计算机基础教育丛书

结构化程序设计及其在 COBOL 中的应用

谭浩强　傅金铎

清华大学出版社

结构化程序设计及其在COBOL中的应用

谭浩强　傅金铎　编著

清华大学出版社

内 容 简 介

结构化程序设计方法是国际上近年来兴起的一种新的程序设计方法。本书详细地介绍了结构化程序设计的概念及其实现方法，包括逐步求精方法、自顶向下模块化设计方法、N-S结构流程图和伪代码方法，以及结构化程序的基本控制逻辑结构。无论用哪一种高级语言编写程序，以上内容都是需要掌握的。学过任何一种高级语言的读者都能读懂这部分内容（本书第一到第四章）。本书还详细介绍了如何用COBOL 语言编写结构化程序。COBOL语言是国内外广泛流行的用于数据处理的一种高级语言。从事数据处理的人员应该掌握这种语言以及利用它编写出结构化的程序。本书还介绍了在数据处理领域中系统设计和程序设计中常使用的几种方法，如JACKSON方法、PAD方法、HIPO方法。此外，本书的每一章后都有一定量的思考练习题，并在书后附有部分习题的选择答案，这将会有助于读者自学并掌握本书的内容。

本书可作为高等学校和在职干部培训班的教材，也可供计算机程序人员自学参考。

结构化程序设计及其在COBOL中的应用

谭浩强　傅金铎　编著

☆

清华大学出版社出版

北京　清华园

北京京辉印刷厂印刷

新华书店北京发行所发行

☆

开本：787×1092 1/16　印张：18　字数：448千字

1989年5月第1版　1989年5月第1次印刷

印数：0001—5000

ISBN 7-302-00404-8/TP 133

定价：7.10元

计算机基础教育丛书

出 版 说 明

近年来，我国的计算机应用事业迅速发展，大批科技人员、大中学生、管理人员、以及各行各业的在职人员都迫切要求学习计算机知识，他们已经认识到，计算机知识是当代知识分子的知识结构中不可缺少的重要部分。

计算机应用人才的队伍由两部分人组成：一部分是从计算机专业毕业的计算机专门人才，他们是计算机应用人才队伍中的骨干力量；另一部分是各行各业中从事计算机应用的人才，他们即熟悉本专业的业务，又掌握计算机应用的技术，人数众多，是计算机应用人才队伍的基本力量。他们掌握计算机知识情况和应用计算机的能力在相当大程度上决定了我国计算机应用的水平。因此，在搞好计算机专业教育的同时，在广大非计算机专业中开展计算机基础教育是十分必要的。

非计算机专业中的计算机教学，无论就目的、内容、教学体系、教材、教学方法等各方面都与计算机专业有很大的不同，它以应用为目的，以应用为出发点。如果不注意这个特点，将会事倍功半。广大非计算机专业的师生、在职干部迫切希望有一套适合他们的教材，以便循序渐进地迈入计算机应用领域，并且不断地提高自己的水平。我们在前几年陆续编写了一些适合初学者使用的教材，受到广大群众的欢迎。许多读者勉励我们在此基础上进一步摸索和总结规律，为我国的广大非计算机专业人员编写一整套合适的教材。

近年来，全国许多专家、学者在这个领域作了有益的探索，写出了一批受到群众欢迎的计算机基础教育的教材。特别是全国高等学校计算机基础教育研究会作了大量的工作，在集思广益的基础上，提出了在高等学校的非计算机专业中进行计算机教育的四个层次的设想，受到广泛的注意和支持。我们认为：计算机的应用是分层次的，同样，计算机人才的培养也是分层次的；非计算机专业中各个领域的情况不同，也不能一律要求，在进行计算机教育时也应当有不同的层次。对于每一个学习计算机知识的人，还有一个由浅入深，逐步提高的过程。

我们认为，编辑出版一套全面而有层次的计算机基础教育的教材，目前不仅是十分必要的，而且是完全有条件的。在全国高等学校计算机基础教育研究会和许多同志的积极推动和清华大学出版社的大力支持下，我们决定编辑《计算机基础教育丛书》。它的对象是：高等学校非计算机专业的学生、计算机继续教育或培训班的学员、广大在职自学人员。

本丛书包括计算机科学技术的一些最基本的内容，例如计算机各种常用的高级语言、计算机软件技术基础、计算机硬件技术基础、微型计算机的原理与应用、算法与数据结构、数据库基础、计算机辅助设计基础、微机网络与应用、系统分析与设计等，形成多层次的结构，读者可以根据需要与可能选学。

本丛书的宗旨是针对广大非计算机专业的需要和特点来组织教材。敢于破除框框，从实

际出发，用读者容易理解的体系和叙述方法，深入浅出、循序渐进地帮助读者更好地掌握课程的基本内容。希望我们的丛书能在这方面闯出自己的风格，在实践中接受检验。

本丛书的作者大多数是高等学校中有较丰富教学经验的教师。但是，由于计算机科学技术的飞速发展以及我们的水平有限，丛书肯定会存在许多不足，丛书的书目和内容也应当不断发展和更新。我们热情地希望得到社会各界和广大读者的批评指正。

主编　谭浩强　林定基　刘瑞挺

1988.10

前　言

结构化程序设计方法是国际上近年来兴起的一种新的程序设计方法。用这种方法设计的程序具有结构清晰、易写易读的优点，它使程序设计规范化，从而降低软件成本。因而日益受到重视。现在在许多国家中，结构化程序设计已成为程序设计的主流。初学计算机的同志如果能从一开始就掌握结构化程序设计方法，就能使他们养成一种良好的风格和习惯，为今后的发展打下良好的基础。

考虑到许多读者在过去阶段学习计算机语言时未曾接触到结构化的概念，而在今后工作中又必须使用结构化的设计方法，因此我们编写了这本《结构化程序设计及其在COBOL中的应用》一书，系统地介绍结构化程序设计的概念、结构化程序设计方法要点、结构化程序设计的实现、以及在COBOL语言程序设计中如何应用结构化程序设计方法。

学习本书需要有一定的计算机高级语言的知识（至少学习过一种计算机高级语言）。本书在写法上既注意了科学性、系统性，同时又注意了通俗性，用初学者容易理解的方式阐述结构化设计方法的基本概念，并通过大量的具体例子来说明如何使用这种方法设计出结构化的程序，使读者能较快地掌握这种方法。作者认为，学习计算机的应用课程，必须强调理论紧密结合实际，学习必要的理论知识是为了解决实际问题，因此，必须善于从实际出发提出问题，总结实践中的有益经验，研讨目前尚存在问题的解决方法，而不能从概念到概念，纯学术性地讨论问题。计算机科学技术的发展过程本身就是不断地从实践中提出问题和解决问题的过程。

本书的第一、二、三、四章叙述的是结构化程序设计的一般原理与方法，不论使用哪一种高级语言，都可以参考这四章的内容，它具有普遍意义。学习这几章不需要COBOL语言的基础，学过任一种高级语言的读者都能很容易地掌握其基本内容。

结构化程序设计不仅适用于数值计算领域，而且适用于非数值运算领域。尤其在事务处理领域中，由于问题本身比较复杂、数据层次较多，因此，在事务处理领域中使用结构化方法进行系统分析、系统设计和程序设计，其重要意义尤为突出。处理这些问题的难度也相对大一些。因此本书除了介绍一般的程序设计的实施方法（例如自顶向下、模块化设计、N-S结构化流程图、伪代码方法等），还介绍在事务管理领域中广泛使用的一些结构化程序设计的实施方法（如JACKSON方法、WARNIER方法、PAD方法、HIPO方法等），读者可以了解并选择使用这些方法。

如果有的读者不准备深入探讨事务管理中的问题，在第一次阅读本书时，可以先跳过第六、七两章。学过COBOL语言的读者，在第一次阅读时可以先阅读第一、二、三、四、五、八、九章。在消化了这些内容以后再回过头来学习第六、七章的内容。

要解决一个事务处理系统的任务，可以用不同的计算机高级语言。目前国内外使用较多的是COBOL语言。它是一种专门为事务处理而设计的高级语言。它的结构严谨、报表输出功能强，用COBOL语言写的程序与自然语言很接近，具有“成文自明”的特点，因此在事务管理领域得到广泛的应用。COBOL语言应该是从事计算机事务管理工作的同志必须掌握

的一种基本语言。因此本书重点结合 COBOL 语言来介绍如何具体实施结构化程序设计。应当说明，本书不是一本系统介绍 COBOL 语言的书，不可能在本书中再花过多的篇幅去具体介绍COBOL的语法规则，而是在读者已初步掌握COBOL 语言的基础上，讨论怎样运用 COBOL 语言去实现结构化程序设计，并提高程序设计的质量。如果读者对 COBOL 语言本身还不太熟悉的话，可以参阅清华大学出版社出版的《COBOL语言》上、下册（谭浩强编著）。

本书是一本综述结构化程序设计的书籍，在第六、七章概要地介绍了几种常用的结构化设计方法，由于篇幅的关系，同时考虑到多数读者的情况，我们只作初步的介绍，以使读者有一大致的了解，提供今后进一步深入学习的基础。如果有的读者在工作中需要用到其中某种方法而需详细掌握其细节，可以参考有关的专门书籍或资料。

为了使读者更好地掌握本书中的内容，在每一章后都有一定数量的思考练习题，读者在学完每一章后可以利用这些思考练习题消化巩固已学的内容并检查自己掌握的程度。在本书的最后提供了部分习题的参考答案（包括了一些主要的习题的答案），以帮助读者学习。

本书可以作为高等学校高年级学生的教材或参考书，也可供计算机学习班使用，或供计算机程序工作人员和管理人员自学参考。

由于计算机科学技术发展很快，以及作者的水平有限，本书内容可能会挂一漏万，甚至会有错误，欢迎读者和专家们提出宝贵意见。

作者谨识

1987.10.

目　录

第一章　结构化程序的概念

§ 1.1　结构化程序产生的背景

任何一场革新，任何一种新方法新技术的出现，绝不是某个科学家或革新者一时灵感的自我表现，而是工业和科学技术发展到一定阶段客观需求的必然结果。

结构化程序设计方法的出现不是偶然的，它是计算机软件行业发展到一定阶段客观的要求，是程序设计方法论的一次革命。为了弄清楚在计算机软件发展的历史中为什么会产生结构化程序设计，首先让我们简单地回顾一下传统程序的设计方法。

在大多数程序设计高级语言中，无例外地都拥有一种无条件转向语句即GO TO语句，有经验的程序员能够灵活地运用GO TO语句创造出风格迥异、技巧丰富的程序作品来。当时，衡量程序设计的水平往往是看程序构思是否巧妙，程序语句是否精炼，程序运行时所占的内存空间是否较小以及程序运行时间是否较短等等。这种判别的标准在50年代至60年代中期确实为大家所默认。

因为，当时电子器件制作工艺水平还没有发展到今天这样的高级和成熟。由于计算机硬件的生产、存储器的容量、中央处理器的运算速度受到当时技术条件的限制，因此计算机存储器的容量不能做得很大，运算速度不能做得很快，特别是大容量磁盘存储器还没有得到广泛的使用。即使这样，初期硬件的成本亦是昂贵的，而程序设计所花费的成本相对来说是比较低的。这种情况可由图 1.1 表示出来，图 1.1 中横轴是时间，纵轴是软件（硬件）的相对成本。

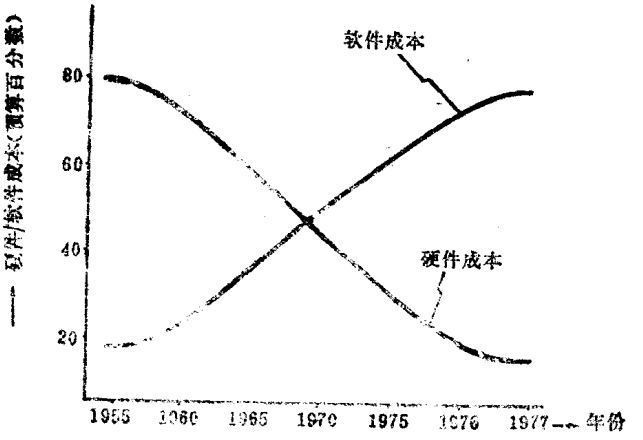

图 1.1

从图 1.1 中可见，随着时间的发展，硬件成本逐步下降，而软件成本在逐渐增高，二者成本价格相互地向自己的相反方向转变。这种软硬件价格变化的客观事实是结构化程序出现的更深刻的经济背景。

如上所述，在计算机发展的初期程序的成本是较低的，程序设计的方式是手工业式的，程序的编写没有什么规范可以依循，全靠程序设计人员的个人习惯和技巧。因此，基于程序员个人的技术和素质而产生的程序设计风格是因人而异的，在程序里充分体现了程序员的个性、素质及聪明才智和技巧的娴熟，这是程序员喜欢津津乐道并为人们赞赏的一面，然而另一方面它却包含了一种巨大的潜在弊端。

由于在传统程序中程序员无约束地使用GO TO语句，使得程序语句的书写顺序和程序语句的执行顺序互不一致。从程序执行角度上观察，好象按书写顺序的那些程序语句象面条一样互相纠缠在一起，这种情况如图 1.2 所示。

上述这种传统风格的程序称为BS型程序（BS是A BOWL OF SPAGHETTI的缩写，即“一碗面条式样”的程序）。由于人们阅读程序时是按照程序的书写顺序去阅读，因此阅读BS 型程序就好象让读程序的人去理顺一团互相缠绕在一起的乱麻一样，这是一件多么困难的任务！这种 BS 型程序不仅使别人难于读懂，就是程序员本人在相隔一段时间后，重新阅读自己编写的源程序清单，回忆他的程序逻辑时，也会感到茫然。既然 BS 程序的易读性差（使人难以读懂），自然也难以维护。特别是近一二十年，随着计算机技术的发展，各个计算机厂家都在极力发展自己的系统软件，使之功能更趋完善。多功能的灵活的系统软件的提供，又促进应用软件更进一步地发展，大量应用软件的开发消耗了大量的人年，然而这些应用软件的维护耗费的人年则更为可观，几乎是开发过程中所用去人年的 3 ～ 5 倍。60年代末出现的软件危机的一个方面就是指应用软件的大量开发以致难于驾驭和维护它。应用软件难于维护的原因就是传统 BS 型程序的非规范性，人们随心所欲地编写程序，以致使大量的程序难以阅读和维护。仔细分析一下我们就会发现，这是由于 BS 型程序的控制流中存在着许多无规范的转移，而人们在阅读程序时，其思路又难以“跟踪”上这种紊乱的控制流，从而使阅读和分析程序成为一件难度很大的“高水平”的工作。即使是一个经验丰富的程序设计人员，在一个大型的BS程序面前也会一筹莫展，这种程序如果只是为自己编写的，问题还不算严重，如果提供社会使用，就会浪费社会上大量人力，而软件人员的劳动成本是昂贵的。人们在实践中，终于认识到，问题的症结在于程序缺乏一种“良好的结构”。

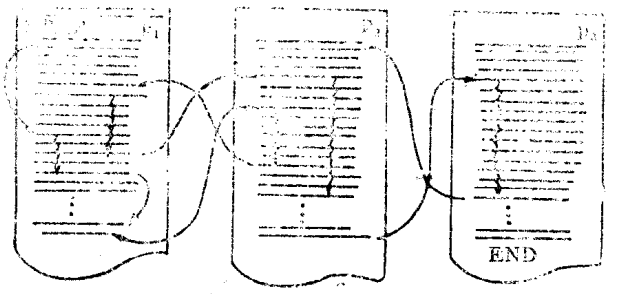

图 1.2

人们设想，如果能确定一种规范，使程序设计者按照这种规范行事，就能写出具有良好结构的程序，这种程序易于编写、易于阅读、易于调试、易于修改维护，那么以上问题就得到解决了。

寻求一种良好结构的程序设计方法，就是当时很多软件工作者努力追求的目标。在60年代末到70年代初，软件逐渐发展成一种行业，从事软件开发的人员越来越多，程序已不再是个人创造的“工艺品”，而逐渐变成了一种大生产下的“软件产品”，往往需要许多人共同来完成一个软件产品，这就要求寻找一种标准化的、规范化的产品式的程序设计方法，以适应软件发展新的要求。在这样的历史条件下，传统的BS型程序设计受到愈来愈多的挑战，结构程序设计的概念及方法也就应运而生了。

§ 1.2 结构化程序设计方法的发展过程

关于结构化程序概念何时正式被人们确认，这个时间界限是很模糊的。如果我们想追究的话，那么可以回朔到1964年，当时在以色列一个非公开性的国际会议上，Böhm和Jacopini

介绍了他们用意大利语写成的一篇论文："流图、图灵机和具有两种格式规则的语言"。然而，当时该篇论文所持有的关于结构化程序观点却被世界计算机大国——美国忽略了。明确地、正式地提出结构化程序设计思想的主要人物之一是荷兰埃茵候温大学 Edsger W. Dijkstra 教授。1965 年Dijkstra教授发表了题为"程序设计是一种人类的活动"的论文，在此篇论文中他积极提倡用结构化方法去进行程序设计，特别提出要从程序设计语言中摈弃GO TO语句的使用，因为GO TO语句是产生传统的 BS 型程序弊病的主要根源。次年P.J.Landin发表的"后续的700个程序设计语言"和P.Naur发表的"程序翻译是通用的数据处理问题"，两篇文章也同样支持了Dijkstra教授的观点。从1965年到1966年相继公布了COBOL-65及 FORTRAN X3.9——1966 版 本。当时，程序界对于传统BS型程序存在的问题确也感到头痛，但尚没有找到一种较好的解决办法，因此Dijkstra 观点的发表理所当然地引起了程序界的注意。

1966年C.Böhm和G.Jacopini的论文"流图、图灵机及仅含两个格式规则的语言"被译成了英语，发表在 Communication of ACM 杂志上，作者从另一个高度提出了程序设计中的三个基本控制结构，它们是：（1）顺序结构（见本书图1.20），（2）选择结构（见图1.21），（3）重复结构或称循环结构（图1.22）（关于这些内容我们将在第三章中给予详细介绍），并断言，用任何高级语言编写的程序都可以分解为上述三种基本控制结构以及它们的组合，该篇文章发表的意义，在于它是继Dijkstra之后又将关于结构化程序的讨论向前大大地推进一步。

用ALGOL，COBOL，PASCAL，和PL/1语言可以不用GO TO语句就能实现上述三种基本结构。有趣的是对FORTRANⅣ语言而言，如果不用 GO TO 语句就难以实现选择结构。究竟程序中能否摒弃GO TO语句成了争论的中心。

1968年W.Dijkstra又发表了题为"有害的 GO TO 语句"一文，他认为程序员在其设计的程序中使用GO TO语句的密度是衡量一个程序员质量优劣的标准，并且建议在所有的高级语言中废除GO TO语句的使用。这篇文章内容虽然多处重复了他以前发表的观点，但读来仍然使不少人感到新鲜和鼓舞。W.Dijkstra 的论点引起了完全不同的强烈的反响，有拥护者，也有反对的。某些拥护者甚至走上了极端，以致歪曲了 W.Dijkstra 的观点，结果在他们眼里结构程序变成了"非GO TO化程序"的同义语。正是由于关于GO TO语句的这种争论使得在当时计算机界引起轩然大波，因此第25届计算机协会（ASSOCIATION OF COMPUTER MACHINES)的国际会议曾以"关于 GO TO 的争论"为题开展过专门的讨论。后来(1972 年)M.E.Hopkins专门为此发表了论文"关于GO TO 的公案"，在此篇文章中他较为全面地评价了GO TO语句的作用，他认为GO TO语句首先被人们滥用，后来又被人们误解，认为它毫无价值，以致要全然否定它，其实GO TO语句在某些可读性好的程序中还是有价值的。读者将从我们后边的论述中看到Hopkins的观点的确是比较公正全面的。

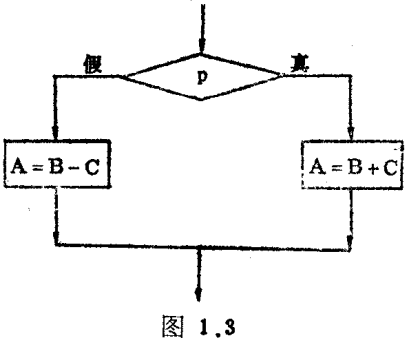

图 1.3

举例来说，如果用FORTRANⅣ语言编写程序，为了要实现"选择结构"（见图1.3），就要用到GO TO语句。

选择结构	用FORTRANⅣ语言来实现
IF(P)	IF(NOT P)GO TO 10

```
  A=B+C                          A=B+C
ELSE                             GO TO 20
  A=B-C                    10    CONTINUE
ENDIF                              A=B-C
                           20    CONTINUE
```

上例右边的FORTRAN Ⅳ程序段就是对左边“选择结构”的模拟。该程序段虽然包含了GO TO语句，但从基本控制结构观点看，是可以接受的。

此外，还有一些关于结构程序的代表性观点。我们应该特别提及的是，1971 年 Niklaus Wirth 的论文“用逐步求精法开发程序”。Niklaus 的论文奠定了结构化程序设计过程所应遵循的基本步骤，Niklaus 认为程序整体结构是由一系列求精过程所组成，而每一步都包括把一个任务分解成若干个“子任务”的过程。例如，“打印出全班的平均成绩”的任务可以由图 1.4 表示的三个“子任务”组成。

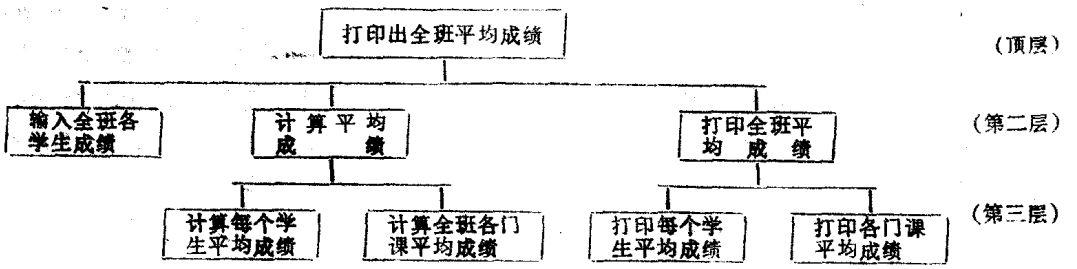

图 1.4

每一个“子任务”可以单独编写成一个模块或子程序。子任务还可以再分成更小的模块，例如，“计算平均成绩”这一子任务又可分解为“计算全班每个学生各门课平均成绩”和“计算每门课的平均成绩”两个子任务。愈往下分解就愈具体化、精细化，也愈易于实现。可以看到，最初拿到的任务“计算和打印全班学生平均成绩和每门课平均成绩”是笼统的、抽象的，难以直接实现，在逐步分解（直到最下层）之后，就很容易写出程序语句来实现它了。这个过程就称为“自顶向下逐步求精”的过程。

一般来说，在程序设计过程中，还要考虑和定义数据结构。随着程序功能的逐步求精，数据结构也要逐步具体化，这就是数据结构的逐步求精。图 1.5 表示“学校信息”逐步求精的过程（有关这方面的知识在此不作详细介绍）。

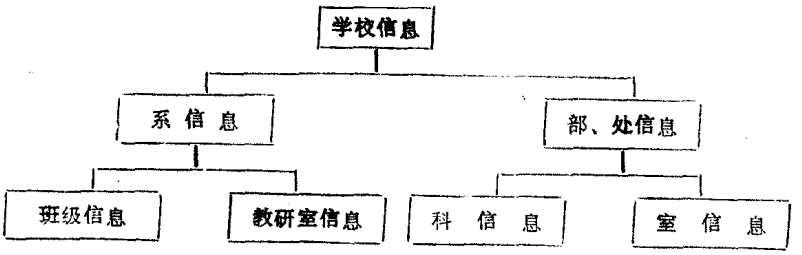

图 1.5

1971年末 Harlan Mills 和F.T.Barker博士在为纽约时报开发的一个联机信息检索系统中，首次在商用数据处理系统中使用结构化程序设计方法。该系统共有 83000 步源程序，应属于大的应用系统，据统计每 10000 步程序只有 4 个错误，其花费的人月是用常规方法开发所

花人月的4/6。实践证明，它是非常有效的程序设计工具。

1973年第12期的DATAMATION杂志则声称结构程序是“程序设计史上的一次革命”。

总之学术界的讨论逐步澄清了结构化程序设计的概念，进一步促进了这一新的设计方法的应用推广，但真正的推动力还是计算机用户本身，只有依靠用户的不断实践和积累经验，才能进一步发展和丰富这一有力的程序设计工具。由于软件开发费用的激增及了解到结构化程序设计方法的优越性，美国政府有关机构曾积极倡导在一些部门中推广和使用结构化程序设计方法。近年来由于我国商业及管理方面数据处理的发展，结构化程序设计也得到广泛的应用。不少数据处理中心、信息中心、计算中心希望把结构化程序设计纳入该单位的程序设计规范。相信随着我国广大程序设计人员对结构化程序使用经验的积累和丰富，这一先进工具将会在我国更好地推广开来并日益显示出它的重要作用。

§ 1.3 逐步求精与软件工程方法

逐步求精方法是结构化程序设计中一个极其重要的组成部分，是结构化设计思想的核心，它指明了结构化设计过程所必须遵循的方法和步骤。

从程序的设计目标（功能规定）到产生源程序之间要经历一段距离，对于简单的程序而言这个距离是很短的，甚至可以一步就达到。例如：从A，B，C三个数目中求出最小数。对于求解这种问题可以不加更多思索地直接写出 COBOL 源程序段如下：

```
IF A<B
   MOVE A TO MIN
ELSE
   MOVE B TO MIN
IF C<MIN
   MOVE C TO MIN
```

程序执行结果在MIN中保留了A，B，C中最小数。

那么对于设计一个从含有1000个数或更多数的数列中寻求最小数字的问题来说，直接写出源程序就不是一件容易的事了。对于一个庞大而复杂的程序而言，从提出程序目标到最后产生源程序，中间需要经历一段更大的“距离”。因此程序设计的目的就是为了能够正确地、有规律地跨越这个“距离”。近年来发展了一套用工程设计的方法来“生产”软件，这就是“软件工程”的方法。

如果我们要处理的是一个不太复杂的程序（即它不是一个庞大的、复杂的数据处理系统），那么，只要采取以下的逐步求精方法就可以得出结果。

1．分析问题。经过分析，将程序需要完成的功能概括地、明确地用文字整理出来，这一步称为“功能的规定”。

2．根据要求，用简明的话概括写出程序最后应完成的目标。例如，图 1.4 中它就是“打印出全班平均成绩。”这只是一个总的任务，或者称为“宏任务”（或“宏功能”）。它是高度概括的任务，它并没有具体指出应包括哪些具体操作。因此这一步可以认为设计了一个“抽象的程序”。

3．将上述宏任务分解为若干子任务（子功能）。如图1.4的“第二层”，它比顶层具体

一些了。需要说明的是，每一步由上一层向下一层的求精过程都应确保其正确性，也就是说，如果实现了第二层的子功能，就能实现顶层设计的宏功能。假设求精过程进行得不合适，例如少了中间那个“计算平均成绩”，就不能认为求精过程是正确的。

4．再向下分解。如图1.4中第三层。也就是将子功能再分解为更小的子功能。这个过程一直继续下去，直到每个子功能简单到不能（或不需要）再分解为止。这就是“逐步求精”。

5．在程序功能逐步求精的过程中，同时实现数据结构的逐步求精。

其实，逐步求精方法不仅适用于程序设计，而且适用于一般的工程设计，甚至社会生活中。例如，写一篇调查报告，可以采用这样的方法：先确定题目和中心思想，再将整个报告分成几个大部分以及各部分的题目，然后再进一步决定出每一部分应表达哪些思想要点……，直到作者认为此提纲已足够详细为止，按此就能顺利地写出报告来。

因此，可以说，逐步求精方法是关于“方法论”知识中重要的一部分。这种从抽象→具体，从宏功能→子功能，从整体→细目的分解过程，以及最后逐一实现这些细目的过程，是非常科学的、具有严密的逻辑性的。因此，Weinberg在其论文“献给程序设计初学者”中指出：程序设计是人们思维活动的一面镜子。也就是说，程序员的程序构思是否有条理，是否有逻辑性，是否经过规范化训练，这些素质都要在程序中表现出来。

逐步求精方法是由“程序设计目标”走向源程序的正确途径，也就是说，用逐步求精方法才能从“提出任务”起一步一步地具体化，最后达到目的——写出源程序。

如果我们面对的不是一个简单的程序，而是一个比较大的、复杂的数据处理系统（一个系统中可能不止一个程序，而是由若干个程序组成的）。那么，进行这个系统的开发，就不仅是上面所说的程序设计过程了。从软件工程的观点，一个软件系统的生存周期应该经历以下几个阶段：

1．问题定义与需求分析
2．概要设计
3．详细设计
4．编写程序与单元测试
5．综合测试与确认运行
6．系统维护

或者说，软件生存周期由以下三个时期组成：①软件定义时期（即上面的第1个阶段）；②软件开发时期（上面的第2～5阶段）；③软件维护时期（即上面第6阶段）。

软件定义时期的任务是确定：“要解决的问题是什么”，“有无可行的办法”，“为了解决这个问题，系统应该做什么”，即确定要开发的软件应当具有哪些功能，以及系统的运行环境等。这是进入开发时期之前必须首先解决的问题。这个时期的工作是由系统分析员和用户共同进行的，最后应提出一个完整准确的系统逻辑模型。

进入开发时期后，首先进行概要设计，这个阶段的任务是：从宏观的角度提出“怎样解决这个问题”，确定解决问题的方案，将一个大的系统分解为若干个子系统，即面向用户的“功能模块”（简称“用户功能模块”）。根据“功能模块”要求实现的功能，确定面向程序员的“程序模块”。程序模块是从程序员的角度考虑如何实现“功能模块”的要求。要为每一个程序模块写出一个“程序设计任务书”。该任务书中所叙述的程序目标、程序处理说明、

算法规定等都应当是能为程序员所理解和接受的。当程序模块的规模较大时，还需要将一个程序模块进一步划分为若干个程序子模块。

在详细设计阶段，程序员需要根据交给他的程序模块或程序子模块的“设计任务书”的要求，详细描述算法。也就是解决“怎样具体实现这个模块”。这个阶段还不是编写高级语言程序，而是详细地设计出解题的每一个步骤，可以用流程图伪代码或IPO图来表示。

在完成详细设计之后，进入编程序阶段，即用高级语言来实现详细设计阶段描述的各个操作步骤。这一工作称为“编码”。只要详细设计阶段的结果是正确的，编码阶段不应有大的问题，关键是要熟练地掌握和使用高级语言。在编写好程序之后，要分别对每一个模块进行测试，检查它们能否正确实现所规定的功能。如有错误应及时发现和改正。

然后将各个模块联结起来测试，使软件达到预定的要求。如已确认整个系统已达到预期目标，即可交付使用。

软件开发时期的工作到此结束。

在程序交付使用后，在实践中必然会发现一些问题；或者根据用户的需要，要求增加一些新的功能；或者由于运行环境的改变（例如计算机系统的更换），需要对软件作某些修改。这就是软件维护时期。这个时期是很长的，直到该软件被淘汰为止。

定义、设计与编码、测试、使用维护各个阶段所花费的工作量大体的比例为：
1：10：50：（50～1000）可见，越到后面的阶段所花的工作量越大。

应当说明，以上各阶段的划分并不是很严格的，也不是一成不变的。一般意义上的程序设计指的是软件开发时期中的设计和编码，即涉及概要设计、详细设计和编码阶段。本书也主要讨论这几个阶段的问题，尤其是详细阶段和编码阶段的问题。

§1.4 结构化程序的特征

什么是结构化的程序呢？可以从不同的角度来理解和叙述它。第一种观点是所谓“分析的观点”，即用分析的方法判别一个程序是否是结构化程序。第二种观点是所谓“综合的观点”，即先定义出一些基本结构，然后用这些基本结构组合（综合）成结构化的程序。本节介绍第一种观点——分析的观点，本章§1.5介绍第二种观点——综合的观点。

下面先介绍结构化程序的第一种观点——分析的观点。看图1.6。这是一个抽象化了的程序流程图。如果问，图1.6是不是一个结构化的程序流程图呢？我们回答是从分析入手产生答案。

为了便于分析，我们先引出结构化程序中的“良结构程序”的概念。所谓“良结构程序”是指该程序的结构应具备以下特征：

1. 只有一个入口，一个出口。例如图1.7中A为入口，B为出口。符合“一个入口，一个出口”的原则。请注意不要将虚线框（是一个“选择结构”）的入口、出口与图中菱形框（判别框）的入口、出口相混淆。对菱形框而言有一个入口两个出口，但对虚线框而言，只有一个入口，一个出口。

2. 通过该结构中的任一部分都应当存在着一条从入口到出口的路径。