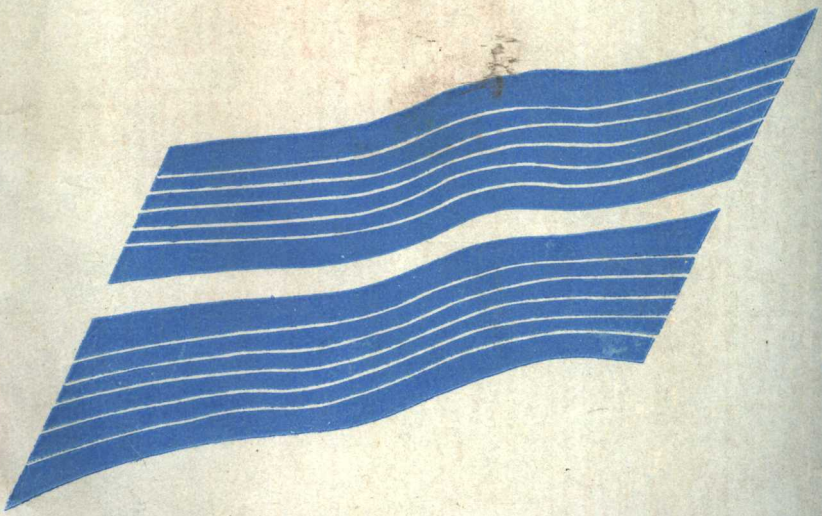
ADVANCED SOFTWARE DESIGN TECHNIQUES ADVANCED

高级软件设计技术

ADVANCED SOFTWARE DESIGN TECHNIQUES ADVANCED

〔美〕Robert J. Rader 著

郑咸义 方伟夫 译 徐秉铮 校



华南工学院出版社

高级软件设计技术

〔美〕 Robert J. Rader 著

郑咸义　方伟夫　译

徐秉铮　校

华南工学院出版社

内 容 提 要

本书用简练的篇幅，介绍软件设计中一些有沿用价值的技术，包括基本设计技术、通用设计技术、线性链表、二叉树、递归、递归的消除、回溯法、优先驱动语法分析、动态规划应用以及一些可结合到新开发的软件中去的通用程序。

本书适用于正在开发或准备开发软件的人员阅读，也适用于科技人员和大专院校的教师、研究生和大学生参考。特别向那些希望尽快提高自己的软件设计水平的朋友们推荐本书。

高级软件设计技术

〔美〕Robert J. Rader 著

郑咸义 方伟夫 译

徐秉铮 校

责任编辑 黎绍发

华南工学院出版社出版发行

（广州 五山）

广东省新华书店经销 华南工学院印刷厂印刷

开本787×1092 1/32 印张：7.4375 字数：167千

1987年5月第1版 1987年5月第1次印刷

统一书号：15410·031 印数：1—3000

定价：1.25元

译 者 序

不少人有这样的感觉：学懂一两种程序设计语言并不难，编写一两个简单的软件也还容易。但是，碰到大型、复杂的软件设计问题，就感到无从下手，难以思考了。这种现象，换一个说法是，刚刚入门的软件人员，很快就会感到，仅学会程序设计语言是远远不够的，非再学习一些软件设计技术不可。对此，我们愿意推荐这本《高级软件设计技术》，特别是推荐给那些希望尽快提高自己软件设计水平的朋友们。

这是一本颇有特色的著作，它用相当简练的篇幅，介绍了计算机软件设计中一些有沿用价值的技术。全书包括：基本设计技术、通用设计技术、线性链表、二叉树、递归、递归的消除、回溯法、优先驱动语法分析、动态规划等九章，还提供一些可结合到新开发的软件中去的通用程序，书中练习也附有答案。

本书之所以称这些技术为高级的，并不是因为它们难以掌握，而是因为大部分程序人员并不知道它们（本来应该知道），而且也不会发现它们。这些技术几乎总是要从别人那里学来的。

本书适用于正在开发或准备开发软件的人员阅读，也适用于科技人员和大专院校的教师、研究生和大学生参考。

本书的翻译出版曾得到徐秉铮教授的鼓励，译稿也承他细心校阅，在此表示衷心的感谢。

译文不妥之处，恳望读者指正。

译 者

目　录

引言……（1）
第一章　基本设计技术……（4）
一　黑箱……（4）
二　算法……（7）
三　数据结构……（10）
四　数据结构图……（13）
五　程序设计的表示法……（14）
六　流程图……（17）
第二章　通用设计技术……（19）
一　程序设计……（19）
二　循环剖析……（23）
练习……（40）
第三章　线性链表……（44）
一　单链表……（44）
二　新结点的插入……（51）
三　双链表……（53）
练习……（56）
第四章　二叉树……（58）
一　略谈堆栈……（61）
二　二叉树遍历……（62）
三　前序遍历程序……（64）
练习……（68）
第五章　递归……（69）
一　递归的例子……（69）
二　阶乘函数……（70）
三　最大公因数……（72）
四　有序数组的对半搜索……（73）
五　数组排序……（76）

六　二叉树的建立……（79）
练习……（85）
第六章　递归的消除……（87）
一　通用方法……（87）
二　CALL-RETURN 序列……（89）
三　堆栈的例子……（92）
练习……（96）
第七章　回溯法……（97）
一　八皇后问题……（97）
二　马的环游问题……（101）
三　稳定婚姻问题……（109）
练习……（115）
第八章　优先驱动语法分析……（116）
一　二元运算符……（117）
二　一元运算符……（117）
三　插入表示和逆波兰表示……（118）
四　二叉树表示法……（118）
五　算术表达式……（118）
六　记号……（119）
七　转换成二叉树形式……（131）
练习……（134）
第九章　动态规划……（135）
一　背包问题……（135）
二　网络的最短路径……（139）
三　生产调度……（142）
附录一　练习答案……（148）
附录二　说明性程序……（187）
附录三　伪码标准……（222）
英汉名词对照与索引……（228）
参考文献……（231）

引　言

软件开发是一个艰难的过程。如果你不相信这一点，我也不打算说服你，但工业上许多令人惊讶的报道却为此提供了大量的证据。

比起五年前来说，软件开发的意义现在更为人们所理解了。这是一项有大批非常有才干的人员参加的高度技术性的活动，因而它能够在技术上取得重大的进展，这是不足为怪的。软件业还很年轻，然而人们似乎已经辨认出了一些将有沿用价值的基本技术。本书的目的之一，就是将其中某些技术作详细介绍。

列入本书的材料都是经过精心挑选的。每一种技术都有它的用场，也是一个设计人员应该具有的知识的一部分。这些技术中的大多数都曾在计算机科学文献上以不同的形式作过详尽的说明。本书是为**专业软件设计人员**写的，它包括通用的基本技术。对这些基本的技术将作深入的介绍，并用实际程序加以说明。

我之所以称这些技术是高级的，并不是因为它们难以掌握，而是因为大部分程序员并不知道它们（本来应该知道），而且也不会发现它们。这些技术几乎总是要从别人那里学来的。

本书的宗旨是传授技术，而不单单是传授概念。一个出色的设计者必须同时是一个出色的实践者，而后者需要有出色的技术。尽管篇幅所限，本书还是对每一类问题作相当详尽的介绍，大部分程序可以结合到要开发的新软件中去。

软件正在愈来愈多地用来解决复杂的问题。与此相反，软件业却发现，要解决这些复杂的问题，工具必须更简单些。我们正在人脑能够处理的复杂性界限上工作。在其它条件都不变的情况下，工具越简单就意味着我们能够处理的问题越复杂。这是近来软件开发技术革命的主题之一。

这次革命通常称为结构化程序设计。对于这个术语人们提出了许多看法，以致几乎没有办法在结构化程序设计的定义上取得一致的意见。撇开理论上的争论，集中精力去学会如何做出更好的程序，这才是可取的办法。本书所介绍的算法都很短，并且自始至终规定只使用结构化程序设计中的最重要部分，即**顺序**、**选择**和**迭代**三种控制结构。所有关于这些控制结构的理论，对于专业软件设计人员来说都是无关紧要的。如果说它们有用处的话，那就是使你相信程序的制作、操作和使用更容易了。本书的所有设计都是结构化程序设计的例子。

由于本书是关于软件设计的书，因此我们必须弄清楚什么是软件设计。软件设计是整个软件开发活动的计划阶段。现在很多程序完全不是设计出来——而是建立起来的。一般来说，按这种方法所得到的程序总是不能令人满意的。这样的程序或者不能工作，或者没有解决所指定的问题，或者除了作者之外其他人都不能理解，或者无法进行修改。计划阶段的任务就是给出程序的总效能的一个表述（做什么）和产生该效能的方式（怎样做）。

设计者通常根据两种主要反馈来改正程序中的错误，这就是：（1）对设计的程序测试失败；（2）其他设计人员发现其中的错误。今天人们从大量的经验得出结论：测试并不是一种有效的手段，它费用太大、速度太慢。另一方面，由可胜任的同事对设计进行严格的复检，却足以能够发现大部

分的错误。当然，测试还是必须的，但不应该用来发现设计上的错误。我们需要的是正确的设计，即没有错误的设计。就本质来说，测试并不能表明没有错误，而只能指出错误的存在。

如果我们相信设计的复检是改正设计的关键，那么设计活动的目标就明确了，这就是使得有意义的设计复检成为可能，使设计简单得对其正确性能一目了然。要始终设想你将要使其他人确信你的设计是正确的。

具有给出清楚的设计资料的能力是相当难得的。到目前为止，也只有一些人具有这种能力，而一些人却没有。现在我们能够解释清楚设计过程中各个环节的合理比例，以便使得那些不具有这种能力的人能够获得这种能力。如果能够帮助一些设计人员沿着这条路子去取得成功，那么本书的目的也就达到了。

在本书中，算法将用一种很接近FORTRAN的语言给出。其中特别地，数组下标范围是所有1，2，…，N。这一事实适用于需要取不允许取的下标值的情形，在那种情形通常取0或－1。事实上，在约束更弱的语言中，应该允许下标值有不同的选取。

对阅读本书的建议

先读一些说明性程序。软件设计的最终产品是代码。在学习各种具体技术的过程中，对中间步骤的目标，你应该心中有数。特别要注意作为每一子程序的序言所给出的注释，这是从使用者的角度对每一子程序的黑箱描述。

第一章　基本设计技术

本章所介绍的技术，全书各章都会用到。每一个认真的软件设计人员，对这些技术都应该运用自如。

一、黑　　箱

所谓黑箱（通常就是一个子程序），是指它执行一种功能（即“箱”的方面），但如何执行这个功能却完全是隐蔽的(即“黑”的方面)。通常，黑箱表示如图1.1所示。图中大写字母是**输入**，小写字母是**输出**，f 是所执行功能的名 字。这三者合在一起便组成了黑箱的定义。为了完全确定起见，黑箱的描述还必须包括有效的输入范围，以及明 确 陈 述 与整个有效输入空间有关的输出关系。对于有效范 围 以 外 的输入，箱的行为不确定。

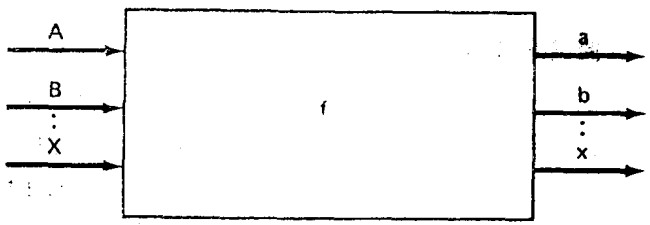


图 1.1

现在，让我们来考虑这种描述所面对的两类典型读者。一类是箱的用户。一旦接口（输入和输出）确定了，箱的功

能也明确了，用户就有了关于评价该箱是否适用以及（如果适用的话）怎样使用它的所有信息。在箱的整个正常使用期间，用户根本就不关心它的内部是如何建造的。另一类读者是箱的建造者。要建造这种箱必须有关于接口和功能的描述。有了这些接口和功能特性，箱的实现才是可行的。

我们可以断定，关于黑箱的这种描述，无论对箱的用户还是箱的建造者都是足够的。而且，增多关于箱的内部结构的信息反而有害。它会使用户感到迷惑和混乱，也会使建造者感到过分拘束而得不到可行的实现方案。

设计活动主要包括对黑箱提出规格要求。用户可以在所拟议的箱的计划阶段对它进行反复检查，以便改正误解和阐明要求。

黑箱也并不总是停留在“黑的”阶段。当我们确定了箱的规格并认为值得建造时，设计的下一个步骤是为箱子制定一个内部结构。通用的表示方法如图1.2。这时，箱子便不再是“黑的”了。我们可以看到，f 内部由称为 g，h 和 k 的箱所构成。它们中的每一个现在又是一个需要加以说明的

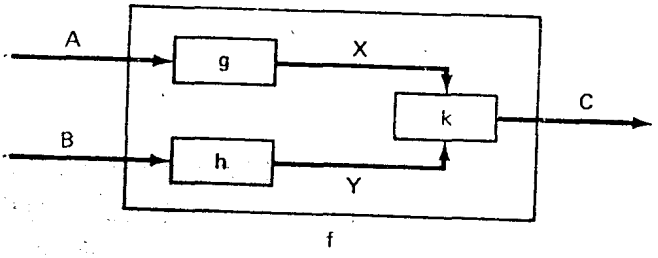


图 1.2

黑箱（见图1.3）。从 f 的黑箱图到其内部结构图这一步常称为设计的**精细**。一种较高层的功能由一些适当连接的较低层的功能构成。要求给出详细的接口说明，部分原因是为

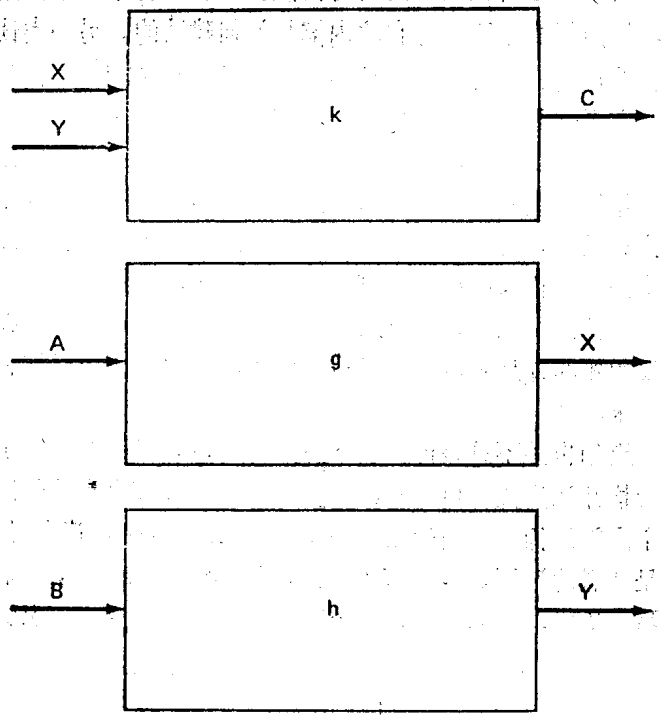


图 1.3

自上而下的设计就是用一系列逐次精细来进行设计。首先，用一个黑箱来描述顶层的功能（下面我们把功能也称函数——译注），然后给出一种内部结构，它用较低层的箱来实现顶层的箱。对这些较低层的箱又依次给出其内部结构，等等。如此继续精细，直至函数足够简单，无需进一步计划就能正确建造起来为止。

一种典型的设计陷阱是，在箱的实现期间又来讨论对箱

的要求。保留黑箱图作为用户复检的材料，这应成为一条规则，也是值得努力去做的。许多箱都是在证实其是否实用之前建成的。设计者的任务是，首先取得关于用户所需要的箱的协议，并且仅有在这以后才动手建箱。只有在得到与协议完全吻合的黑箱说明书时，用户才会心满意足。

二、算　法

箱的函数部分将用算法来设计和实现。算法是由计算机执行的一些步骤。在最低的一层是无需进一步解释的计算机指令，即它们就是黑箱。指令依赖于所用的语言，C = A + B 可作为一个例子。要产生有用的算法，必须把大量这样的基本指令组合起来。基本语句的组合方法称为**控制结构**。

近十年来在理解控制结构方面已取得了很大的进展。被

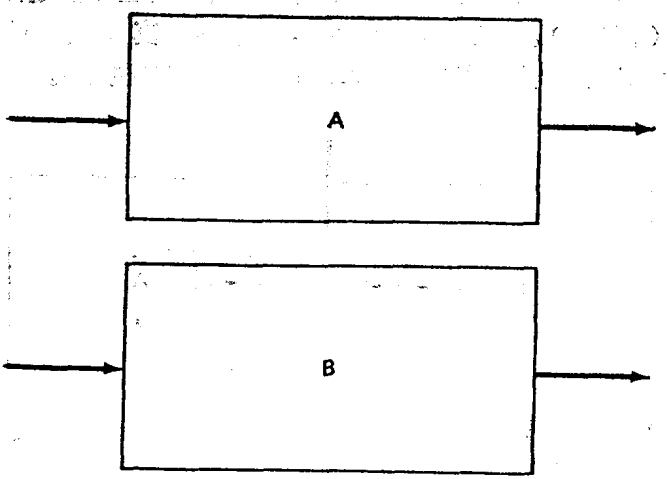


图 1.4

称为结构化程序设计的一个重要部分就与控制结构的适当选取有关。根据本书的目的，我们考虑相当简单的情况，这就是仅允许使用三种控制结构——顺序、选择和迭代。

假设有两个黑箱（如基本语句）A和B（图1.4）。我们可以把它们以**顺序**形式组合成一个新的箱C（图1.5），其意义是先执行A，然后执行B。通常是简单地把语句按顺序写下来作为代码。

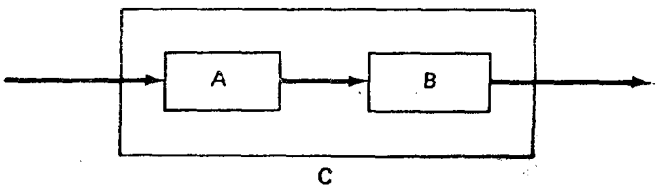


图 1.5

A和B的另一种组合方式是，在条件P的基础上**选择**其一（图1.6）。P必须是布尔条件，即只能取真或假。新箱C的意义是，当P为真时执行A，而当P为假时执行B。

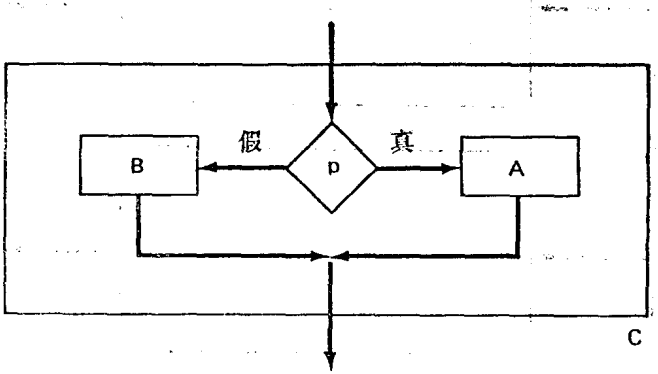


图 1.6

组合箱的最后一种方式是，在布尔条件的基础上对一个箱作**迭代**执行（图1.7），其意义是当P为真时 A 被执行零次或多次，而当P为假时迭代停止。

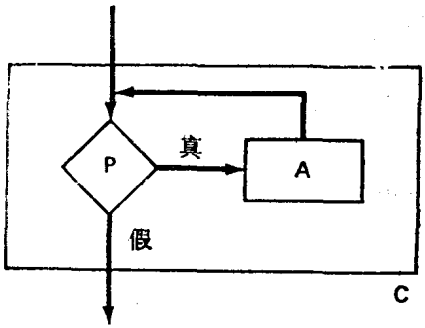


图 1.7

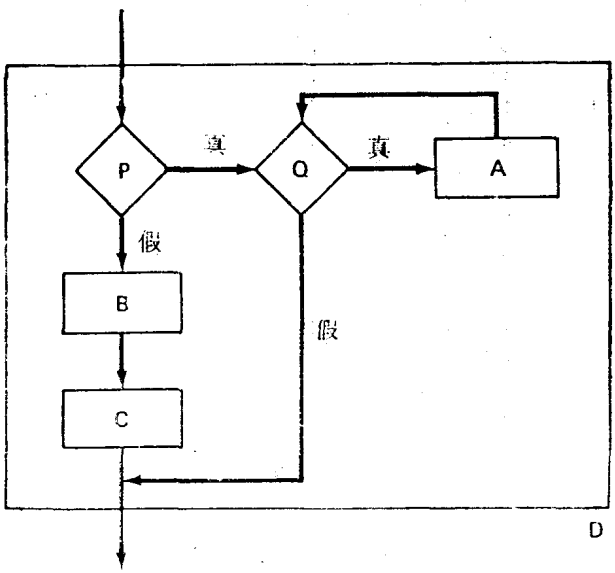


图 1.8

顺序、选择和迭代是我们用来组合箱的仅有的方法，但可反复使用，以产生复杂的算法。例如，考虑图 1.8 中的箱 D。经分析，可以看出 D 是重复使用顺序、选择和迭代构成的，如图1.9的虚线所示。D1 是由布尔条件 Q 控制的 A 的迭代，D2 是先 B 后 C 的序列，而 D 却是由布尔条件 P 决定的 D1 或 D2 的选择。

所有的箱不管其内部结构多么复杂，它们的重要特征是只有一个入口和一个出口。

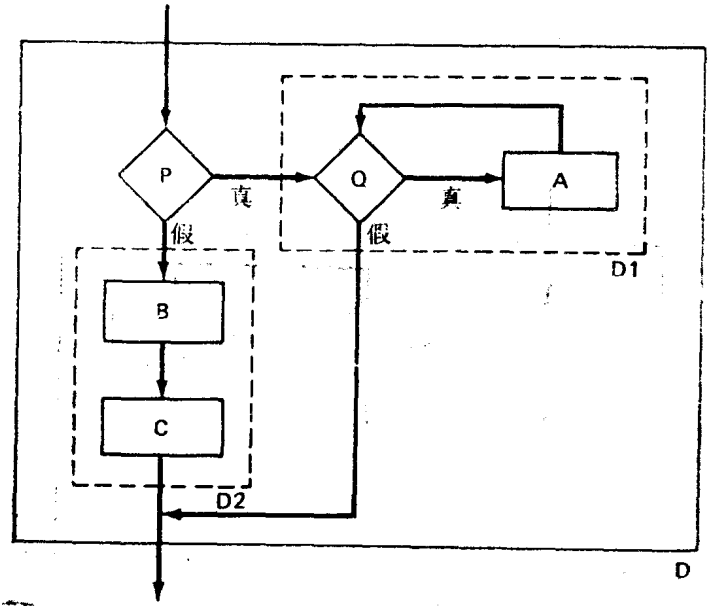


图 1.9

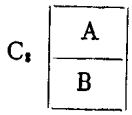
三、数 据 结 构

算法是程序中人们最熟悉的部分，然而，本质上同样重

要的却是数据结构。数据结构所遇到的情况类似于控制结构所遇到的情况，只是时间上有所延迟。这就是说，在理解控制结构上已取得了很大进展，发展了必要的理论，其思想也为实践者所广泛接受。但在数据结构方面，大多数实践者却可能还不知道存在什么问题。在文献中理论家们的看法似乎还不那么一致，但答案看来是清楚的，这就是，对于数据结构，也使用顺序、选择和迭代。

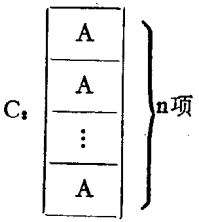
基本数据单位因问题不同而异。这里我们所说的基本数据单位指的是已命名了的最低层次的数据片。通常的例子有变量和字段。

应用顺序、选择和迭代的手段，一个复合数据单位可由两个已有的数据单位来产生。如果 A 和 B 是数据单位，则下图表示如何把 A 和 B 排成顺序来形成 C：



C 是一个数据组合的名，它总是由 A 接着是 B 所组成。这种数据结构的典型代表是磁带记录。

数据单位的迭代如下面所示：



C 由 n 个象 A 一样的数据单位组成。这种结构的典型代表

---

* 如Michael Jackson在《Principles of Program Design》中所述