Cube::Cube():Square()
   {
     cout << "Constructing Cube()..."
          << this << endl;
   }
   Cube::Cube(const Point& p, double s):Square(p,s)
   {
```

```
    cout << "Constructing Cube(Point, double)..."
         << this << endl;
}
double Cube::volume()
{
  return pow(getSide(), 3);
}

void Cube::print(ostream& out) const
{
  Rectangle r;
  r = *this;  //A cube is a rectangle.

  //Print the depth
  out << "Depth: " << this->getSide() << " ";

  //Call Rectangle << to finish the job.
  Rectangle::printOut();
}
/*-------------------------------------------------------*/
```

下面给出了一个测试 Cube 类的程序。

```
/*-------------------------------------------------------*/
/* Program chapter10_8                                   */
/* Filename: testCube.cpp                                */
/* This program tests the Cube class                     */

#include<iostream>//Required for cout
#include "Point.h"//Required for Point
#include "Cube.h"//Required for Cube
using namespace std;

int main()
{
  Point p1(4,2);

  //Test constructors.
  Cube c1, c2(p1, 3);

  //Test << operator
  cout << "c1: ";
  c1.print(cout);
  cout << endl;
  cout << "c2: " ;
  c2.print(cout);
  cout << endl;

  //Test volume.
  cout << "Volume of a c2 is " << c2.volume()
       << endl;
  return 0;
}
/*-------------------------------------------------------*/
```

程序 chapter10_8 的一次示例运行的输出如下：

```
Constructing Rectangle() ...0xbffffbe0
Construction Square()... 0xbffffbb8
Constructing Cube() ...0xbffffb68
Constructing Rectangle(parameter list)...0xbffffb98
```

```
Constructing Square( Point, double)... 0xbffffb68
Constructing Cube(Point, double)...0xbffffb90
c1: Depth: 1 Width: 1 Height: 1
Origin at: (0,0)
c2: Depth: 3 Width: 3 Height: 3
Origin at: (4,2)
Volume of c c2 is 27
```

在这里当派生类对象被构造时，我们再次看到了构造函数的调用顺序。

10.7 虚方法

所有在 Rectangle、Square 和 Cube 类中定义的方法，默认都是非虚（non-virtual）方法。当一个非虚方法被调用时，所执行的方法是由调用对象的静态类类型定义的方法。考虑下面的例子：

```
...
Point p1(2,1);
Cube c1(p1,4);
Rectangle r1(1,3, 5,2.5);
r1 = c1;
r1.print(cout);
...
```

在上面的例子中，c1 的值被赋给了 r1。因为一个 Cube 是一个 Rectangle，所以这是合法的。但是，当 r1 调用 print 方法时，它调用的是 Rectangle 中定义的 print 方法，而不是 Cube 中定义的，因为 r1 的静态类类型是 Rectangle。上面代码段的输出将是下面的形式：

```
Constructing Rectangle(parameter list)...0xbffffb68
Constructing Square( Point, double)... 0xbffffb08
Constructing Cube(Point, double)...0xbffffbc0
Constructing Rectangle(parameter list)... 0xbffffb90
Width: 4 Height: 4
Origin at: (2,1)
```

如果一个方法被定义为虚（virtual）方法，且指向对象的指针或引用不再使用静态定义的对象，C++ 支持动态绑定（dynamic binding）。动态绑定表示在运行时将对象与特定类型绑定。考虑下面使用指向对象的指针的示例：

```
...
//Define an array of pointers to Rectangles
Rectangle* rPtrs[4];

//Define 2 Cubes and 2 Squares.
Point p1(2,1), p2;
Cube c1(p1,4);
Cube c2(p1,5);
Square s1, s2(p2,5);

rPtrs[0] = &c1;
rPtrs[1] = &c2;
rPtrs[2] = &s1;
rPtrs[3] = &s2;

for(int i=0; i<4; ++i)
{
   rPtrs[i]->print(cout);
   cout << endl;
}
...
```

数组 rPtrs 保存了指向 Rectangle 的指针，但是这些指针被用来引用 Cube 和 Square。又为在 Rectangle 中定义的 print 方法是非虚方法，所以在 Rectangle 类中定义的 print 方法将被调用 4 次。如上面代码段生成的输出：

```
Constructing Rectangle(parameter list)...0xbffffb20
Constructing Square( Point, double)... 0xbffffb20
Constructing Cube(Point, double)...0xbffffb30
Constructing Rectangle(parameter list)...0xbffffb40
Constructing Square( Point, double)... 0xbffffb40
Constructing Cube(Point, double)...0xbffffb50
Constructing Rectangle() ...0xbffffb60
Constructing Square()... 0xbffffb60
Constructing Rectangle(parameter list)...0xbffffb80
Constructing Square( Point, double)... 0xbffffb80
Width: 4 Height: 4
Origin at: (2,1)
Width: 5 Height: 5
Origin at: (2,1)
Width: 1 Height: 1
Origin at: (0,0)
Width: 5 Height: 5
Origin at: (0,0)
```

当在 print 方法原型中使用 virtual 关键字时，即

```
virtual void print(ostream&) const;
```

上面的代码段将生成下面的输出：

```
Constructing Rectangle(parameter list)...0xbffffb10
Constructing Square( Point, double)... 0xbffffb10
Constructing Cube(Point, double)...0xbffffb10
Constructing Rectangle(parameter list)...0xbffffb34
Constructing Square( Point, double)... 0xbffffb34
Constructing Cube(Point, double)...0xbffffb34
Constructing Rectangle() ...0xbffffb58
Constructing Square()... 0xbffffb58
Constructing Rectangle(parameter list)...0xbffffb7c
Constructing Square( Point, double)... 0xbffffb7c
Depth: 4 Width: 4 Height: 4
Origin at: (2,1)
Depth: 5 Width: 5 Height: 5
Origin at: (2,1)
Width: 1 Height: 1
Origin at: (0,0)
Width: 5 Height: 5
Origin at: (0,0)
```

注意 cube 和 square 被打印时，使用的是对应于数组 rPtrs 中指针的动态类型所指向的方法的版本。这里不需要对 Square 或 Cube 进行修改。在 C++ 中，动态绑定只有通过使用虚方法、指针或引用来支持。

练习

假定 Rectangle 类中定义的 print() 函数是一个虚函数，给定下面的声明语句：

```
Point p1(1,2), p2;
Cube *cptr, c1(p1,4);
Square *sptr, s1(p2,7);
Rectangle *rptr, r1(2,2,7,9);
```

给出下面代码段生成的输出，使用本节所定义的 Cube 类和 Square 类。

```
1. ptr = &c1;
   c1.print(cout);
   ptr->print(cout);
```

```
2. r1 = c1;
   r1.print(cout);
```

```
3. ptr = &s1;
   r1 = s1;
   ptr = &r1;
   r1.print(cout);
   cout << endl;
   ptr->print(cout);
```

10.8 解决应用问题：可重复的囚徒困境

可重复的囚徒困境（Iterated Prisoner's Dilemma，IPD）是游戏理论中一个流行的游戏。囚徒困境游戏的传统形式中有两个玩家。每个玩家必须从两种可能的选择中选择一种：背叛或合作。你和你对手选择的组合决定了一种结果形式，通常使用分数来表示的。

"囚徒困境"的名字来源于下面的场景：两个囚犯因犯轻微罪正在服刑。但是他们都被怀疑犯有更严重的罪行。警察分别单独对两个囚徒询问相同的问题。每个囚徒必须给出下面选择之一：

1）牵连另一名囚徒（即背叛，相对于另一名囚徒而言），并获得假释。

2）不牵连另一名囚徒（即合作，相对于另一名囚徒而言），并继续因轻微罪服刑。

在这个例子中，每个嫌疑人都只有一种选择和一种结果。如果都选择合作，那么每个人都将在监狱完成剩下的服刑期。如果都背叛，每个人都会被假释，但是每个人都会被怀疑犯有更严重的罪行并因此被判新的、更长的刑期。如果一个人背叛而另一个合作，那么背叛者将被释放，而合作者将在监狱里度过更长的日子。在这种情况下你会怎么办呢？你希望你的囚徒伙伴怎么做呢？如果你的囚徒伙伴将你牵连入罪，你将来会怎么对他呢？在许多年中这种情况都是一些有趣的研究的基础，包括政治科学、生物学和经济学。

在 IPD 游戏的实现中，你将会与你的对手有重复的交互，而不是只有一次。因此名为可重复的囚徒困境。你在游戏中的第一次选择（背叛或者合作），是在不知道对手选择的情况下决定的。但是，在后续的选择中，你可以根据对手上一次的选择来决定是否合作或背叛。这就是策略变得有趣的时候。

你将与你的对手竞争，还必须与其他与你和你的对手对抗的玩家竞争。游戏的结果是累计在给定选择次数的情况中点数的最大数目。囚徒困境的收益表如下：

IPD 收益表

	合作	背叛
合作	3,3	0,5
背叛	5,0	1,1

如果你认真地看这个表，会注意到最高的收益出现在你的对手合作而你选择背叛时。但是，如果两个玩家都合作，那么每个玩家都会得到比两人都背叛时更高的收益。

如果游戏只由一次迭代组成，那么你可以认为在第一次选择中背叛是合理的，这会给你赢得游戏的最好机会，因为你不会再遇到你的对手。但是，因为你还会多次遇到你的对手，并且对于知道你的行动，如果你希望与对手形成一种合作关系，那么可能在统计点数时会有更好的选择。开发一个合作策略，避免自己成为背叛者，这将使游戏变成一个有趣的行为实型，同时也是一次有挑战的编程任务。网络是有关 IPD 信息的良好来源。好的策略会结合

学习对手策略的AI算法并优化自身。一个最简单有效的策略称作“针锋相对”。使用这种策略，你的下一次选择总是与对手上一次的选择相同。“针锋相对”是我们将设计的策略之一。

为了使用C++实现可重复的囚徒困境，我们将定义一个玩家类作为基类。玩家类的虚成员方法将被定义成进行游戏。但是，每个玩家的实际选择将由派生自玩家类的类来定义。派生类将覆写在玩家类中定义的函数。基类Player的定义如下：

```
/*                                                          */
/* The Player class declaration.                            */
/* File Player.h                                            */

#ifndef PLAYER_H
#define PLAYER_H
#include <iostream>
class Player
{
public:
// Constructor
  Player();
// Accessor function
  virtual int get_score() const;

// Print player's name.
  virtual void print_name();

// Print name of the player's algorithm.
  virtual void print_algorithm();

// Player's first move.
  virtual bool play();

// Player's subsequent moves.
  virtual bool play(bool opponents_last_play);

// Cumulative score.
  virtual void accumulate(int);
protected:
int score;
};
#endif
/*                                                          */
/*                                                          */
/*                                                          */
/* The Player class implementation.                         */
/* This implements the always cooperate strategy.           */
/* File Player.cpp                                          */

#include "Player.h"

// Constructor
   Player::Player() : score(0)
   {
   }

// Accessor function
   int Player::get_score() const
   {
      return score;
   }

// Print player's name.
   void Player::print_name()
```

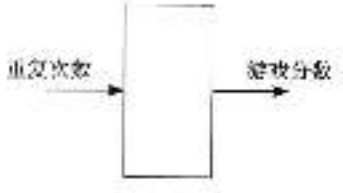
```
{
 cout << "Base Class Player";
}

// Print name of the player's algorithm.
void Player::print_algorithm()
{
 cout << "Always Cooperate\n";
}

// Implement player's first move.
bool Player::play()
{
 return true;
}

// Implement player's subsequent moves.
bool Player::play(bool opponents_last_play)
{
 return true;
}

// Keep a running cumulative score.
void Player::accumulate(int x)
{
  score += x;
```

1. 问题描述

编写一个程序实现有两个玩家的可重复囚徒困境。使用 Player 类作为基类，派生出一个新的类来实现"针锋相对"策略。使用这个策略与 Player 类的默认策略对抗。

2. 输入/输出描述

程序的输入是游戏中的重复次数。输出是每个玩家的总分数和游戏的赢家。



3. 用例

如果两个玩家在每局中都合作，游戏重复 10 次，那么每个玩家的分数都是 30。程序将输出下面的结果：

```
Player1 and Player 2 tied at 30 points each.
```

4. 算法设计

我们首先给出分解提纲，因为它将解决方案分解为一组顺序执行的步骤。

分解提纲

1）为每个玩家定义一个 Player 对象，设置游戏；

2）输入重复次数；

3）玩家 1 做第一次选择；

4）玩家 2 做第一次选择；

5）确定得分；

6）完成剩下的指定重复次数的选择和分数计算；

7）打印分数。

步骤1要求每个玩家都为游戏设计一个策略并定义一个派生自基类Player的类来实现它们的策略。然后定义初始玩家类的对象，以及有关玩家的游戏报告。我们将编写一个函数来设置游戏。步骤2～6组成了游戏的核心。我们将编写一个函数来完成这些步骤，并执行要求重复的次数。我们还会编写一个函数来确定每个回合的分数。该函数将在步骤5和步骤6中使用。步骤7打印最后的分数。这些也将在一个函数中完成。现在我们将给出细化的伪代码：

细化的伪代码

```
main():
    setup(player1, player2)
    play_game(player1, player2)
    report(player1, player2)
end main
play_game(player1, player2, player1, player2):
    input number of iterations.
    p1move = player1->play()
    p2move = player2->play()
    player1->accumulate(payoff(p1move, p2move))
    player2->accumulate(payoff(p2move, p1move))
    for(count =1, count < iterations, count++)
       pleave = p1move
       p1move = player1->play(p2move)
       p2move = player2->play(pleave)
       player1->accumulate(payoff(p1move, p2move))
       player2->accumulate(payoff(p2move, p1move))
end play_game
payoff(p1move, p2move):
   if(p1move)
      if(p2move)
         return 5
      else
         return 0
   else
      if(p2move)
         return 5
      else
         return 1
end payoff
report(player1, player2, player1, player2):
   if(player1->score > player2->score)
      output player 1 wins
   elseif(player1->score < player2->score)
      output player 2 wins
   else
      output tie
end report
```

我们的main函数调用了用于实现游戏的三个函数。每个函数都有一个player*类型的引参。因为Player类在其类声明中使用了virtual关键字，所以引参所引用的动态对象将调用由对象的动态数据类型所定义的覆写函数，这样游戏功能对多个玩家也是可用的。现在我们准备将伪代码转换成C++代码：

```
/*-----------------------------------------*/
/* Program chapter13_9                     */
/* This program implements a version of the Iterated
Prisoner's Dilemma */
#include <iostream>
using namespace std;

#include "player.h"      //base class player
#include "TitForTat.h"  //TitForTat player

//Function prototypes
void setup(player*, player*);
void play_game(player*, player*);
void report(player*, player*);
int payoff(bool move, bool opponent_move);

int main()
{
// Declare objects.
   player p1;
   TitForTat p2;
   player* ptr1 = &p1;
   player* ptr2 = &p2;

// Notify players.
   setup(ptr1, ptr2);
   play_game(ptr1, ptr2);
   cout << endl;
   report(ptr1, ptr2);

   return 0;
}
void setup(player* p1, player* p2)
{
  // Announce players.
  p1->print_name();
  cout << " is playing ";
  p1->print_algorithm();
  cout << endl;
  p2->print_name();
  cout << " is playing ";
  p2->print_algorithm();
  cout << endl;
}

int payoff(bool move, bool opponent_move)
{
  if (move)
  {
    if (opponent_move)
    {
      return 3; // Both cooperate.
    }
    else
    {
      return 0; // I cooperate, opponent defects.
    }
  }
  else
  {
    if (opponent_move)
    {
```

```
        return 5; // I defect, opponent cooperates.
      }
      else
      {
        return 1; // Both defect.
      }
    }
}
/* Play a single game of the iterated
 * prisoner's dilemma between two players.
 */
void play_game(player* p1, player* p2)
{
// Declare objects.
   int max_iterations;
   bool p1_move, p2_move, old_p1_move;
   cout << "Enter the number of iterations for the game: ";
   cin >> max_iterations;

   p1_move = p1->play();  // get the first move.
   p2_move = p2->play();  // get the first move.
   for(int i=1; i<max_iterations; i++)
   {
      old_p1_move = p1_move;
      p1_move = p1->play(p2_move);      // get the next move
      p2_move = p2->play(old_p1_move);  // get the next move

      // Update the scores for this round of play.
      p1->accumulate(payoff(p1_move, p2_move));
      p2->accumulate(payoff(p2_move, p1_move));
   }
}
void report(player *p1, player *p2)
{
  if (p1->get_score() > p2->get_score())  // Player 1 won.
  {
    p1->print_name();
    cout << " (" << p1->get_score() << ") beat ";
    p2->print_name();
    cout << " (" << p2->get_score() << ").\n";
  }
  else if (p2->get_score() > p1->get_score()) // Player 2 won.
  {
    p2->print_name();
    cout << " (" << p2->get_score() << ") beat ";
    p1->print_name();
    cout << " (" << p1->get_score() << ").\n";
  }
  else // The players tied.
  {
    p1->print_name();
    cout << " and ";
    p2->print_name();
    cout << " tied at " << p1->get_score() << " each.\n";
  }
}
```

类 TitforTat 派生自 play 类。类 TitforTat 的完整定义如下：

```
#ifndef TITFORTAT_H
#define TITFORTAT_H

#include "player.h"
/*--------------------------------------------------------------------*/
/*                                                                    */
/* This implements the Tit for Tat strategy.                          */
/* filename TitforTat.h                                               */
class TitforTat : public Player
{
public:

// Print player's name.
  void print_name();

// Print name of the player's algorithm.
  void print_algorithm();

// Implement player's first move.
  bool play();

// Implements player's subsequent moves.
  bool play(bool opponents_last_play);

};
#endif

/* This implements the Tit for Tat strategy with initial defect. */
#include <iostream>
using namespace std;

#include "TitForTat.h"

  void TitforTat::print_name()
  {
    cout << "Jeanine ";
  }

// Print name of the player's algorithm.
  void TitforTat::print_algorithm()
  {
     cout << "Tit for Tat";
  }

// Implement player's first move.
  bool TitforTat::play()
  {
    return false;
  }

// Implements player's subsequent moves.
  bool TitforTat::play(bool opponents_last_play)
  {
    return opponents_last_play;
  }
```

5. 测试

程序的一次示例运行如下所示，使用 Jeanine 与基类玩家一起游戏：

```
Base Class Player is playing Always Cooperate.

Jeanine is playing Tit for Tat.

Enter the number of iterations for the game: 10

Jeanine (22) beat Base Class Player (7).
```

当玩家 Jeanine 背叛，基类玩家合作时，分数上的差异为第一次选择的结果。在剩下的游戏中，两个玩家都选择合作。

修改

定义一个派生自基类的玩家类，实现一个总是背叛的算法。

1. 运行程序 chapter10_9，使用总是背叛对抗总是合作。在 25 次重复后的结果是多少？
2. 运行程序 chapter10_9，使用总是背叛对抗针锋相对。在 25 次重复后的结果是多少？
3. 将关键词 virtual 从 player 类的声明中去掉。运行程序 chapter10_9，输出是什么？

本章小结

在 C++ 中的类机制支持面向对象的编程。类被用于定义新的抽象数据类型。对于操作符的重载允许这些新的数据类型为工作在内建操作符类似定义。友元和操作符的重载为这些新的数据类型提供了简单方便的用法。继承和虚函数的使用组成了大型程序面向对象设计的基础。关于这些高级主题的使用，在 Pixel 类的设计和派生自 Rectangle 类的两个新类的设计中进行了说明。继承和虚函数的作用在实现可重复的囚徒困境问题中进行了演示。

关键术语

base class（基类）
binary tree（二叉树）
derived class（派生类）
dynamic binding（动态绑定）
generic programming（泛型编程）
image processing（图像处理）
inheritance（继承）
iterated prisoner's dilemma（可重复的囚徒困境）
overloading operators（重载操作符）
pixel（像素）
polymorphism（多态）
public inheritance（公共继承）
this pointer（this 指针）
tree traversal（树遍历）
virtual function（虚函数）

C++ 语句总结

重载的 "+" 操作符成员函数原型

```
返回数据类型 operator +(数据类型);
```

示例：

```
Pixel operator +(Pixel) const;
```

重载的 "+" 操作符友元函数原型

```
friend 返回数据类型 operator+ (数据类型, 数据类型);
```

示例：

```
friend Pixel operator +(Pixel, Pixel);
```

重载的“<<”操作符友元函数原型

```
friend ostream& operator <<(ostream&, 数据类型);
```

示例：

```
friend ostream& operator <<(ostream&, Pixel);
```

重载的“>>”操作符友元函数原型

```
friend istream& operator >>(istream&, 数据类型);
```

示例：

```
friend istream& operator >>(istream&, Pixel&);
```

公共继承的类定义

```
class 类名 : public 基类名
```

示例：

```
class Square : public Rectangle
{
public:
...
Print();
...
};
```

虚函数的说明

```
语法：virtual 返回数据类型 函数名(参数列表);
```

示例：

```
virtual void print(ostream&);
class Complex operator
template <typename identifier>
```

注意事项

使用基类的构造函数和成员函数为派生的数据成员赋值。

调试要点

1. 重载“<<”操作符的函数必须返回一个 ostream 引用。
2. 重载“>>”操作符的函数必须返回一个 istream 引用。
3. 派生类必须包括一组构造函数。
4. 派生类不能从基类继承构造函数。

习题

判断题

1. 派生类继承了基类的构造器函数。
2. 重载函数必须有唯一的函数特征。
3. 重写函数的函数特征必须唯一。
4. 虚函数被调用时是基于调用对象的动态数据类型。

5. 类的友元函数对类的 public、protected 和 private 成员函数都有访问权限。

多选题

6. 假定名为 Myclass 的类声明中有下面的函数原型：

```
friend void inputlstream&, Myclass&);
```

关于该原型，下面（　　）函数头是合法的。

```
(a) friend void inputlstream& in, Myclass C)
(b) extern void Myclass::inputlstream& in, Myclass C)
(c) Myclass::input(lstream& in, Myclass&
(d) void input(lstream& r, Myclass&
(e) Myclass::friend inputlstream& in)
```

7. 假定名为 Myclass 的类声明中有下面的函数原型：

```
virtual void fn(lstream&);
```

关于该原型，下面（　　）函数头是合法的。

```
(a) virtual void Input(istream& in)
(b) virtual void Myclass::input(istream& in)
(c) void Myclass::input(istream& in)
(d) void input(istream& in)
(e) void Myclass::virtual inputlstream& in)
```

编程题

8. 为 BinaryTree 类添加一个返回方法，返回树上叶子节点的数目。

9. 编写一个程序，使用 BinaryTree 类对一组从数据文件读取的数据进行排序。你的程序应当打开一个数据文件，读取数据，并存入你的树中，然后使用 inOrder 打印方法打印出数据。

10. 从网络上下载一幅图像，实现一个函数，通过将每间隔一个的像素设为黑色达到图像的“浸色”效果。不允许出现黑色的垂直线。

11. 下载一幅图像，实现函数，以完成对图像的褪色效果。

12. 从网络中下载一个函数，实现一个函数，对图像完成你自己的创意修改。

13. 定义一个有理数类。一个有理数是一个可以表示成两个整数相除的数，如 1/2、2/3、4/5。一个有理数使用两个整数对像表示：分子和分母。重载“<<”和“>>”操作符以及算术操作符，完成下面的操作：

```
a/b + c/d = (a*d + b*c) / (b*d) (addition)
a/b - c/d = (a*d - b*c) / (b*d) (subtraction)
(a/b) * (c/d)   (a*c)/(b*c) (multiplication)
(a/b) / (c/c) = (a*c) /(c*b) (division)
```

14. WAR 游戏是一个简单的两人卡牌游戏。每个玩家收到一副 52 张的扑克牌。每副牌都经过了洗牌，牌面朝下放在玩家面前。然后开始一系列回合的游戏。在每个回合，玩家都将牌堆最上面的牌拿出并翻开。牌面点数大的玩家赢得两张牌。如果两张牌点数相等，那么出现 WAR。在出现 WAR 的情况下，两个玩家都从他们的牌堆中抽取 6 张牌，并将牌面朝下。然后抽取第 7 张牌，并翻开牌面。拥有高点数牌的玩家赢得已经使用过的所有牌。如果两张牌点数相等，那么出现 WAR。在传统游戏中，当某个玩家的卡牌用完时游戏结束。但你可能希望简化游戏过程，使得当所有 52 张牌都被使用过时就终止游戏。写一个程序仿真 WAR 游戏，使用第 8 章开发的 card 类。

15. 写一个用于仿真自动售货机的仿真器。在机器的初始状态建立后（即待售的物品、价格、以及初始的存货量），用户可以通过交互对话框与仿真器交互。在对话框中，用户可以看到机器的当前状态，向机器内投币（美分），购买机器内的物品，得到物品和找零，假定自动售货机有 9 种待售物品。一个物品可以使用 3 个对象表示：

表示物品名的字符串
物品的价格（美分）
初始可售的物品数量
下面的数据可用于建立机器的初始状态：

```
Tortilla_Chips 60 3
Pretzels 60 10
Popcorn 60 5
Cheese_Crackers 40 2
Creme_Cookies 85 1
Mint_Gum 25 5
Chocolate_Bar 90 3
Licorice 85 9
Fruit_Chews 65 7
```

从概念上看，物品可以组织成一个 3×3 的矩阵，如下所示：

```
A1  A2  A3
B1  B2  B3
C1  C2  C3
```

定义两个类（一个物品类和一个机器类）来实现这个仿真器。

16. 修改程序 chapter10_9，允许两个以上的玩家来玩 IPD 游戏。这可以被看做是一系 IPD 比赛，每个玩家都与其他每个玩家在一次 IPD 中对抗。在这种情况下，你可能希望你的策略随着对手的不同而不同。赢家是在比赛中获得点数最多的玩家。
17. 修改 IPD 比赛程序，允许玩家在比赛中与其他每个玩家博弈一次以上。在游戏中，你可以通过"记忆"你对手的策略来改进的策略，并从上一次与其碰面的时候。
18. 定义一个 Image 类。一个 Image 有一个 Pixel 类型的二维数组和支持对图像的平滑化和亮化的成员函数。

C++ 标准库

本附录给出了标准 C++ 库中部分头文件定义的信息的简要讨论。这些简短的讨论并不为使用这些函数提供所有细节，但为确定这些函数是否适用于特定的应用提供了足够信息；你可以从网络上获得更多的细节。网站 http://www.cplusplus.com/ref/ 就是一个很好的网络资源。

<cassert>

头文件 <cassert> 提供了断言函数的定义。断言函数用于程序测试时提供诊断信息。这些诊断信息与系统相关，被存储于标准错误文件中，在程序执行完成后可以访问错误文件以确认诊断信息。

<cctype>

头文件 <cctype> 定义了用于字符测试和转换的若干函数。在函数原型语句及相应的讨论中使用了下列定义：

数字	字符 0123456789 中的一个
十六进制数字	数字中的一个或者字符 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	一个大写或小写字母
字母数字字符	一个数字或字母字符
标点字符	这些字符中的一个：! " # % & ' ( ) ; < = > ? [\ ] * + , - . / : ^
可视字符	一个字母数字字符或者一个标点字符
打印字符	可视字符和空格
移动控制字符	控制字符之一：换页（FF）、换行（NL）、回车（CR）、水平制表（HT）、垂直制表（VT）
空白	空格或者移动控制字符之一
控制字符	移动控制字符之一、响铃（BEL）或退格（BS）

现在我们列出每个函数原型，并对每个函数给出相应的简要定义：

```
int  isalnum(int c);
```

当且仅当输入字符为一个数字或大小写字母时返回一个非零（真）值

```
int  isalpha(int  c);
```

当且仅当输入字符为大小写字母时返回一个非零（真）值

```
int  iscntrl(int c);
```

当且仅当输入字符是一个控制字符时返回一个非零（真）值

```
int  isdigit(int c);
```

当且仅当输入字符是一个数字时返回一个非零（真）值

```
int  isgraph(int c);
```

当且仅当输入字符是一个可视化字符时返回一个非零（真）值

```
int  islower(int c);
```

当且仅当输入字符是一个小写字母时返回一个非零（真）值

```
int  isprint(int c);
```

当且仅当输入字符是一个可打印字符时返回一个非零（真）值

```
int  ispunct(int c);
```

当且仅当输入字符是一个标点字符时返回一个非零（真）值

```
int  isspace(int c);
```

当且仅当输入字符是一个空白字符时返回一个非零（真）值

```
int  isupper(int c);
```

当且仅当输入字符是一个大写字母时返回一个非零（真）值

```
int  isxdigit(int c);
```

当且仅当输入字符是一个十六进制数字时返回一个非零（真）值

```
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

<climits>

头文件<climits>提供了若干宏定义，这些宏定义给出了整型值的范围限制和特征。这些宏和它们的定义如下：

```
int  CHAR_BIT;
```

最小的非位值的比特数

```
int  CHAR_MIN;
int  CHAR_MAX;
```

char 类型的最大值和最小值

```
int  INT_MIN;
int  INT_MAX;
```

int 类型的最大值和最小值

```
int  LONG_MIN;
int  LONG_MAX;
```

long int 类型的最大值和最小值

```
int  MB_LEN_MAX;
```

多字节字符的最大字节数

```
int  SCHAR_MIN;
int  SCHAR_MAX;
```

signed char 类型的最大值和最小值

```
int  SHRT_MIN;
int  SHRT_MAX;
```

short int 类型的最大值和最小值

```
int  UCHAR_MAX;
```

unsigned char 类型可最大值

```
int  UINT_MAX;
```

unsigned int 类型的最大值

```
int  ULONG_MAX;
```

unsigned long int 类型的最大值

```
int  USHRT_MAX;
```

unsigned short int 类型的最大值

<cmath>

头文件 <cmath> 定义了许多有用的科学计算函数。

```
double  acos(double x);
```

计算 x 的反余弦值，x 必须在 [-1, 1] 中，返回一个 [0, π] 之间的弧度值

```
double  asin(double x);
```

计算 x 的反正弦值，x 必须在 [-1, 1] 中，返回一个 [-π/2, π/2] 之间的弧度值

```
double  atan(double x);
```

计算 x 的反正切值，返回一个 [-π/2, π/2] 之间的弧度值

```
double  atan2(double y, double x);
```

计算 y/x 的反正切值，返回一个 [-π, π] 之间的弧度值

```
int  ceil(double x);
```

将 x 向正无穷方向最接近的整数取整

```
double  cos(double x);
```

计算 x 的余弦值，这里的 x 采用弧度制

```
double  cosh(double x);
```

计算 x 的双曲余弦值，等于 (e^x+e^-x)/2

```
double  exp(double x);
```

计算 e^x 的值，这里 e 是自然对数的底，近似等于 2.718282

```
double  fabs(double x);
```

计算 x 的绝对值

```
int  floor(double x);
```

求 x 向负无穷方向最接近的整数取整

```
double  log(double x);
```

计算 x 关于底为 e 的自然对数 ln x；如果 x ≤ 0 将出错

```
double  log10(double x);
```

计算 x 以 10 为底的常用对数 $\log_{10}x$；如果 x ≤ 0 将出错

```
double  pow(double x, double y);
```

计算 x 的 y 次幂 x^y；如果 x=0 且 y ≤ 0，或者如果 x<0 且 y 不是整数，那么将出错

```
double  sin(double x);
```

计算 x 的正弦值，这里 x 采用弧度制

```
double  sinh(double x);
```

计算 x 的双曲正弦值，等于 $(e^x-e^{-x})/2$

```
double  sqrt(double x);
```

计算 x 的平方根，这里 x ≥ 0

```
double  tan(double x);
```

计算 x 的正切值，这里 x 采用弧度制

```
double  tanh(double x);
```

计算 x 的双曲正切值，等于 sinh x / cosh x

<cstdlib>

头文件 <cstdlib> 定义了不与其他任何头文件相容的类型、宏和函数。类型 div_t 和 ldiv_t 是用于存储商和余数的结构。下面列出了宏：

```
NULL
```

一个值为二进制 0 的整数

```
EXIT_FAILURE
EXIT_SUCCESS
```

用于返回给主机的分别表示不成功终止状态和成功终止状态的整型表达式

```
RAND_MAX
```

表示 RAND 函数返回的最大值的整型表达式

```
MB_CUR_MAX
```

用于表示多字节字符中最大字节数目的正整数表达式

最常用于工程应用中的函数如下（包括函数原型和简要的描述）：

```
void  abort(void);
```

使程序异常终止

```
int  abs(int x);
long int  labs(long int x);
```

计算整数 k 的绝对值

```
int  atexit(void (*func)(void));
```

注册一个由 func 指向的无参数函数，在程序正常终止时执行该函数

```
  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);
```

将 s 指向的初始部分转换成数值表示

```
void*  bsearch(const void *key, const void *base, size_t n,
               size_t
size,  int(*compar)(const void *,const void *));
```

在有 n 个对象的数组中搜索由 key 所指向的值

```
void*  calloc(size_t n, size_t size);
```

为一个包含 n 个对象的数组分配空间，每个对象的大小为 size

```
 div_t  div(int numer, int denom);
ldiv_t  ldiv(long int numer, long int denom);
```

计算 numer 数以 denom 的商和余数

```
void  exit(int status);
```

使程序正常终止

```
void  free(void *ptr);
```

释放 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 *));
```

将一个包含 n 个对象的集合进行升序排列

```
int  rand(void);
```

返回一个 0～RAND_MAX 的伪随机整数

```
void*  realloc(void *ptr, size_t size);
```

改变 ptr 所指向的对象的大小

```
void  srand(unsigned int seed);
```

使用种子初始化一个 RAND 函数产生的新的随序列

<cstring>

头文件 <cstring> 定义了类型 size_t，该类型是一个无符号整数，同时定义了宏 NULL，其值为二进制 0。此外，头文件定义了大量操作字符串的函数。

```
void*  memchr(const void *s, int c, size_t n);
```

返回s指向的对象的前n个字符中c第一次出现的位置的指针

```
int  memcmp(const void *s, const void *t, size_t n);
```

当s指向的字符串大于、等于或小于t指向的字符串时，相应地返回一个大于、等于或小于0的整数

```
void*  memcpy(void *s, const void *t, size_t n);
```

从t指向的对象中复制n个字符到s所指向的对象中

```
void*  memmove(void *s, const void *t, size_t n);
```

使用一块临时区域，将t所指向的对象中复制n个字符到s所指向的对象中

```
void*  memset(void *s, int c, size_t n);
```

将c的值复制到s所指向对象的前n个字符中

```
char*  strcat(char *s, const char *t);
```

将t所指向的字符串连接到s所指向字符串的末尾；返回指向字符串s的指针

```
char*  strchr(const char *s, int c);
```

返回字符c在字符串s中第一次出现的位置的指针

```
int  strcmp(const char *s, const char *t);
```

将字符串s和t进行逐字符的比较；当字符串s大于、等于或小于t时，分别返回一个大于、等于或小于0的整数

```
int  strcoll(const char *s, const char *t);
```

字符串s大于、等于或小于字符串t时，分别返回一个大于、等于或小于0的整数

```
char*  strcpy(char *s, const char *t);
```

将字符串t复制到字符串s中；返回一个指向字符串s的指针

```
size_t  strcspn(const char *s, const char *t);
```

返回字符串s从开头开始不含字符串t中任意字符的字符串长度

```
size_t  strlen(const char *s);
```

返回字符串s的长度

```
char*  strncat(char *s, const char *t, size_t n);
```

将字符串t中至多n个字符连接到字符串s上；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t进行逐字符比较，至多比较n个字符；当字符串s大于、等于或小于t时，分别返回一个大于、等于或小于0的整数

```
char*  strncpy(char *s, const char *t, size_t n);
```

从字符串t中复制至多n个字符到字符串s中；如果t的字符数少于s，那么将使用null字符对s进行填充；返回指向s的指针

```
char*  strpbrk(const char *s, const char *t);
```

如果字符串s中有字符出现在字符串t中，则返回第一次出现的字符在s中位置的指针

```
char*  strrchr(const char *s, int c);
```

返回字符 c 在字符串 s 中最后一次出现的位置的指针

```
size_t  strspn(const char *s, const char *t);
```

返回字符串 s 从开头开始包含在字符串 t 中的连续字符的长度

```
char*  strstr(const char *s, const char *t);
```

返回字符串 t 在字符串 s 中出现的位置的指针

<ctime>

头文件 <ctime> 中定义了两个宏、4 种类型和若干表示、操作日历时间及本地时间的函数。类型 clock_t 和 time_t 是用于表示时间的算术结构，结构 tm 包含了将日历时间分解成秒（tm_sec）、分（tm_min）、时（tm_hour）、日（tm_mday）、自一月开始的月（tm_mon）、自 1990 年开始计算的年（tm_year）、从周日开始计算的日期（tm_wday）、从 1 月 1 日开始计算的日期（tm_yday）和夏令时标志（tm_isdst）。结构中这些值的顺序与系统相关。相关的宏如下：

```
CLOCKS_PER_SEC
```

clock 函数返回的每秒的时钟滴答数

```
NULL
```

表示二进制 0 的整数

相关计算的函数原型和简要介绍如下：

```
char*  asctime(const struct tm *timeptr);
```

将时间转换成字符串表示，并返回指向字符串的指针

```
clock_t  clock(void);
```

返回当前的处理器时间

```
char*  ctime(const time_t *timer);
```

将时间转换成字符串表示，并返回指向字符串的指针

```
double  difftime(time_t time1, time_t time0);
```

计算两个日历时间的差

```
struct  tm* gmtime(const time_t *timer);
```

返回一个指向使用协调世界时表示时间的结构指针

```
struct  tm* localtime(const time_t *timer);
```

返回一个指向日历时间的结构指针

```
time_t  mktime(struct tm *timeptr);
```

将分解的时间值转换成一个日历时间值

```
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);
```

将时间转换成格式化的多字节字符序列

<iostream>

头文件 <iostream> 包含了许多用于标准输入、输出的函数。我们以最常用的形式列出了其中的一部分：

istream 函数：

```
istream& operator >>
```

输入（标准）数据

```
int gcount();
```

返回上一次输入操作释放的字符数目

```
int get(char ch);
```

从输入流中释放一个字符

```
getline(c_string var, int max, [char delimiter]);
```

读取 max-1 个字符或遇到分隔符时停止，'\n' 是默认的分隔符

```
ignore();
```

释放一个字符并将其从输入流中去除

```
char peek();
```

返回输入流中下一个字符的值，但将其留在输入流中（不释放）

```
putback(ch);
```

将上一个字符放回输入流中

ostream 函数：

```
ostream functions:
```

```
ostream& operator<<
```

输出（插入）操作符

```
flush();
```

冲洗输出缓冲区

```
put(ch char);
```

输出输入流中的字符

ASCII 字符编码

下面的表中包含了 128 个 ASCII 字符和它们对应的整数值及二进制表示。对应于整数 0～31 的字符对于计算机系统有特殊的含义。例如，字符 BEL 对应于整数 7，它将使键盘发出响铃声。

字符从低到高排列，它有一些有趣的特征。可以看到，数字字符小于大写字符，大写字符小于小写字符。此外，特殊字符并不是在一起的，有些在数字字符之前，有些在数字字符之后，还有一些在大写字符和小写字符之间。

字符	等价整数	等价二进制表示
NUL（二进制 0）	0	00000000
SOH（Start of Header，开头标题符）	1	00000001
STX（Start of Text，正文开始符）	2	00000010
ETX（End of Text，正文结束符）	3	00000011
EOT（End of Transmission，传输结束符）	4	00000100
ENQ（enquiry，询问符）	5	00000101
ACK（Acknowledge，确认符）	6	00000110
BEL（Bell，响铃符）	7	00000111
BS（Backspace，退格）	8	00001000
HT（Horizontal Tab，水平制表符）	9	00001001
LF（Line Feed 或 New Line，换行符）	10	00001010
VT（Vertical Tabulation，垂直制表符）	11	00001011
FF（Form Feed，换页）	12	00001100
CR（Carriage Return，回车）	13	00001101
SO（Shift Out，移出）	14	00001110
SI（Shift In，移入）	15	00001111
DLE（Data Link Escape，数据传送换码字符）	16	00010000
DC1（Device Control 1，设备控制 1）	17	00010001
DC2（Device Control 2，设备控制 2）	18	00010010
DC3（Device Control 3，设备控制 3）	19	00010011
DC4（Device Control 4–Stop，设备控制 4）	20	00010100
NAK（Negative Acknowledge，否定应答）	21	00010101
SYN（Synchronization，同步）	22	00010110
ETB（End of Text Block，传输块结束符）	23	00010111
CAN（Cancel，取消字元）	24	00011000
EM（End of Medium，媒介结束符）	25	00011001
SUB（Substitute，替代符）	26	00011010
ESC（Escape，转义符）	27	00011011

（续）

字符	等价整数	等价二进制表示
FS（File Separator，文件分隔符）	28	00011100
GS（Group Separator，组分隔符）	29	00011101
RS（Record Separator，记录分隔符）	30	00011110
US（Unit Separator，单元分隔符）	31	00011111
SP（Space，空格）	32	00100000
!	33	00100001
"	34	00100010
#	35	00100011
$	36	00100100
%	37	00100101
&	38	00100110
'（单引号）	39	00100111
(	40	00101000
)	41	00101001
*	42	00101010
+	43	00101011
,（逗号）	44	00101100
-（连字符）	45	00101101
.（句点）	46	00101110
/	47	00101111
0	48	00110000
1	49	00110001
2	50	00110010
3	51	00110011
4	52	00110100
5	53	00110101
6	54	00110110
7	55	00110111
8	56	00111000
9	57	00111001
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111
@	64	01000000
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101

（续）

字符	等价整数	等价二进制表示
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010
[	91	01011011
\	92	01011100
]	93	01011101
^（Circumflex，抑扬符）	94	01011110
_（Underscore，下划线）	95	01011111
`（开单引号）	96	01100000
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111

（续）

字符	等价整数	等价二进制表示
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010
{	123	01111011
Ω	124	01111100
}	125	01111101
~	126	01111110
DEL（Delete/Ruhour，删除）	127	11111111

使用 MATLAB 从 ASCII 文件中绘制数据点

为了理解工程问题和问题的解决方案，将相关的数值信息进行可视化很重要。因此，从数据文件中轻松地获取简单的 xy 坐标图的能力在解决工程问题的过程中很重要。

在本附录中，我们使用一个简单的 C++ 程序生成一个数据文件，然后展示如何使用 MATLAB 来得到数据图。在本附录和正文章节中我们选择 MATLAB（MATrix LABoratory）来生成图示，因为在交互式数值计算、数据分析和图像处理中，它都是一个极其通用的软件环境。有关生成不同类型的图示和附件选项的延伸讨论可以参见《Engineering Problem Solving with MATLAB》的第 7 章，该书由 D. M. Etter 著，Prentice Hall 于 1993 年出版。

在下面的例子中，我们使用一个 C++ 程序生成一个 ASCII（American Standard Code for Information Interchange）数据文件，然后使用 MATLAB 绘制出相关信息。ASCII 数据文件也可以使用字处理器生成，然后利用 MATLAB 通过相同的步骤绘制信息。如果数据文件是由字处理器生成的，在选择保存文件选项时必须注意保存成一个文本文件，而不是一个字处理器文件。

下面的程序生成了包含 100 行信息的数据文件。每行包含相应的时间和阻尼正弦函数值。

$$f(t) = e^{-t}\sin(2\pi t)$$

其中 $t = 0.0, 0.1, 0.2, \cdots, 9.9$ 秒。

生成数据文件的 C++ 程序

```
/*------------------------------------------------------------*/
/*  Program App_C                                             */
/*                                                            */
/*  This program generates a data file of values              */
/*  from a damped sine function.                              */

#include<iostream>
#include <fstream>
#include <cmath>
using namespace std;
const double PI = 3.141592;

int main()
{
  /*  Define objects.  */
  double t, f;
  ifstream data;

  /*  Generate data file.  */
  data.open("demo.dat");
  for (t= 0.0; t<10.00; t+=
  {
```

```
    t = 0.1*(k-1);
    f = exp(-t)*sin(2*PI*t);
    dsine << t << f << endl;
  }

  /*  Close data file and exit program. */
  dsine.close();
  return 0;
}
/*------------------------------------------------------------------*/
```

由 C++ 程序生成的 ASCII 数据文件

在由程序生成的 ASCII 文件中，每行包含两个数字。前面部分行和最后部分行的信息如下：

```
0.0  0.000
0.1  0.532
0.2  0.779
...
9.9  0.000
```

使用 MATLAB 生成图示

为了使用 MATLAB 生成上述信息的图示，我们需要两条语句。第一条语句将数据文件载入 MATLAB 工作区中，第二条语句生成 *xy* 坐标图示：

```
>>load dsine.dat
>>plot(dsine(:,1),dsine(:,2))
```

这些步骤生成了图 C.1 中所示图像。

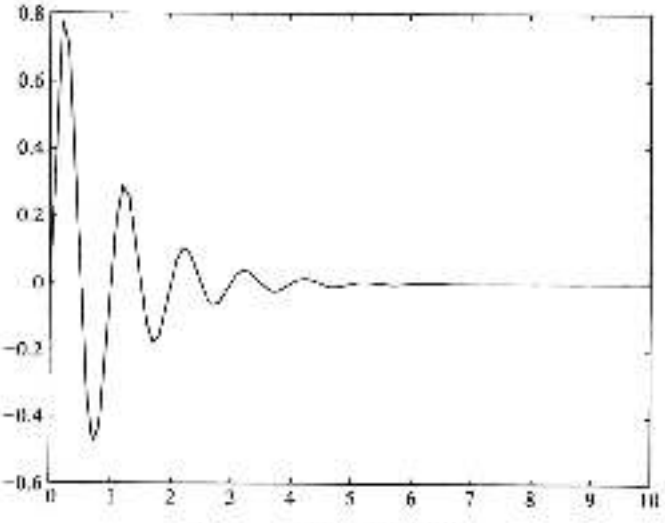


图 C.1　阻尼正弦函数图像

在图示中标识出信息很重要。我们还可以增加语句来为图示增加标题、坐标轴标识和背景网格：

```
>>load dsine.dat
>>plot(dsine(:,1),dsine(:,2));
>>title('Damped Sine Function');
>>xlabel('Time, s'), ylabel('f(t)'), grid
```

图 C.2 中增加了这些标签标识。

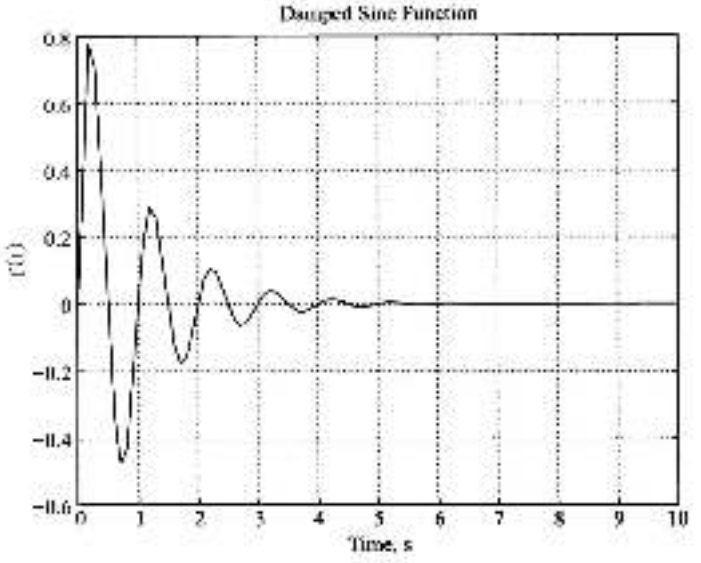


图 C.2 增强的阻尼正弦函数图像

练习答案

第 1 章

第 12 页

1. $921_{10} = 1110011001_2$
2. $8_{10} = 1000_2$
3. $100_{10} = 1100100_2$
4. $100_8 = 64_{10}$
5. $247_8 = 167_{10}$
6. $16_8 = 14_{10}$
7. $100_2 = 4_8$
8. $3716_8 = 011111001110_2$
9. $110100111_2 = 423_{10}$
10. $221_8 = 125_8$

第 13 页

1. $921_{10} = 399_{16}$
2. $8_{10} = 8_{16}$
3. $100_{10} = 64_{16}$
4. $1C0_{16} = 448_{10}$
5. $29F_{16} = 670_{10}$
6. $16_8 = 22_{10}$
7. $10010011_2 = 93_{16}$
8. $3A1B_{16} = 0011101000011011_2$
9. $110100111_2 = 423_{10}$
10. $261_8 = B1_{16}$

第 15 页

1. 00110001
2. 00000010
3. 10111011

第 2 章

第 26 页

1. 合法
2. 合法
3. 合法
4. 合法
5. 合法
6. 不合法（特殊字符 -）
7. 合法
8. 不合法（特殊字符 *）
9. 合法
10. 不合法（关键字）
11. 不合法（特殊字符 @）
12. 不合法（特殊字符 $）
13. 合法
14. 不合法（关键字）
15. 不合法（特殊字符（和））
16. 合法
17. 合法
18. 不合法（特殊字符 .）
19. 合法
20. 合法
21. 不合法（特殊字符 /）

第 27 页

1. 3.5004×10^1 4 位
2. 4.2×10^{-4} 1 位
3. -5.0×10^4 0 位
4. $3.157\,23 \times 10^0$ 5 位
5. -9.997×10^{-7} 3 位
6. $1.000\,002\,8 \times 10^7$ 7 位
7. 0.000 010 3
8. −105 000
9. −3 552 000
10. 0.000 667
11. 0.09

12. −0.022

第 31 页

1. const double LightSpeed = 2.99792e08;
2. const double ChargeE = 1.602177e-19;
3. const double N_A = 6.022e23;
4. const double G_mss = 9.8;
5. const double G_fss = 32;
6. const double EarthMass = 5.98e24;
7. const double MoonRadius = 1.74e6;
8. const char UnitLength = 'm';
9. const char UnitTime = 's';

第 37 页

1. 6
2. 4.5
3. 计算结果是 3.1，赋给整数 a 的是 3
4. 计算结果是 3.0
5. pl-> 0.0 / 0.0
6. x-> 2.0
   y-> 4.3
   pl-> 2.0 / 4.3

第 39 页

```
const double G_mss = 9.80665;
```

1. distance = x + v*t + a*t*t;
2. tension = (2*a1*a2)/(a1+a2)*G_mss;
3. p2 = p1*(p*v*(a2*a2-a1*a1))/(2*a1*a1);
4. centripetal = $\frac{4\pi^2 r}{T^2}$
5. potential energy = $\frac{GMm}{r}$
6. change = $GMm\left(\frac{1}{R_1}-\frac{1}{R_2+h}\right)$

第 42 页

1. z [8] x [3] y [4]
2. z [12] x [3] y [4]
3. x [6] y [4] z [?]
4. y [0] x [2] z [?]

第 45 页

1. 输出：
   150 12.368
2. 输出：
   15012.368
3. 输出：
   150
   12.368
4. 输出：
   150
   12
5. 输出：
   150,12.4
6. 输出：
   150,12.368
7. 输出：
   150
   12
8. 输出：
   12.368
   150

第 56 页

1. −3
2. −2
3. 0.125
4. 3.16228
5. 25
6. 11
7. −1
8. −32

第 57 页

1. velocity = sqrt(pow(v0,2) + 2*a*(x − x0));
2. length = pow(len − pow(v/c,2),1.0/k);
3. center = (38.1972*(pow(r,3) − pow(s,3))*sin(a))/((pow(r,2) − pow(s,2))*a);
4. frequency = $\frac{1}{\sqrt{2*\pi*\frac{c}{l}}}$

5. $range = \frac{v_0^2}{g} * \sin(2\theta)$

6. $v = \sqrt{\frac{2gk}{\sqrt[3]{1+\frac{I}{mr^2}}}}$

第 58 页

1. cothX = cosh(x)/sinh(x);
2. sechX = 1.0/cosh(x);
3. cschX = 1.0/sinh(x);
4. acothX = 0.5*log((1 + 1/x)/(1 - 1/x));
5. acoshX = log(x + sqrt(pow(x,2)-1));
6. acschX = log(1/x + sqrt(1 + pow(x,2))/fabs(x));

第 3 章

第 74 页

1. true
2. true
3. true
4. false
5. true
6. true
7. true
8. false

第 79 页

1.
```
if(time > 15.0)
   ++time;
```

2.
```
if(sqrt(poly) < 0.5)
   cout << poly;
```

3.
```
if(abs(volt1 - volt2) > 10.0)
   cout << volt1 << ' ' << volt2;
```

4.
```
if(den < 0.05)
{
  result=0;
}
else
{
  result = num/den;
}
```

5.
```
if(log(x) >= 3)
{
  a=a+6;
  count++;
}
```

6.
```
if(dist < 50.0 && time > 10.0)
{
  time-=2;
}
else
{
  time-=2.5;
}
```

7.
```
if(dist >= 100.0)
{
  time+=2;
}
else if(dist >=50)
{
  time++;
}
else
{
  time+=2.5;
}
```

第 81 页

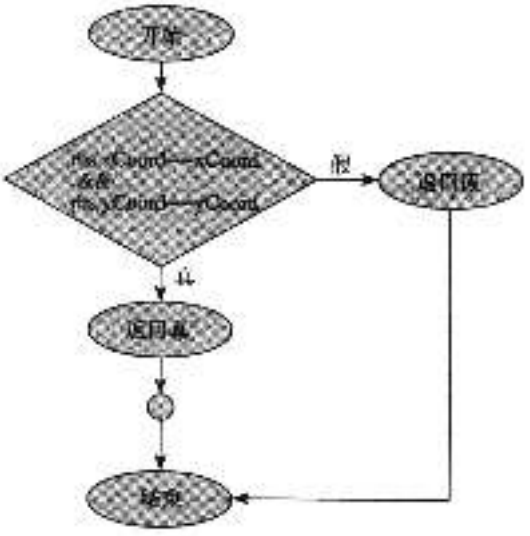
1. 75.92　89.35　111.25　109.92
2. 0.69　1.6　1.71　1.87
3. 如图 3.10 所示，右 5 个与 110 华氏度对应的时刻值。这 5 个时刻值为：1.71　2.84　3.39　4.42　5.33

第 87 页

```
switch(rank)
{
  case 1:
  case 2:
   cout << "Lower division" << endl;
   break;
  case 3:
  case 4:
   cout << "Upper division" << endl;
   break;
  case 5:
   cout << "Graduate student" << endl;
   break;
  default:
   cout << "Invalid rank" << endl;
}
```

第 97 页

1.



2. Set *diffX* to *rhs.xCoord - xCoord*
Set *diffY* to *rhs.yCoord - yCoord*
Set distance to $\sqrt{diffX^2 + diffY^2}$
Return distance

第 4 章

第 107 页

1. 5
2. 无限循环
3. -2
4. 3

第 110 页

1. 18
2. 18
3. 17
4. 0
5. 无限循环
6. 15

第 5 章

第 153 页

1. i [1] j [0] badbit:0 failbit:1 eofbit:0 goodbit:0
2. x [1] y [0] badbit:0 failbit:1 eofbit:0 goodbit:0
3. ch1 [1] ch2 [,] badbit:0 failbit:0 eofbit:0 goodbit:1
4. x [1] ch [.] y [2.3] badbit:0 failbit:0 eofbit:0 goodbit:1

第 6 章

第 178 页

1.

形参	a	b	c
	25	5	-5
函数参数	x	sqrt(x)	x-30
	25	5	-5

2. total = 2

第 183 页

1. 今次

调用前：x [1] y [3]

调用后：x [3] y [1]

2. 不合法：不能把常量传递给引用参数
3. 不合法：不能把表达式 (y+5) 传递给引用参数
4. 错误：在你自己的系统上测试一下
5. 输出：

0

2

第 203 页

1. 不可以
2. 是。对于 Point 类已经定义了默认的构造函数
3. 不是。xCoord 是私有的
4. 是。getY( ) 是公共方法
5. 是。对于 Point 类操作符已经重载了，返回 p1 和 p2 之间的距离
6. 方法实现：

```
void Point::setXY(double xVal,
double yVal)
{
   xCoord = xVal;
   yCoord = yVal;
}
```

第 7 章

第 234 页

1. x

5	4	3	0	0	0	0	0	0	0

2. letters

a	b	c

3. z

?	-5.5	5.5	5.5

4. time

-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4

5. arr[0] is 0
arr[1] is 3
arr[2] is 6
arr[3] is 9
arr[4] is 12

第 237 页

1. 输出：
3 8
15 21
30 41

2. 输出：
8 30

第 241 页

1. 9.8
2. 9.8
3. 3.2
4. 1.5

第 249 页

1. 9.0
2. 6.0
3. 5.36
4. 2.32
5. 2.5
6. 5.75

第 264 页

1. The Cheese
2. The mice, Cheese
3. The mice, Sniff and Scurry, had only simple brains.

第 8 章

第 287 页

1.

1	0
2	0
3	0

n

2.

1	2	3
4	5	6
0	0	0

h

3.

1	2	3	4
0	3	0	6
8	3	-6	-1

c

4.

1	2	3	0
3	0	6	3
-6	-1	0	0

d

第 289 页

1.

1
4
6

2.

5	2
-2	3
0	0
0	0
0	0
0	0

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	0
1	2	3	4	0
2	3	4	5	0
3	4	5	6	0
4	5	6	7	0

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

第 292 页

1. 9
2. 4
3. 6
4. 3

第 294 页

1. 13
2. 2
3. 18
4. 22

第 303 页

1. r [3] c [5]

?	?	?	?	?
?	?	?	?	?
?	?	?	?	?

2. series

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

3.
```
double maxColumnVal(vector<vector
<double>> v, int colNum)
{
   //Preconditions:  The vector v has
at least colNum+1
   //                columns of data
   double largeVal = v[0][0];
   int rowSize = v.size( );
   for(int i=0; i<rowSize; ++i)
   {
     if(v[i][colNum]> largeVal)
     {
       largeVal = v[i][colNum];
     }
   }
   return largeVal;
}
```

4.
```
double maxRowVal(vector<vector<double
>> v, int rowNum)
{
   //Preconditions: The vector v has
at least rowNum+1
   //          rows of data
   double smallVal = v[0][0];
   int colSize = v[rowNum].size();
   for(int i = 0; i<colSize; ++i)
   {
     if(v[rowNum][i]< smallVal)
     {
       smallVal = v[rowNum][i];
     }
   }
   return smallVal;
}
```

第 307 页

1. -8
2.

5	-1	3
3	3	2

3.

-5	9
1	-5
-6	6

4.

-2	-2	4
7	-9	10

5.

8	-24	32
12	20	-24
14	-18	20

第 311 页

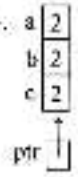
1. x = 2, y = 1
2. x = 3, y = -1, z = 2

第 9 章

第 326 页

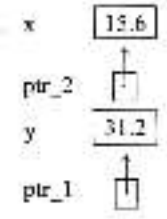
1. a [1]
   b [2]
   ptr [ ]
2. a [2]
   b [2]
   ptr [ ]
3. a [1]
   b [5]
   c [1]
   ptr [ ]

4.

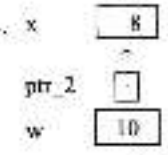


第 329 页

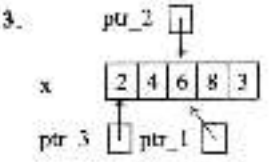
1.



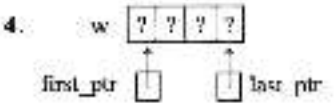
2.



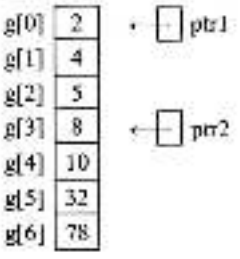
3.



4.



第 330 页



1. 2
2. 4
3. 3
4. 32
5. 2
6. 8
7. 4
8. 32

第 336 页

1. 合法
2. 不合法（ptr 是常量）
3. 不合法（不能使用 cpu 修改对象）
4. 不合法（不能使用 cpu 修改对象）
5. 合法
6. 合法

第 345 页

1. 10 10
2. 0 1 2 3
3. 0 2

第 10 章

第 371 页

1.
```
pixel pixel::operator/(pixel p)
{
  pixel temp;
  temp.red = red/p1.red;
  temp.green = green/p1.green;
  temp.blue  = blue/p1.blue;
  return temp;
}
```

2.
```
bool pixel::operator==(pixel p1)
{
  if( red==p1.red &&
      green==p1.green&&
      blue==p1.blue )
    return true;
  return false;
}
```

第 372 页

1.
```
pixel operator+(pixel p1, pixel p2)
{
  pixel temp;
  temp.red = p1.red+p2.red;
  temp.green = p1.green+p2.green;
  temp.blue =p1.blue+p2.blue;
  return temp;
}
```

2.
```
pixel operator/(pixel p1, pixel p2)

  pixel temp;
  temp.red = p1.red/p2.red;
```

```
    temp.green = p1.green/p2.green;
    temp.blue =p1.blue/p2.blue;
    return temp;
}
```

3.
```
pixel operator-(pixel p1, pixel p2)
{
    pixel temp;
    temp.red = p1.red-p2.red;
    temp.green = p1.green-p2.green;
    temp.blue = p1.blue-p2.blue;
    return temp;
}
```

第 392 页

1. 0
2. 0
3. 5
4. 0xbffffb74, 0, 0

第 410 页

1. Fixed point at: (1,2)
   Width: 4
   Height: 4
   Depth: 4 Fixed point at: (1,2)
   Width: 4
   Height: 4
   Depth: 4
2. Fixed point at: (1,2)
   Width: 4
   Height: 4
3. Fixed point at: (0,0)
   Width: 5
   Height: 5
   Fixed point at: (0,0)
   Width: 5
   Height: 5

参考文献

[1] "10 Outstanding Achievements, 1964–1989." National Academy of Engineering, Washington, DC, 1989.

[2] Etter, D. M. *Structured FORTRAN 77 for Engineers and Scientists*. 4th ed. Benjamin/Cummings, Redwood City, CA, 1993.

[3] "The Federal High Performance Computing Program." Executive Office of the President, Office of Science and Technology Policy, Washington, DC, September 8, 1989.

[4] Etter, D. M., and J. Bordogna. "Engineering Education for the 21st Century." IEEE International Conference on Acoustics, Speech, and Signal Processing, April 1994.

[5] Fairley, R. *Software Engineering Concepts*. McGraw-Hill, New York, 1985.

[6] Etter, D. M. *Engineering Problem Solving with MATLAB*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[7] Plauger, P. J. *The Standard C Library*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[8] Jones, E. R., and R. L. Childers. *Contemporary College Physics*. Addison-Wesley, Reading, MA, 1990.

[9] Etter, D. M. *FORTRAN 77 with Numerical Methods for Engineers and Scientists*. Benjamin/Cummings, Redwood City, CA, 1992.

[10] Spanier, Jerome and Keith B. Oldham. *An Atlas of Functions*. Hemisphere Publishing Corporation, 1987.

[11] Edwards, Jr., C. H., and D. E. Penney. *Calculus and Analytic Geometry*. 3d ed. Prentice Hall, Englewood Cliffs, NJ, 1990.

[12] Master, G. M. *Introduction to Environmental Engineering and Science*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[13] Gille, J. C., and J. M. Russell, III. "The Limb Infrared Monitor of the Stratosphere; Experiment Description, Performance, and Results," Journal of Geophysical Research, Vol. 89, No. D4, pp. 5125–5140, June 30, 1984.

[14] Roberts, Richard A. *An Introduction to Applied Probability*. Addison-Wesley, Reading, MA, 1992.

[15] Richardson, M. *College Algebra*, 3d ed. Prentice Hall, Englewood Cliffs, NJ, 1966.

[16] Kahaner, D., Cleve Moler, and Stephen Nash. *Numerical Methods and Software*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[17] Ralston Anthony and Edwin D. Reilly, eds. *Encyclopedia of Computer Science*, 3d ed. Van Nostrand Reinhold Publishing Company, New York, NY, 1993.

[18] Kreyszig, E. *Advanced Engineering Mathematics*. John Wiley & Sons, New York, 1979.

[19] Wirth, Niklaus. *Algorithms + Data Structures = Programs*. Prentice Hall, Englewood Cliffs, NJ, 1976.

[20] Stroustrup, Bjarne. *C++ Programming Language*. Addison Wesley, 1997.

[21] Wampler, Bruce E. *The Essence of Object Oriented Programming with Java and UML*. Addison Wesley, 2002.

[22] Kolman, Busby, Ross. *Discrete Mathematical Structures*, Prentice Hall, 2000.

[23] www.wikipedia.com