高等院校课程设计案例精编

赠送光盘中附有完整的案例源代码

数据结构课程设计案例精编

（用C/C++描述）

李建学　李光元　吴春芳　编著

• 线性表的应用案例 • 农夫过河问题 • 深度优先搜索算法 •
• 广度优先搜索算法 • 汉诺塔问题 • 迷宫问题 •
• 八皇后问题 • 二叉搜索树应用案例 • 图类的实现与应用 •
• 霍夫曼编码解码器的实现 • 算法计时器的实现 •
• STL组件的综合应用 • 词典检索系统 •

赠送光盘

清华大学出版社

高等院校课程设计案例精编

数据结构课程设计案例精编
(用 C/C++描述)

李建学　李光元　吴春芳　编著

清华大学出版社
北　京

内 容 简 介

本书是数据结构案例教程，以软件重用为指导思想，以 STL 库中实现的数据结构(容器)为参照，重构了数据结构的视角和知识体系，并突出实战性和应用性。

本书具体内容安排如下：第一部分是理论基础，包括绪论和第 1、2、3、4 章，介绍数据结构和算法的基础知识，C++语言的关键特征以及支撑 STL 设计的核心理念和机制。第二部分是基础数据结构，从第 5 章到第 9 章，涵盖了基础的数据结构，并包含丰富的例子。第三部分(即第 10 章)是综合应用，包括 4 个大的案例，是对前面章节所讲的基础数据结构的综合应用。本书附带的光盘中包含经过主流 C++编译器编译通过的所有程序的源代码及编译后生成的可执行程序和第三方软件。

本书将 C++泛型编程知识与数据结构知识紧密地结合在了一起，是国内比较罕见的、有技术深度的、符合现代发展方向的优秀教材/教辅读物。

本书适合于在校信息科学与技术类学生作为课程设计指导用书，亦可随数据结构课程同步学习，也非常适合于工作中的程序员以更加实践化的角度温习和应用数据结构。

图书在版编目(CIP)数据

数据结构课程设计案例精编(用 C/C++描述)/李建学，李光元，吴春芳编著. —北京：清华大学出版社，2007.2
(高等院校课程设计案例精编)
ISBN 978-7-302-14536-3

Ⅰ. 数…　Ⅱ.①李…　②李…　③吴…　Ⅲ.数据结构—课程设计—高等学校—教学参考资料　Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字(2007)第 004962 号

责任编辑：李春明　宋延清
封面设计：山鹰工作室
版式设计：杨玉兰
责任印制：李红英
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
c-service@tup.tsinghua.edu.cn
社 总 机：010-62770175　**邮购热线**：010-62786544
投稿咨询：010-62772015　**客户服务**：010-62776969
印 刷 者：北京国马印刷厂
装 订 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185×260　**印　张**：27　**字　数**：652 千字
附光盘 1 张
版　　次：2007 年 2 月第 1 版　　**印　　次**：2007 年 2 月第1 次印刷
印　　数：1～5000
定　　价：45.00 元

前　言

信息技术正在渗透到人类社会生产、生活的各个方面，作为信息技术灵魂的软件，其规模持续膨胀，在社会的正常运行中扮演着日益重要的角色。而软件开发的节奏也越来越快，应对这一挑战的不二法门，在于重用(reuse)。从汇编语言、C 语言，到 C++、Java 的流行，是基础设计方法重用程度的提高：从指令语句到函数，从对象到构件，支持重用的语言机制日渐成熟。从 20 世纪 60 年代到现在，软件工程持续演进，且引起大家的普遍重视，也是为了提高软件开发的重用性：能重用优秀的、以往开发好的可靠模块(库，构件)。使新开发的功能也能很好地模块化，是软件工程努力的一个重要目标。要成长为一个专业的程序员，在入门开始就对重用性和软件工程有清醒的认识，是很有必要的。

在软件开发领域中，从事数据库方向的人员，可以不甚了解硬件体系结构；从事计算机图形学的人员，可以不懂 SQL 语言；做处理器设计的人员，也可以不了解数据仓库。但是他们都必须要掌握数据结构和算法，因为这是程序的灵魂。各种基础的数据结构，是程序中的基础构件。在越来越重视重用性的今天，一种设计良好的、健壮的、高效的基础库的出现也是必然的。

C 语言和它的标准库本身，只提供了数组和结构体这样的基础结构，及过程化的几种排序算法，远不能达到基本重用库的规模。标准 C++语言却应时代呼声，产生了这样的可重用的数据结构库——在 C++语言里被称作容器，也提供了大量普遍适用的算法。C++标准库的重要组成部分：标准模板库(Standard Template Library，STL)则提供了完整的实现。

本书正是在这样的背景下，以软件重用为指导思想，以 STL 库中实现的数据结构(容器)为参照，重构了数据结构的视角和知识体系。与高校现行的数据结构教材相比，本书最大的特色，就是实战性强、应用性强。不得不看到，许多学生在学完数据结构这门课程后，懂得了二叉树、图结构的性质和应用，但在实际实践中，自己去实现这些结构时，却又勉为其难了。从应用的角度看，这样的教学不算是成功的。因此我们策划了一本新的案例教程——就是您手头上的这本书。

本书从 STL 出发，从各种数据结构的应用角度出发，给读者讲解这些结构和在 C++标准库中的实现，而不纠缠于各种细节。对程序设计和软件开发而言，STL 是程序员手中的金刚钻，可以节省大量时间，去做好“瓷器活儿”；对于编程技术而言，STL 是智囊和武库器，与编程工作最有直接关系的数据结构和算法，STL 都有实现，并且充分优化，效率最优。此外，STL 的设计理念，更具有可学习与可借鉴性，STL 中对象的耦合性低，重用性高，各种构件独立设计，又能够灵活地互相组合。从应用角度看，任何一位程序员都不该舍弃现成的工具和方案，来重新造一个轮子(那是违背时代潮流和软件工程原则的)。我们编写这本书，就是为搭建攀上巨人肩膀的阶梯。

本书的一大特色，是对图结构库的处理，STL 库涵盖了数据结构中所有的线性结构和树结构(以 map 和 set 方式)，但并没有覆盖图部分。为此，我们引入了与 STL 设计风格完

全一致同时具有工业强度、开放源码的 Boost Graph Library(BGL)图库，BGL 库在 C++标准化委员会中，是具有半官方地位的优秀库。这部分的内容，在国内现已发行的同类图书中，尚为解见。读者通过阅读本书第 9 章，相信必有耳目一新的感觉。

本书的另一特色是对编程规范自始至终的强调与遵守，通过严格的规范约束，塑造职业作风。规范本身与语言已经融为一体，在职业程序员圈子里，已经形成一种默认的规则。希望读者对这些规范也能有足够的重视。

本书的结构包括三个部分。第一部分：理论基础，即绪论和第 1、2、3、4 章，讲解数据结构和算法的基础知识、C++语言的关键特征以及支撑 STL 设计的核心理念和机制。绪论部分给出了针对全书提纲挈领的分析，对于有 C++面向对象经验的读者，我们也建议仔细阅读第 3、4 章，因为泛型设计作为现代 C++的设计方式，与经典的面向对象设计有很大的不同。第二部分：基础数据结构，从第 5 章到第 9 章，涵盖了基础的数据结构，并包含丰富的例子。第三部分：综合应用(第 10 章)，由 4 个大的案例构成，是对前面章节所讲的基础结构的综合应用。

本书附带的光盘中包含有经过主流 C++编译器编译通过的所有程序的源代码、已编译的可执行文件和第三方软件。

本书适合于在校信息科学与技术类学生作为课程设计指导用书，亦可随数据结构课程同步学习，以增强实践能力，也非常适合于工作中的程序员以更加实践化的角度温习和应用数据结构。

作为预备知识，学习者应具有一定的 C/C++程序设计的基础。但为了降低读者的门槛，在第一部分中，我们择要讲解了 C++语言特征，重点讲述了泛型设计。囿于本书的定位和篇幅，不可能十分详细地介绍 C++，读者对该语言方面的问题，建议参考本书最后参考文献给出的 C++教程类的世界级经典教材。

本书的 3 位主要作者都是有丰富实践经验且工作在一线的程序员，在本书编写过程中，尽力向读者介绍编程语言的最新进展和成果，并从学习者的学习曲线考虑，合理组织，降低学习曲线的陡峭度。

李光元负责编写本书的第 2 章、第 6 章、第 7 章内容，吴春芳负责编写本书的第 5 章内容(清华大学出版社的宋延清负责编写本书的第 3 章内容)，李建学负责第 1 章、第 4 章、第 8 至第 10 章内容及全书的统稿工作，其他章节及程序开发由余健、韩阳、黄晓燕、贾音、郭刚、张永宝、李享、陈凯共同完成。

在写作过程中，作者们反复斟酌，从章节顺序到程序变量命名、注释文档，都做了精心的安排。

尽管本书经过了反复的讨论和推敲，但限于水平，难免有诸多不妥之处，希望读者批评指正，以便再版时改进。可以发电子邮件到 ChiensyueLee@gmail.com 与作者联系。

编　者

目　录

第一部分　理论基础

第二部分 基础数据结构

第三部分 综合应用

第一部分　理论基础

绪　　言

——致成长中的程序员们

数据结构与算法是 IT 专业人士必须掌握的一门核心课程。学好它，在软件设计领域登堂入室，为成为专业程序员铺平道路，是 IT 学子的深切愿望。这本书从程序员的视角，以程序实践为基础，用标准 C++提供的平台作为工具，展示了数据结构与算法的具有良好重用性的实现方法，给出了应用这些数据结构和算法的丰富案例。相信本书会为您成长为专业软件开发人员助一臂之力。

在开篇之前，我们先理清一些基本问题。

一、为什么要使用 C++?

在学习和实践数据结构与算法时，首先遇到的问题就是用什么语言来描述和实现。

作为一门传统的基础课程，数据结构所用的描述语言从早期的教学语言 Pascal，到流行的 C 语言，以及最近的语言新宠 Java，不一而足。那么我们为什么要在本书中特意用 C++语言来描述和实现数据结构呢？

的确，C++难学，其广博的语法，以及语法背后的语义，语义背后的深层思维，以及深层思维背后的对象模型都会需要我们花费大量精力来学习；C++的难学，还在于它提供了 4 种不同而相辅相成的程序设计思维模式：基于过程(procedural-based)，基于对象(object-based)，面向对象(object-oriented)，泛型模式(generic paradigm)。

“在如此庞大复杂的机制下，万千使用者前仆后继的动力是：一旦学成，妙用无穷。”——侯捷语。

学习语言还只是新手跨入软件开发领域的第一步，单单学习语言本身是远远不够的，还要学习相关的程序库(例如 C++标准程序库)、相关的平台技术(如.NET)，说得更远一点，还要锻炼对目标问题的分析和归纳能力等。工作之前，技术路线由自己做主；有工作之后，绝大多数程序员将被公司的技术路线所左右。所以，在学生时代，趁现在还有时间，可以学一些自己感兴趣的内容。如果想搞软件开发(例如系统软件的开发)，学好 C++不会令我们失望。当我们进入 C++的门槛，然后经过一段黑暗之路的摸索，再沿着陡峭的阶梯攀登到光明的顶峰后，我们会充分地体味到“一览众山小”的感觉。

我们不再会简单地认为 C++是 C 的升级或扩充，但是绝对有很多人会认为 C++是面向对象的程序设计语言。实际上 C++是一门偏向于系统编程的通用编程语言。归纳起来，对 C++的特征可以这样描述：

- 是一个更好的 C。
- 支持数据抽象。
- 支持面向对象编程。
- 支持泛型编程。

软件设计的关键手段是抽象化，而软件开发的目的，除了为解决眼下的问题，提高软件的重用度更是现在非常看重的一个设计原则。这个原则始终是推动编程语言演化与发展的动力，也是各种软件工程方法努力的方向。C++的上述特征正是为了更好地抽象和实现代码重用。

在过去的十年或二十年中，抽象通常都与“对象”有些关系，这反映了支持面向对象技术的理论与实践的巨大发展。但“对象”概念忽略了程序员经常使用的“共同设计结构”，它们并非是面向对象的：模板、重载函数系列、泛型函数以及其他一些内容。泛型编程是一种基于发现高效算法的最抽象表示的编程方法。也就是说，以算法为起点并寻找能使其工作且有效率工作的最一般的必要条件集。令人惊讶的是，很多不同的算法都需要相同的必要条件集，并且这些必要条件有多种不同的实现方式。类似的事实在数学里也可以看到。大多数不同的定理都依赖于同一套公理，并且对于同样的公理存在多种不同的模型。抽象机制！泛型编程假定存在某些基本的法则在支配软件组件的行为，并且基于这些法则有可能设计出可互操作的模块，甚至还有可能使用此法则去指导我们的软件设计。使用 STL 和泛型编程标志着一个不同于一般 C++编程风格的新起点。

最有效的程序设计风格是联合使用这些技术，这也就是我们常说的“多范型程序设计(multi-paradigm programming)”。演化与发展之中的 C++语言的目标是使其成为一门更好的多范型程序设计语言。如 Bjarne Stroustrup[①]所言：“设计和程序设计的真正目标在于产生能够完成工作的最简单的解决方案，并且对该解决方案的表达要尽可能的清晰。……，这样的简单代码将会包含大量的相对简单的泛型程序设计，同时带有一些类继承层次结构，后者为需要运行期解决方案的领域提供服务。用今天的话来说，此即为‘多范型程序设计(multi-paradigm programming)’”。

我们在本书的理论基础部分，对 C++语言的基本要素进行了言简意赅的介绍，其基于对象和面向对象的特性已经有大量相关书籍和资料，我们书中该部分的目的在于为读者提供一个快速概览(第 2 章)。读者编程实践遇到问题可以查阅相关文档。

而使用模板的泛型程序设计，已经成为现代 C++语言的风格基础，C++98(1998 年标准化的 C++)提供的标准程序库中有 80%的成分是使用模板机制实现的 STL(Standard Template Library，标准模板库)。泛型程序设计的优势在于提供通用性和灵活性的同时，又保证了效率，这也是标准库中大量使用模板机制的原因。而国内现阶段教学并未对 C++的泛型程序设计给予足够的重视，书籍资料也不是很多。因此我们在理论基础部分(第 3、

注：① Bjarne Stroustrup，有“C++之父”的美誉，是 C++语言的设计者和第一位实现者。现任美国德州农工大学(Texas A&M University)计算机科学系首席教授，此前他一直担任 AT&T 实验室的大规模程序设计研究部门(自 2002 年底创建以来)的主管。

4章)不惜笔墨，比较深入地讲解了泛型设计与STL库。

二、为什么要使用STL?

我们是要学习数据结构与算法，不是在专修C++语言。上面说的那么多，与我们要学习的数据结构和算法课程又有什么关系呢?

数据结构的重点内容，在于设计能够高效存储大型数据集合的结构，针对不同的结构设计操作结构中数据元素的算法，以及根据对数据的操作需求，选用合适的数据结构。在实际的开发过程中，组织数据和选择操作于这些数据上的算法几乎同等重要。组织系列元素的方式将直接影响到诸如插入、删除、访问以及重新组织它们(如排序)等操作的时间复杂度。尽管数组能够方便地完成随机存取操作，但是往其中插入元素相当麻烦。而对于链表来说，插入与删除操作只需简单动作，并且存取任意元素都是线性时间复杂度。事实上，当程序对时间要求很苛刻时(如实时控制系统)，数据结构的选取尤为重要，甚至关乎程序开发的成败。

数据结构作为软件开发的基本要素，在早期的实践中，我们不断地重复实现一些诸如向量(vector)、链表(list)等常见的数据结构。这些代码都十分相似，只是为了适应不同数据的变化而在一些细节方面有不同的安排。可不可以实现一些基础数据结构来解决上述问题呢?C++语言既然具有那么强大的描述能力，为什么不用它去尝试呢?

于是STL就闪亮登场了。AT&T贝尔实验室和HP研究实验室的研究人员将泛型程序设计和面向对象程序设计的原理结合起来，创造了研究数据结构与算法的一套统一的方法，即STL，它现在是C++标准库的一部分。STL提供了实现与看待数据结构的新途径。它将(数据)结构(即组织数据的存储结构)抽象为容器，将之分为3类：序列容器、关联容器和适配器容器。通过使用模板和迭代器，STL库使得程序员能够将广泛的通用算法应用到各种容器类上。

本书的重点不是讨论STL，而是参照STL的设计结构，来设计数据结构。我们想达到两个目标：一是采用与STL容器类相同的接口定义并实现这些数据结构的类，使读者学到设计和实现容器的方法；二是介绍STL容器的基本用法，在综合案例部分，我们应用标准C++和STL库开发若干大的应用，展示如何高效使用STL的容器来提高程序设计开发的效率。

C++之父Bjarne先生对于如何学习语言、设计模式和数据结构如是说：“我并不喜欢根据特定的具名模式去思考，但我知道并且通常会使用这些在《设计模式》中描述的技术。……。除了明显的强大能力之外，我认为模式有两大弱点：它倾向于鼓励‘精致的专用术语’，这会阻碍新手的学习；如果没有具体的‘工具’支持，要想把一个思想广泛地传播到应用中是极其困难的。”

“例如，一个优秀的库本身携带了很多优秀的思想，并允许程序员(和设计者)直接利用这些思想在库中的实现品来工作。而模式只是对某个思想(或一系列互相关联的思想)的尽量一般性的描述，并刻意避免将这些思想作为库实现出来而招致的特殊性。这就导致了这些思想在传播上的问题，以及从代码中如何识别出模式的问题——特别是在代码被维护修改过之后。同样，要想从抽象层面上来理解一个模式也是非常困难的。在某个模式的抽

象描述之后的实例代码进入我的视野之前，我倾向于对自己的理解持保留态度。我见到过一些人，他们认为自己是在使用某个模式，而实际上做的却是该模式被设计用来避免的事情。这些都说明思想的传授可能会异常困难。”

“可以让模式更具有可利用性的方式之一是为某些特定的环境提供模式的库的实现。Andrei Alexandrescu 的书[②]和他的 Loki 库可以被看成一次寻求结合高灵活性和高效率(和手写代码一样高效)编程风格的尝试。而模板元编程在大多数情况下都符合这个描述，STL 亦然。为了从这种非常一般性的参数化中获益，设计(或编码)抉择必须从运行期转移到编译期，从而程序才更容易在时间或速度上得到优化。”

“我的目标是先教给新手最小的一套原则、技术以及语言设施，让他们可以先开始第一个实在的项目。基本上，我打算让那些想要成为职业 C++程序员的人由此起步。为了达到这个目的，我一开始的讲授要涵盖很多背景知识，包括数据结构、算法、图以及类设计等，这在传统的教学中是不会很早涉及的。这比我了解到的当前大多数的教学方式更加雄心勃勃。当然，我使用 C++作为编程语言，并且我会在讲授中涵盖 STL 的基础知识。”

C++难学易用，让我们就从使用 STL 库来开始吧！

三、编程的一些规范

好的编程规范能够带来许多相互关联的优点：

- 改善代码质量——鼓励开发人员一贯地正确行事，从而能够直接提高软件的质量和可维护性。
- 提高开发速度——开发人员不需要总是从一些基本原则出发进行决策。
- 增进团队精神——有助于减少在一些小事上不必要的争论，使团队成员更容易阅读和维护其他成员的代码。

好的编程规范反映了业界最久经考验的经验，它包含了以经验和对语言的深刻理解为基础的公认的惯用法。具体而言，这些规范应该牢固地建立在大量丰富的软件开发的基础之上。作为程序员，也应该尽早步入正轨，以同样的方式实施同样的过程，不断积累惯用法。平时不知道软件工程良好实践的(或者不习惯应用这些实践的)马虎程序员，在压力下会编出更加马虎、错误百出的代码。相反，养成良好习惯并经常按此习惯工作的程序员将保持自己的有条理性，能够快速提交高质量的代码。

作为面向实践的教程，我们在本书始终强调并贯彻着业界广泛认同的好的编程规范，对于这些规范的强调贯穿了本书全文。而在开篇之时，我们先给出一些最基本的规范，在开始编码之前，最好就能将它们熟记于心，这样你才能做到“胸中有丘壑”。

1. 命名规范

好的命名原则在软件开发中的重要性不容小觑，尤其在可重用代码中更是如此。好的命名应该是直观而容易理解的，在移入新环境或上下文中之后仍能保持这种清晰的特点，

注：② Andrei Alexandrescu, Modern C++ Design: Generic Programming and Design Patterns Applied 国内版本：《C++设计新思维：泛型编程与设计模式之应用》，华中科技大学出版社，《C++设计新思维(影印版)》，中国电力出版社

并且不易与其他组件产生名字冲突。下面一些规则请尽量遵守。

(1) 名字应本着清楚和简单的原则。首先要做到清楚，从而帮助(而不是混淆)对代码的理解；其次在清楚的前提下尽量简单，尽量用可发音的名字——可发音的名字更好读、更好懂，也更便于交流。

```
// 不好理解的名字
int rs;
int num;
// 比较一下
int returnStatus;
int alarmNumber;

// 不可发音的名字
class Ymdhms;    //年月日时分秒的英文缩写
// 可发音的名字
class Timestamp_T;
```

一些具体量化的做法为：可以将名字长度限制在 3 到 25 个字符之间，以保证名字做到“简单、清楚”。少于 3 个字符通常不够清楚(以 3 为界是因为 lhs、rhs 之类的名字已经足够清晰了)；大于 25 个字符则会嫌罗嗦而不够简洁。

(2) 命名时避免使用国际组织占用的格式以减少潜在的命名冲突，更可以防止阅读者误以为是国际组织提供的代码。已知的被占用的格式有：

- 单、双下划线开头——ISO C++、ANSI C 占用。
- 包含双下划线——ISO C++占用。
- E[0-9 A-Z]开头——ANSI C 占用。
- is[a-z]开头——ANSI C 占用。
- to[a-z]开头——ANSI C 占用。
- LC_开头——ANSI C 占用。
- SIG[_A-Z]开头——ANSI C 占用。
- str[a-z]开头——ANSI C 占用。
- _t 结尾——POSIX 占用。
- 其他国际组织占用的格式。

(3) 对不同类型的命名，注意字母大小写，这在业界也有普遍认可的规范。建议为：

- 宏名、常量名全部都只用大写字母，绝对不要使用普通单词或缩写词作为宏名，这样能有效地避免与其他类型的名字(主要是变量名)相冲突。
- 模板形式参数命名使用单词首字母大写形式, 如 InMixedCase。
- 类名、函数名和枚举变量的命名 **LikeThis**，为首字母大写；变量命名 **likeThis**，为除了首个单词而外，每个单词的首字母都大写；私有成员变量命名 **likeThis_**，或者 **m_likeThis**。
- 指针标识符命名建议以 p 开头或 Ptr 结尾，如 char* **pName** 或 char* **namePtr**;

(4) 注意长命名和短命名的合理使用：可以用完整的单词或词组命名(循环变量和遍历算子除外)，一个好名字比一段解释该名字含义的注释更好。应当避免简写，如 Iterator 比 Iter 好，短单词比被缩短的单词好。短名字的使用应当依照通用惯例，如 i、j 和 k 用于

循环控制变量，p、q 用于指针和迭代器。

(5) 各种类型命名时应注意词性的选择：类和对象名称应该用名词，变量名应该用名词，类的存取和查询成员函数名应是名词或形容词，如得到 container 长度的函数应叫 size()而不是 get_size()。应当选择表示目的的名字而不是表示实现的名字，比如定义一个已知设备列表 list<device>，命名为 known_devices 而不是 device_list。布尔变量和布尔函数应当是英文中的谓词短语(is_adj)，这样条件语句读起来就好像是英文中的条件句了，如 if(buffer->is_empty())。

2. 风格规范

在进行实际编码时，风格规范涉及从注释、代码块到类和函数域，以至于整个项目文件范围等方面形成的良好风格习惯。

(1) 注释：初学者往往注重完成特定的算法功能，而忽视了注释以及代码可读性、可维护性。注释应当是编码的一部分。没有注释，编码不算完整。写出好的注释如同写出好的代码一样，需要经验积累与素养。注释要清晰、简洁，并且有价值。对于注释，请尽量遵守下面的规范：

- 使用“//”，因为“/*... */”不支持嵌套注释。
- 长注释应和代码分处不同的行。
- 如果注释掉大段代码，请使用“#if 0... #endif”或“//”注释代码，不要使用“/*...*/”。用“/*...*/”作注释可能会导致嵌套注释，当被注释掉的代码块很大时更容易出现这种情况，这样可能导致注释掉的区域不是我们想要的范围。而“#if 0... #endif”方式可以嵌套，# if 0 是废掉代码，#if 1 是打开代码，非常方便。
- 确保所有注释(随代码)及时更新。一定要牢记注释是编码的一部分，所以修改代码时，相应的注释也要改。没有及时更新的注释会误导代码阅读和维护，甚至产生严重的副作用。

(2) 正确运用表达式间隔、语句缩进和各种括号，这些是统一、美观的代码格式的重要部分，合理的设计能显著增强可读性和可维护性，可以帮助预防和发现代码错误：

- 二目运算符，用空格将它和它的操作数分开，如 int b = a + 2;
- 不要在单目运算符和其操作对象间加空格，如!foo，~foo，++foo，-foo，&foo，sizeof(foo)，(int)foo。不加空格，保证了语义的连贯性，在阅读时能帮助区分符号相同的单目和多目运算符。
- 不要在引用操作符“.”、“->”、“[ ]”前后加空格。正确性写法如 foo.bar，pFoo->bar，foo[bar]，这样就不至于打断语义的连贯性。
- 如果表达式要分开多行，请在某个运算符之前分行，这样下一行很明显能看出是上一行的继续，如：

```
int b = a
        + 2;
```

(3) 块和语句的格式化：

- 水平缩进每次用两个空格，不建议使用制表符(Tab 键)。因为制表符的宽度是可

以自定义的，如果用在代码中会造成不同环境下代码的缩进和对齐不同，从而造成(显示)混乱的情况，因此不建议使用；但是在编辑代码时，如果可能，将 Tab 键定义为“自动替换为两个空格”方式可以加快录入进度。

- “{”和“}”应当单独占用一行，并且和控制语句的缩进相同。Visual C++编译环境提供的代码编辑器默认的编码风格就如此。另一种流行的风格是将“{”与前面的语句放在一起，这样代码更紧凑，尤其是在花括号中的语句不多时更明显。这种风格源于杂志和书籍排版时的美观要求，作为编程者，我们看问题的角度与排版不尽一致。我们认为，单独放置更明显一点，查找和配对也更直接。但有几处例外，如下：

```
// do-while 例外
do
{
    // ...
} while (length-- > 0);      // 放在一行

// struct 和 union 联合例外
typedef struct
{
    // ...
} Message;                   // 放在一行

//花括号紧跟“;”的情况
class MyClass
{
    // ...
};                           // 放在一行
```

- 应当用空行将代码按逻辑块划分，文件中的主要部分也要用空行分开。连续的两个多行定义之间需要用空行隔开，多行定义和其他代码之间应该用空行隔开。块局部变量和代码之间用空行分开。这些空行如同文章的段落标志一样，显示了代码的逻辑流。

以上给出的都是最基本的代码书写与组织规范，实践这些规范并不难，一开始就养成良好的编码习惯，会让你在职业生涯中受益终生，素质来自点滴积累。本教程正文涵盖了更多的设计规范。本书提供的源代码也实践了这些规范，大家可以在阅读和编写代码中积累经验，逐步掌握这些规范。

四、应当使用什么开发环境?

工欲善其事，必先利其器。在认准目标之后，我们准备上路实践了。现在我们说说编译器。

1. 编译器的门派之争

在 C++之外的任何语言中，编译器都从来没有受到过如此之重视。因为 C++是一门相