计算机科学与应用丛书

结构程序设计基础

编著　施伯乐　夏宽理　冯志林　丁宝康

THE FUNDAMENTAL OF
STRUCTURED PROGRAMMING

Zhejiang Kexue Jishu Chubanshe

计算机科学与应用丛书

THE FUNDAMENTAL OF STRUCTURED PROGRAMMING

结构程序设计基础

浙江科学技术出版社

复旦大学

施伯乐　夏宽理

冯志林　丁宝康

编　著

责任编辑　周伟元

封面设计　孙　菁

结构程序设计基础

复旦大学

施伯乐　夏宽理
冯志林　丁宝康　编著

*

浙江科学技术出版社出版
浙江新华印刷厂印刷
浙江省新华书店发行

开本：787×1092　1/16　印张：19.75　字数：491,000

1988年9月第一版

1988年6月第一次印刷

印数：1—6,450

ISBN 7-5341-0086-0/TP·1

统一书号：15221·152

定　价：5.00元

内 容 提 要

本书比较全面地介绍结构程序设计的概念、原则和方法，包括由顶向下、逐步求精的结构化程序开发的方法和一些程序测试技术、程序证明方法，以及 Pascal、C、Ada 语言的结构程序设计，还讨论了使用非结构化语言 BASIC、COBOL、FORTRAN 语言进行结构程序设计的技巧。本书附有丰富的实例，以帮助读者理解和运用这些方法。

本书可供软件人员阅读，以提高他们的程序设计能力，也可作为高等院校计算机专业的结构程序设计课程的教材或程序设计课程的教学参考书。

前　言

计算机的应用已从早期的科学计算逐步推广到现在的数据处理、信息通讯和过程控制等应用领域，计算机程序系统从结构简单的计算程序发展到操作系统、数据库和实时过程控制等复杂的大型软件系统，程序的研制方式从个体手工方式演变为集体化大规模的生产方式。

由于软件系统的规模越来越大，结构越来越复杂，因此，60年代末期几个典型的大型软件系统的研制都陷入了困境。软件工作者从失败的教训中认识到，必须变革原先的程序研制方法。程序的研制不应该是自由发挥程序员个人才能的一种艺术创作，而应该如同工程项目的开发，需要有一套完整的设计方法和规程。通过人们不断地讨论和实践，提出了结构程序设计的方法。实践证明，现在广泛应用的程序设计结构化，由顶向下、逐步求精的程序开发技术，是卓有成效的。

现代软件工作者的程序设计能力至少包括：能熟练使用程序设计语言，有运用结构程序设计方法系统地开发程序的技能，并能切实地掌握程序测试的基本技术。本书从这些基本要求出发，系统地阐述了结构程序设计的原理和方法，较全面地讨论了一些实用的程序测试技术，介绍了几个有代表性的程序设计语言。全书共分四个部分。第一部分是本书的重点，介绍了结构程序设计的基本概念、原则和方法，其中详细讨论的由顶向下、逐步求精的程序开发技术，给读者提供了系统地开发程序的方法。第二部分扼要地介绍了三个结构程序设计语言：Pascal,C和Ada语言，阐述了它们的功能和用它们进行结构程序设计的技术。第三部分论述了程序测试和证明的方法，详细介绍了目前已被广泛采用的测试技术，并概述了程序证明的基本原理。第四部分讨论使用非结构化语言：BASIC、COBOL、FORTRAN语言，进行结构程序设计的技巧，结合我国计算机应用的实际状况，给那些习惯使用这三种语言的读者提供了一些有益的指导，帮助他们也能运用结构程序设计的方法来开发程序。

本书面向从事计算机程序研制和开发的读者。对于具有一定的程序设计能力的读者，能受到结构程序设计的训练；对于已有较高水平的软件工作者，也能从掌握新的程序设计方法中获得有益的帮助。

本书叙述深入浅出，通俗易懂，实例丰富，每章附有适量的习题，便于读者自学。本书可作为结构程序设计课程的教材，建议重点讲解第一、第三两部分，第二、第四部分可作适当筛选，以40学时讲授为宜。本书也可作为程序设计课程的教学参考书。

中国计算机学会软件专业委员会副主任朱三元同志仔细审阅了本书全稿，谨此表示衷心的感谢。

限于水平，加之时间匆促，书中欠妥之处，诚望读者指正。

编著者

1985年11月于上海

目　录

第一部分　结构程序设计的原理和方法

第二部分　结构程序设计语言

第三部分 测试和证明

第四部分 非结构化语言的结构程序设计方法

第一部分　结构程序设计的原理和方法

第一章　引　论

§1.1　结构程序设计的由来和发展

结构程序设计（Structured Programming）首先是由计算机科学界的知名学者、美国计算机学会图灵(Turing)奖获得者E.W.Dijkstra提出的。E.W.Dijkstra在程序语言、操作系统和程序设计方法等方面都有重要贡献，他也是结构程序设计和程序正确性证明等方面的创始人之一。早在1969年，在北大西洋公约组织的一次学术会议上，Dijkstra发表了一篇论文，题目就是"Structured Programming"。他在文中指出，正确运用结构程序设计技术，能使我们的程序设计能力提高一个数量级。他在这篇文章中阐述了与结构程序设计密切相关的一些问题。

首先，由于程序规模的增大将对程序的正确性产生影响，因而结构程序设计的提出是面向中、大型的程序系统的(对于规模较小的程序，不一定非按结构程序设计的原则去办)。我们都知道，程序越大，可靠性的保证就越差。设一个大程序是由N个小程序构成，并设每个小程序正确的概率为P，则大程序正确的概率不超过$P_1 = P^N$。若N很大，则P_1就很小。若要使P_1和P近似，也即P和P^N近似，则要求P几乎等于1。60年代以来，计算机的程序系统日趋庞大，为了保证大型程序系统的正确性，才提出结构程序设计的思想。

其次，Dijkstra分析了程序正确性问题。他指出，程序测试只能用来证明错误的存在而不能证明错误的不存在。于是，程序的正确性只依赖于程序本身的结构。他还认为：对于结构程序设计而言，我们并不特别关注如何证明一给定程序的正确性，所关心的问题是：对于怎样的程序结构，我们可以花不太多的努力得到正确性的证明，即使程序非常大的时候也是如此。也就是说，对于一个给定的任务，我们如何去求得相应程序的一种好的结构。应该说，Dijkstra的这种认识是很深刻的。

接着Dijkstra讨论了采用逐步抽象的方法对于程序设计的作用。考虑如下形式的程序：

$$S_1;\ S_2;\ \cdots;\ S_n \qquad (1)$$

并假定：当每个语句S_i的作用给定后，对程序(1)的n个语句的累积作用的理解，需要有n

步推导。

再看如下形式的程序：

IF B THEN S_1 ELSE S_2 (2)

并假定：当给定语句 S_1 和 S_2 执行的作用后，对程序(2)的作用的理解需要有两步推导。即，一步是对于情况B，另一步是对于情况非B。

现在，来考虑如下形式的程序：

IF B_1 THEN S_{11} ELSE S_{12};

IF B_2 THEN S_{21} ELSE S_{22};

. . .

IF B_n THEN S_{n1} ELSE S_{n2} (3)

对应于程序(2)的情形，(3)中的每个语句

IF B_i THEN S_{i1} ELSE S_{i2} (4)

的作用的理解需要有两步推导，而(4)的执行等价于执行一条抽象语句S_i。由于有n个这样的语句，所以把程序(3)的形式化为程序(1)的形式就需要 2n 步推导。于是，对程序(3)的作用的理解，总共需要 3n 步推导。

但是，如果不引进抽象语句 S_i 而直接以语句 S_{ij} 的形式去理解程序(3)，那么我们将要在 S_{ij} 的 2^n 种不同的路径中去选择，而每种选择又需要 n 步推导去理解，则程序(3)的作用的理解将共需 $n \cdot 2^n$ 步的推导。例如设 $n = 3$，则可供选择的路径有：

S_{11}; S_{21}; S_{31}　　S_{12}; S_{21}; S_{31}

S_{11}; S_{21}; S_{32}　　S_{12}; S_{21}; S_{32}

S_{11}; S_{22}; S_{31}　　S_{12}; S_{22}; S_{31}

S_{11}; S_{22}; S_{32}　　S_{12}; S_{22}; S_{32}

而每种路径选择又需要三步去理解，则共需要二十四步推导才能理解整个程序的作用。

显然，引进抽象语句 S_i 对于我们理解程序是很有利的。同样，引进抽象的数据结构也能使我们受益匪浅。这里不再详述。

最后，Dijkstra 把程序看成是由一些珍珠串成的项链，这些珍珠是严格按照由顶到底的次序串起来的。顶上的珍珠表示程序最抽象的形式，而下一粒珍珠，一方面细化解释其上面的珍珠，另一方面本身又由其下面的珍珠来细化解释。以此类推，直至最底层的珍珠。在相邻两粒珍珠之间可以剪一刀，刀口以下是提供解释的程序，而刀口以上即是使用的程序部分。他认为以这样的结构模型来看待程序，对于理解和修改程序是很有帮助的。

Dijkstra 这篇论文内容丰富，见解深刻，它奠定了结构程序设计的思想基础。

1972 年，E.W.Dijkstra 与 C.A.Hoare 和 O.J.Dahl 合作写了一本书名为 "Structured Programming"(结构程序设计)的书。在书中，Dijkstra 提出了大的程序系统的质量和可靠性问题，并讨论了其设计和实现的方法。Hoare 提出了数据结构(如笛卡尔积、数组、序列等)的设计原理，并阐述了抽象概念的作用。Dahl 和 Hoare 一起用SIMULA 67 语言说明了怎样的程序系统能最贴近地反映程序和数据结构之间的关系。该书对结构程序设计的基本原理、原则和方法作了透彻的说明，其影响是很广泛和深刻的。

由于以上论文和著作均未给结构程序设计这个名词以明确的定义，因此，后来许多人根据各自的认识，给出了很多不同的定义，不少人还以专文对这个名词的定义进行过讨论。Gries 于 1974 年 11 月在给CACM(Communication of the Association for Computing Machinery)的

一封信中，归纳了当时对结构程序设计(他以 SP 表示)的十三种不同的定义：

1. 它使程序设计返回到常识。

2. 它是主要程序员进行程序设计的一般方法。

3. 它是不用 GOTO 语句的程序设计。

4. 它是对于给定任务和其环境之间交互作用的一种控制过程，为的是使交互作用的次数成为给定任务的参数的某个线性函数。

5. 它是由顶向下的程序设计。

6. SP 理论是把大规模的复杂的过程转换成由少数基本的标准控制逻辑结构的循环、嵌套所组成的标准形式。

7. SP 是构思和编制程序的一种方式，目的是使程序易于理解和修改。

8. SP 的目的是通过理论和规范来控制复杂性。

9. SP 的特征不是无"GOTO"，而是"有结构"。

10. SP 的主要目标是使程序适宜于正确性证明。

11. SP 的基本概念是正确性证明。

12. SP 允许在设计的全过程中验证其正确性，并自动地形成"自明"和"自卫"的程序设计风格。

13. SP 不是万灵药，它由一些形式的符号所构成，目的是为了使程序员有条理地思考，而这并非程序员天生的品质，这是一种后天得到的训练，并应通过不断的有意识的努力去加强它。

Basili 还补充了如下一条：

14. 人们应追求程序的简单和精美，最简单的解答通常是最易理解的。

以上定义都从某个侧面反映了结构程序设计的一部分原理。Gries 认为 Hoare 的如下定义概括了 SP 的本质：

所谓结构程序设计是对人们思考方法的一种组织方式，它在合理的时间里把计算任务表达成易于理解的形式。

由此可见，结构程序设计强调的是方法论，是思考方法的组织形式，它注重于程序的易理解性。

Dijkstra 虽然从 GOTO 语句的问题开始提出结构程序设计的思想(这在§1.3 要详细讨论)，但他从来没有把删除 GOTO 语句包含在结构程序设计的定义之中。他在提及结构程序设计概念时，虽然也经常涉及程序正确性的证明，但所谓正确性证明并非指从公理出发的形式推导，而是指任何一种有足够说服力的证明(形式的或非形式的)，其实是指一种理解。实际上，结构程序设计是按照一组能提高程序的可阅读性与易维护性的规则而进行程序设计的方法。

综上所述，可以这样来定义：结构程序设计是一整套进行程序设计的准则，目的是为了使程序具有一种合理的结构，以使程序易于理解和维护，便于保证和验证程序的正确性。

在结构程序设计提出之前，普遍存在这样一种观点，认为程序设计是计算机和语言的问题，只要有了好的语言和大容量高速度的计算机，程序设计就不会有什么困难了。事实证明这种观点是片面的。随着计算机应用的日益推广，程序的规模日趋增大，软件的逻辑结构日趋复杂，程序员面临着日益严重的挑战。要解决好程序设计中效率低、质量差等问题的根本途径之一，还是要从程序设计工作的本身着手。结构程序设计的原理和方法是在"软件危机"的背景下产生的。特别是大型软件系统的逻辑结构很复杂，需要不断地修改。而且在研制软

件的过程中，涉及的人员多，候选的方案多，研制的周期长。因此，60年代以来，屡屡出现所谓“软件危机”。一些大型系统的软件研制，有的不能如期完成，有的经济上大大超过预算，有的满足不了用户预期的要求，有的难以修改和维护，有的甚至无法实现，彻底崩溃。软件的研制有时完全失去了控制；研制人员按各自的习惯工作，无章可循；管理人员无法估计所需的经费和时间；无法预料系统的成功与否；并且，失败的系统常常是无法挽回。有一个很典型的例子，就是IBM公司的OS/360系统的设计。研制该系统花了几千人年的努力，历尽艰辛，但是最后以失败告终。F.P.Brooks被称为“IBM360系统之父”，1964年，他是OS/360的负责人。他生动地回忆起当时的困难和混乱：“……象巨兽在泥潭中作垂死的挣扎，挣扎得越猛，泥浆就沾得越多。最后，没有一只野兽能逃脱淹没在泥潭里的命运，……，程序设计就象这样一个泥潭，……，一批批程序员在泥潭中挣扎，……，没人料到问题竟会这样棘手…。虽然负责OS/360的开发是失败的，但却是一种很有教育意义的经历，……花几百万美元去犯错误是非常沮丧的经历，但却也是值得记取的。”

软件工作者就是在这样严重的挑战前，提出了结构程序设计的原理、方法，实现了程序设计领域里的一场思想革命。正如Knuth所说：“一场革命正在我们如何写程序和教程序设计的方式方面发生。……一个人不可能读了那本关于结构程序设计的新书之后而不在自己的工作中引起变化。”《Software Practice and Experience》杂志1975年第一期也提出：“一场革命正在程序实践所依据的概念中发生，慢慢地，一种新的程序设计原理正在出现——其目的即在于能写出正确的(没有错的)程序，并且知道它们是正确的。”

我们知道，程序设计的目标是编写出逻辑上正确而又易于阅读理解和修改的程序，而结构程序设计正是顺应了这样的要求。60年代末和70年代初，人们在开发软件的实践中，提出并逐步实现了一系列结构程序设计的技术、方法。如：只用三种基本控制结构；由顶向下设计的逐步求精；主程序员组等。

我们知道，软件系统的生命期分为五个阶段：

(1) 分析阶段；(2) 设计阶段；(3) 编码阶段；(4) 测试阶段；(5) 运行阶段。

前四个阶段总称为开发期，最后阶段称为运行期。结构程序设计的技术主要是用于编码阶段的，这在70年代初国际上就有比较成功的经验。近年来，随着软件工程学的发展，结构程序设计的思想不断得到完善和拓广。70年代中期，人们开始重视建立整个软件系统的结构，于是结构化的思想前移到设计阶段，出现了用于设计阶段的结构设计方法，如SD方法，Jackson方法等。70年代后期，人们更进一步认识到在设计阶段前必须先对用户的要求进行分析，所以研究的重点又前移到分析阶段，提出了用于分析阶段的结构分析方法，即SA方法。SA、SD和SP方法互相衔接，配合使用，构成了较为完整的结构化系统开发的技术族。目前，用什么样的方法对软件进行测试和维护，已是软件工作者十分关心的问题。

结构程序设计的内容十分广泛，主要有密切相关的两方面的研究：

1. 程序设计方法。它包括研究用来开发易于理解的、正确的程序的有效方法，并探索在程序设计时如何清晰地进行思考。结构程序设计的一个基本观点是：为了得到一个可靠的程序，必须使程序有一个合理的结构；而为使程序有一个合理的结构，则首先应有一种好的程序设计方法，并对程序员有一个好的组织形式。

所谓合理的程序结构，从宏观上讲，是指系统程序的总体为分层结构，程序和其包含的子程序为模块结构。从微观上讲，是要研究如何使程序的静态结构和其执行的动态结构相一致，关于这方面的讨论比较集中于限制使用GOTO语句。所谓好的程序设计方法，比较成熟的是由顶

向下设计、逐步求精。关于程序设计人员的组织，普遍认为主程序员组是值得借鉴的组织形式。

2. 程序设计语言。语言直接影响到使用它的民族的思想和文化。在程序设计中，语言也同样重要，使用不同的语言编写的程序中，很少出现同样的错误。程序设计语言的研究和程序设计方法的研究是密切相关的。

从宏观上讲，系统程序的分层结构、程序及子程序的模块结构反映在语言上，表现为相应的结构程序语言的系统结构和程序的模块结构。而结构程序语言所表示的系统结构和程序的模块结构，也同样应该是程序设计人员组织形式的一种反映。从微观上讲，程序的静态、动态结构的一致性问题反映为结构程序语言应具有的控制结构问题；而结构程序语言所应具有的数据结构，则可以通过逐步求精的设计方法很好地得到实现。

此外，结构程序设计还涉及到程序设计的工具和环境等，限于篇幅，本书不可能均作深入讨论和详细介绍。我们选择了对提高程序设计水平关系比较密切的内容作一些简要的介绍，希望对我国的计算机工作者有所裨益。

第一部分主要是结构程序设计的原理和方法。第二部分介绍三种体现构结程序设计思想的有代表性的语言，即 Pascal、C、Ada 语言，使读者掌握如何用其来编写结构化程序。第三部分是程序测试和证明。这方面的内容很丰富，完全应该用一本专著来介绍，本书只选用一些常用的方法和技术进行讨论。第四部分是为已熟悉非结构化程序设计语言的读者编写的，我们提供了编写结构化 BASIC 程序，结构化 FORTRAN 程序和结构化 COBOL 程序的基本方法，以供参考。

§1.2 软件的易维护性和易理解性

1.2.1 软件质量

早期，人们对于软件质量的评价比较强调效率和正确性。近年来，随着软件规模的日益增大，软件逻辑性的日趋复杂，评价软件质量的主要依据是：

易维护性，易理解性，可靠性，效率。

让我们来分析结构程序设计对软件质量的影响。

一、易维护性

由于测试只能证明错误的存在而不能证明错误不存在，因此测试后投入运行的软件很可能还有错误，在运行阶段要不断发现并修改错误。另外，用户还会不断提出一些新的要求，系统的操作环境也可能发生变化，故投入运行的软件需要不断修改扩充。这就是通常所称的维护。软件的易维护性一般包括易阅读、易发现和改正错误，易修改扩充等含义。

由于软件规模的增大和逻辑性的复杂化，维护工作的困难性和工作量也越来越大。据某些典型课题的统计，在整个软件生命期中，维护工作量要占一半以上，如图 1—1所示。

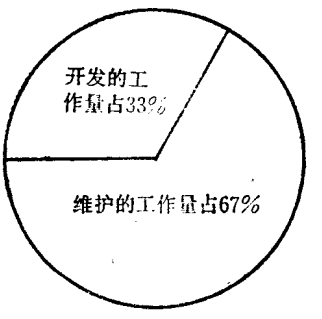

图 1—1 维护的工作量

同时，近年来由于软件研制的人工费用开销上升，因此维护在总投资中所占的比例迅速上升，目前已占软件研制费用总投资的一半以上，如图 1—2

所示。因此，易维护性是一个很重要的软件质量标准。

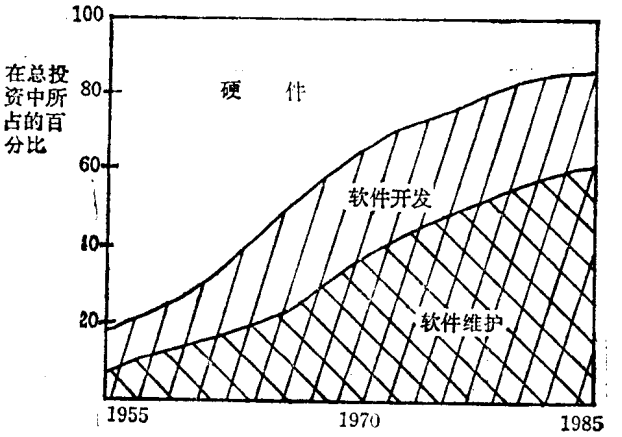

图 1—2　维护的费用

二、易理解性

软件经常要被人阅读，在测试、排错、修改时，都要由人来阅读程序，所以，易理解性也是一个重要的质量标准，易理解性也是易维护性的前提。

对于结构程序设计而言，易理解性和易维护性是其重要的目标。不少人都有这样的体会：读一个程序往往比写一个程序要花费更多的精力。在某种情况下，例如，相隔时间太长而且程序又写得不易阅读和理解时，即使程序员本人去读程序都是很费劲的事。因此，要注重程序的易读性。程序员要记住，今后会有人(包括本人)反复阅读你编制的程序，并试图沿着你现在的思路去理解程序的功能及其正确性。

比起非结构化的程序设计来，结构程序设计需要更多的思索，并对程序员有更多的约束，从而换得程序的易维护性和易理解性。从总体来讲，这是很值得的，也是很有必要的。

三、可靠性

由于软件规模增大将直接影响软件的可靠性，结构程序设计就是为了保证和便于验证程序的正确性而提出的，因此能提高软件的可靠性。

四、效率

效率是指编制成的软件系统是否能有效地使用计算机资源，如时间和空间等。值得注意的是，追求软件的效率与追求易维护性，易理解性和可靠性等往往是矛盾的。由于硬件价格下降，人工费用上升，因此目前人们宁可牺牲一些效率，而求程序有较好的易维护性、易理解性和可靠性。

综上所述，对于较大的软件系统，结构程序设计能获得高质量的软件，尤其是可以求得较好的易维护性、易理解性和可靠性。

1.2.2　程序设计风格

程序设计的所谓“风格”(style)，也有人称为“文体”，是指程序员使用计算机语言的习惯和方法。它也是结构程序设计的一个重要的概念。程序设计风格和程序的易读性关系很密切。

有相当长一段时间，很多人认为程序只是给机器执行而并非供人阅读的，所以只要程序的逻辑正确，能被计算机理解执行就足够了。但是实践证明，读程序是软件开发过程的一个重要组成部分，而且它往往比写程序所花的时间要多。正如我们以后要谈到的，目前读程序

仍然是检查程序错误的有效手段。逻辑正确但风格不好的程序，就无法供人阅读，所以也难以测试和维护。70年代初，随着结构程序设计思想的逐步形成，有人提出了程序设计风格这一新的观念，认为程序也是一种供人阅读的文章，只不过它不是用自然语言而是用程序设计语言编写的而已，因此，程序也有文体、风格的优劣。这很快被大家接受了。一个称职的程序员，应该能编写出结构合理、层次分明、思路清晰的程序来，而应该避免那种易读性差和容易出错的不好的风格。我们在此归纳了很多程序员在长期实践中总结的有关程序设计风格方面的经验。

一、程序的清晰性

我们务求程序清晰、简单和易于理解，为此牺牲一些效率或增加一些程序的篇幅也是值得的。

1. 使用有意义的变量名。这是程序风格中一条重要的原则。有时使用稍长一些的带说明性的变量名，能有效地提高程序的易读性，使需要的注解大大减少。一般讲，变量名最好由4～12个字符组成。当然，变量名太长也不好，这会妨碍人们对流程的注意，也容易写错。

2. 不要用相似的变量名。使用相似的变量名不仅不易阅读，还容易产生笔误。

3. 标识符中字母之间不要夹有数字，否则0、1、2和5容易同字母O、I、Z和S相混淆。

4. 不要把语言的关键词用作标识符。例如以下语句：

```
IF IF = THEN THEN THEN: = ELSE;
```

在语法上是正确的，但读起来很别扭。

5. 尽量不用临时变量。临时变量会增加程序的细节，从而影响程序的易读性。而且，临时变量使编译时产生的交叉引用表篇幅增大，使查阅不方便。引进临时变量有时会提高执行的效率，例如把

```
X: = A(I) + 1/A[I];
```

改写成

```
AI: = A[I];
X: = AI + 1/AI;
```

可能会节约些执行的时间，但影响了可读性。

6. 使用括号以减少差错。在程序设计语言中，关于算术运算符和逻辑运算符的优先级的规定很容易引起混淆，如 $-A**2$ 可能被理解成 $(-A)**2$，或 $-(A**2)$。在这类情况下，最好用括号把运算的顺序明确表达出来。

7. 一行只写一个语句，行首空格，保持程序整体的清晰性。最好把程序按逻辑深度的不同作锯齿状排列。

8. 在循环体内不要改变循环变量的值，否则会使循环难以理解，也难以验证是否进入死循环。

9. 尽量少用标号，不写不必要的语句标号。

二、语言的使用

程序设计风格的第二个重要方面是正确地使用语言。一般来讲，要理解并使用语言的全部特性，但应避开那些设计得不好的语言特性。

1. 要了解并充分使用语言那些简便而明确的特性。例如在PL/1语言中，要把数组A的元素全部置0，用不着使用DO循环，只用A=0就可以了。

2. 了解并使用程序的内部函数。不但要熟悉数值函数，还应熟悉非数值函数。例如