计算机基础教育丛书

FORTRAN 语言

FORTRAN 77 结构化程序设计

谭浩强 田淑清

清华大学出版社

FORTRAN 语言 FORTRAN 77 结构化程序设计

计算机基础教育丛书

FORTRAN 语 言

FORTRAN 77 结构化程序设计

谭浩强 田淑清 编著

清华大学出版社

内 容 简 介

本书是作者在原著《FORTRAN语言》一书的基础上，总结高等学校计算机基础教育的经验，精心修改而成的，特别适合于用作高等学校的教材。因为本书以 FORTRAN 77 代替了 FORTRAN Ⅳ，所以是《FORTRAN 语言》的更新版本。

本书是按照“算法＋数据结构＝程序”的思想组织的。将算法、数据结构、计算机语言、结构化程序设计方法四要素有机地结合了起来。针对初学者的特点，对各章内容作了精心安排，使读者能循序渐进。本书突出算法设计，采用先进的 N-S 结构化流程图。概念清晰，通俗易懂，例题丰富，每章后附有习题，便于进行教学。

本书可作为高等学校、电视大学程序设计课程的教材，也可作各类计算机学习班的教材，还可供初学者自学、参考。

计算机基础教育丛书
FORTRAN 语言
FORTRAN 77 结构化程序设计
谭浩强　田淑清　编著
☆
清华大学出版社出版
北京　清华园
北京联华印刷厂印刷
新华书店总店科技发行所发行
☆
开本：787×1092 1/16　印张：22　字数：560 千字
1990年3月第1版　1990年9月第2次印刷
印数：32001—82000
ISBN 7-302-00623-7/TP·217
定价：4.75 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，他们既熟悉本专业的业务，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

FORTRAN语言是世界上广泛流行的、最适用于数值计算的一种计算机语言，是世界上最早出现的高级语言。从1954年第一个FORTRAN语言版本问世到现在，已经有35年了。但是它并不因古老而显得过时，而是随着时间的推移不断发展。前一个时期国内普遍使用的版本是1972年国际标准化组织（ISO）公布的“完全的（一级）FORTRAN”，它相当于美国标准化协会（ANSI）1966年通过的标准FORTRAN（X3.9—1966），或在美国流行的FORTRAN Ⅳ。FORTRAN 77是ANSI在1978年公布的、到目前为止最新版本的FORTRAN语言。

学习计算机语言的目的是利用它去编写计算机程序及解决各种问题。计算机语言只是一种工具。进行程序设计的关键是设计解题的算法。所谓算法就是解题的具体步骤。计算机程序就是用某一种计算机语言表示的算法。在设计好算法之后，就可以用任何一种计算机语言来表示它。因此，学习程序设计不应当只局限于一种语言，而应能举一反三，能根据需要选用某种特定的语言。因此，本书的重点不放在介绍FORTRAN语言的具体语法上，而是从算法入手，使读者养成这样一个习惯：在拿到一个问题之后，首先应当考虑算法，而不是马上动手写程序。本书在写法上就是以算法设计贯彻始终的。这样，读者在学完本书以后，当需要改用另一种计算机语言编程序时，便不会感到太困难了。

程序处理的对象是数据。数据与数据之间会存在某种形式联系，这就是数据结构。在程序设计时，除了要考虑算法外，还要考虑并选择适当的数据结构。对于不同的数据结构，要用不同的算法去处理。著名的计算机科学家沃思有一个有名的公式：

算法＋数据结构＝程序

本书介绍数据结构的概念以及在FORTRAN中允许使用的数据结构，并按照上述公式展开程序设计。

十几年前，在国际上兴起了“结构化程序设计方法”。它的出现改变了过去在程序设计中无规范可循而由程序设计者随心所欲的状况，程序不再是表现个人技巧的“艺术品”，而应当是采用“工程”的方法，按照一定的规范生产出的“产品”。读者应很好地掌握这种设计方法。

FORTRAN 77对FORTRAN 66的一个重要改进就在于增加了一些适合于结构化程序的语句，从而避免了使用GOTO语句进行无规律的转移（当然，FORTRAN 77还有一些弱点，它不是彻底的结构化语言。现在有一些计算机系统采用的FORTRAN 77又增加了DO WHILE，DO UNTIL等语句，使它的结构化程度更高了）。

综上所述，组织一个程序的四要素是：算法、数据结构、计算机语言、结构化程序设计方法。本书将它们有机地结合起来，使之浑然一体。这是本书的一个重要特点。

在前几年，我们曾编写过几本FORTRAN语言的教科书。它针对初学者的特点和认识规律，有的放矢，循序渐进，把科学性与实用性、易读性结合起来，受到读者的欢迎。仅清华大学出版社81年出版的《FORTRAN语言》发行量累计已达70万册以上，创国内外同

类书的最高记录。考虑到国内多数计算机系统已配置了FORTRAN 77编译系统，同时结构化程序设计方法已在国内广泛使用，因此有必要对《FORTRAN语言》一书进行全面的修订。本书就是在《FORTRAN语言》基础上重新编写的。它仍然保持了《FORTRAN语言》一书的风格，使初学者便于理解和接受。同时，按照前面所叙述的原则，我们重新组织了教材体系，使本书具有新的特点。即使是从未学过其它计算机语言的读者，也能看懂本书，并掌握其中的内容。

本书每一章中都介绍了一定数量的程序，每个程序都体现了一定的算法。希望读者认真阅读这些程序，并从中学习编程序的方法，做到举一反三，能解决其它类型的问题。书中的程序是按FORTRAN 77全集的规定编写的。应当说明，有的微型计算机使用的是FORTRAN 77子集，不具备FORTRAN 77全集的某些功能，在不同计算机系统上进行编辑、编译、连接和运行一个FORTRAN程序的方法将略有不同。因此，读者应了解在本系统上使用FORTRAN的有关规定。

本书分为两个单元共13章。第一单元包括第一章到第八章，是FORTRAN的基本部分。有了这一部分的基础就可以编写出简单的FORTRAN程序。第八章是常用算法举例，它综合应用了前七章的知识，也作为前七章的小结。课程到此为一个阶段，应进行期中小结或测验，以巩固此阶段的学习成果。第二单元包括第九到第十三章，是在第一单元的基础上进一步提高和深入的部分，包括一些比较复杂的算法和程序。在学习完全书之后，读者应能独立地编写出各种典型问题的一般程序。

本书第一章到第八章由谭浩强教授编写，第九到第十三章由田淑清副教授编写。在本书的编写过程中，我们得到了全国高等学校计算机基础教育研究会许多专家的关心和帮助，许多读者寄来了热情洋溢的信件，我们在此表示衷心的感谢。

为了帮助使用本书的教师和学生更好地掌握程序设计的方法与技巧，我们还编写了《FORTRAN习题集与上机指导》（由高等教育出版社出版）。书中有近400个FORTRAN习题与例题（包括本书的大部分习题解答），以及在几种典型的计算机系统上编译和运行FORTRAN程序的方法，可以作为学习参考书，也可作习题库或试题库使用。

由于水平所限，书中难免有错误，敬请批评指正。

编著者

1989.3

目　录

第一单元

第一章 算 法

要利用计算机处理问题，需要编写出使计算机按人们意愿工作的计算机程序。所谓程序就是一组计算机指令。每一个指令使计算机执行特定的操作。因此，每一个学习计算机知识的人以及希望利用计算机进行某项工作的人，都应当学习如何进行程序设计。

为了有效地进行程序设计，应当至少具有两个方面的知识，即：（1）掌握一门高级语言的语法规则；（2）掌握解题的方法和步骤，也就是说，在拿到一个需要求解的问题后，怎么能将它分解成一连串的操作步骤。

计算机语言只是一种工具。光学习语言的规则还不够，最重要的是学会针对各种类型的问题，拟定出有效的解题方法和步骤。这就是本章所要讨论的问题——算法。有了正确而有效的算法，可以利用任何一种计算机高级语言编写程序，使计算机进行工作。因此，设计算法是程序设计的核心。希望读者对它给予足够的注意。

本书的叙述，是以算法为核心而展开的，目的不仅在于介绍一种计算机语言，而在于使读者能利用计算机语言编写出所需解决的问题的程序。

1.1 算法的概念

做任何事情都有一定的步骤。例如，你要看电影，就要先买票，然后按时到电影院，进场，找座位，坐下，看电影，退场，等等。你要考入大学，首先要填报名单，交报名费，拿到准考证，按时参加考试，得到录取通知书，到指定学校报到注册等。这些都是按一系列的顺序进行的步骤，缺一不可，次序错了也不行。因此，我们从事各种工作和活动，都必须事先想好进行的步骤，以免产生错乱。事实上，在日常生活中，由于已养成习惯，所以人们并不意识到需要事先设计“行动步骤”。例如吃饭、上学、打球、做作业等，事实上都是按照一定的规律进行的，只是人们不必每次都重复考虑它而已。

不要认为只有“计算”的问题才有算法。广义地说，为解决一个问题而采取的方法和步骤，称为“算法”（Algorithm）。例如，描述太极拳动作的图解，就是“太极拳的算法”。一首歌曲的乐谱，也可以称为该歌曲的算法，因为它指定了演奏该歌曲的每一个步骤，按照它的规定就能演奏出预定的曲子。

对同一个问题，可以有不同的解题方法和步骤。例如，求$1+2+3+\cdots+100$，即$\sum_{n=1}^{100} n$，有的人可能先进行$1+2$，再加3，再加4，一直加到100，而有的人采取这样的方法：$\sum_{n=1}^{100} n = 100 + (1+99) + (2+98) + \cdots + (49+51) + 50 = 100 + 50 + 49 \times 100 = 5050$。还可以有其它的方法。当然，方法有优劣之分。有的方法只需进行很少的步骤，而有些方法则需要较多的步骤。一般说，希望采用方法简单，运算步骤少的方法。因此，为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法。

要完成一件工作，包括设计算法和实现算法两个部分。例如，作曲家创作一首曲谱就是设计一个算法，但它仅仅是一个乐谱，并未变成了音乐，而作曲家的目的是希望使人们听到悦耳动人的音乐。由演奏家按照乐谱的规定进行演奏，就是“实现算法”。一个菜谱是一个算法，厨师炒菜就是在实现这个算法。设计算法的目的是为了实现算法。因此，我们不仅要考虑如何设计一个算法，也要考虑如何实现一个算法。

本书所关心的当然只限于计算机算法，即计算机能执行的算法。例如，让计算机算1×2×3×4×5，或将100个学生的成绩按高低分次序排列，是可以做到的，而让计算机去执行“替我理发”或“煎一份牛排”，是不可能的（至少目前如此）。

计算机算法可分为两大类别：数值运算算法和非数值运算算法。数值运算的目的是求数值解，例如求方程的根、求一个函数的定积分等，都属于数值运算范围。非数值运算包括的面十分广泛，最常见的是用于事务管理领域，例如图书检索、人事管理、行车调度管理等。计算机在非数值运算方面的应用远远超过了在数值运算方面的应用。由于数值运算有现成的模型，可以运用数值分析方法，因此对数值运算的算法的研究比较深入，算法比较成熟。对各种数值运算都有比较成熟的算法可供选用。常常把这些算法汇编成册（写成程序形式），或者将这些程序存放在磁盘或磁带上，供用户调用。例如有的计算机系统提供“数学程序库”，使用起来十分方便。而非数值运算的种类繁多，要求各异，难以规范化，因此只对一些典型的非数值运算算法（例如排序算法）作比较深入的研究。其它的非数值运算问题，往往需要使用者参考已有的类似算法重新设计解决特定问题的专门算法。

本书不可能罗列所有算法，只是通过一些典型的算法的讨论，帮助读者了解如何设计一个算法，推动读者举一反三。希望读者通过这些例子了解怎样提出问题，怎样思考问题，怎样表示一个算法。

1.2 简单算法举例

【例 1.1】 求1×2×3×4×5。

可以用最原始的方法进行：

步骤1：先求1×2，得到结果2。

步骤2：将步骤1得到的结果2再乘以3，得到结果6。

步骤3：将6再乘以4，得24。

步骤4：将24再乘以5，得120。这就是最后的结果。

这样的算法虽然是正确的，但太繁琐。如果要求1×2×…×100，则要写999个步骤，显然是不可取的。而且每次都直接使用上一步骤的数值结果（如2,6,24等），也不方便。应当能找到一种通用的表示方法。

可以设两个变量：一个变量代表被乘数，一个变量代表乘数。不另设变量存放乘积结果，而直接将每一步骤的乘积放在被乘数变量中。今设T为被乘数，I为乘数。同时用循环来表示算法，可以将算法改写如下：

S1：使T=1

S2：使I=2

S3：使T×I，乘积结果仍放在变量T中，可表示为：T×I⇒T

S4：使I的值加1，即I+1⇒I。

S5：如果I不大于5，返回重新执行S3，以及其后的步骤S4，S5；否则，算法结束。最后得到T的值就是5!的值。

上面的S1，S2…代表步骤1，步骤2…。S是Step(步)的缩写。这是写算法的习惯用法。

请读者仔细分析这个算法，是否能得到预期的结果。显然这个算法比前面的一种形式的算法简练。

如果题目改为：求1×3×5×7×9×11。

算法只需作很少的改动即可：

S1：1⇒T

S2：3⇒I

S3：T×I⇒T

S4：I+2⇒I

S5：若I≤11，返回S3。否则，结束。

可以看出用这种方法表示的算法具有通用性、灵活性。S3到S5组成一个循环，在实现算法时要反复多次执行S3，S4，S5等步骤，直到某一时刻，执行S5步骤时经过判断，乘数I已超过规定的数而不返回S3步骤为止。此时算法结束，变量T的值就是所求结果。

由于计算机是高速进行运算的自动机器，实现循环是轻而易举的，所有计算机高级语言中都有实现循环的语句，因此，上述算法不仅是正确的，而且是计算机能实现的较好的算法。

请读者仔细分析循环结束的条件，即S5步骤。如果在求1×2×…×11时，将S5步骤写成：

S5：若I<11，返回S3。

这样会有什么问题？得到什么结果？

【例 1.2】 有50个学生，要求将他们之中成绩在80分以上者打印出来。

用N表示学生学号，N_1代表第一个学生学号，N_i代表第i个学生学号。用G代表学生成绩，G_i代表第i个学生成绩，算法可表示如下。

S1：1⇒i

S2：如果$G_i \geqslant 80$，则打印N_i和G_i，否则不打印。

S3：i+1⇒i

S4：如果$i \leqslant 50$，返回S2，继续执行。否则，算法结束。

本例中，变量i作为下标，用它来控制序号（第几个学生，第几个成绩）。当i超过50时，表示已对50个学生的成绩处理完毕，算法结束。

【例 1.3】 将2000～2500年中每一年是否闰年打印出来。

闰年的条件是：（1）能被4整除，但不能被100整除的年份都是闰年；（2）能被100整除，又能被400整除的年份是闰年。如1989，1900年不是闰年，1992，2000年是闰年。

设Y为年份，算法可表示如下：

S1：2000⇒Y

S2：若Y不能被4整除，则打印Y“不是闰年”。然后转到S5。

S3：若Y能被4整除，不能被100整除，则打印Y“是闰年”。

S4：若Y能被100整除，又能被400整除，打印Y“是闰年”。

S5：Y+1⇒Y

S6：当Y≤2500时，转S2继续执行，如Y>2500，算法停止。

在这个算法中，采取了多次判断，先判断Y能否被4整除，如不能，则Y必然不是闰年。如Y能被4整除，并不能马上决定它是否闰年，还要看它能否被100整除。如不能被100整除，则肯定是闰年（例如1990年）。如能被100整除，还不能判断它是否闰年，还要被400整除，如果能被400整除，则它是闰年，否则不是闰年，在这个算法中，每做一步，都分别分离出一些年份（为闰年或非闰年），逐步缩小范围，使被判断的范围愈来愈小，直至最后，见图1.1示意。

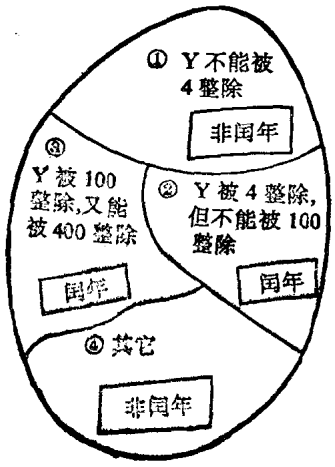

图 1.1

从图1.1可以看出：“其它”这一部分，包括能被4整除，又能被100整除，而不能被400整除的那些年份（如1900）。

在考虑算法时，应当仔细分析所需判断的条件，如何一步一步缩小被判断的范围。有的问题，判断的先后次序是无所谓的，而有的问题，判断条件的先后次序是不能任意颠倒的，读者可根据具体问题决定其逻辑。

【例 1.4】 求$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}\cdots+\frac{1}{99}-\frac{1}{100}$

算法可以表示如下：

S1：sum=1

S2：deno=2

S3：sign=1

S4：sign=(-1)×sign

S5：term=sign×(1/deno)

S6：sum=sum+term

S7：deno=deno+1

S8：若deno≤100 返回S4，否则算法结束。

本例中用有含义的单词作变量名，以使算法更易于理解。sum表示累加和，deno是分母denominator的缩写，sign代表数值的符号，term代表某一项，读者理解这个算法是不困难的。

【例 1.5】 对一个大于或等于3的正整数，判断它是不是一个素数。

所谓素数，是指除1和该数本身之外，不能被其它任何整数整除的数。例如，13是素数。因为它不能被2，3，4，…，12整除。

判断一个数N(N≥3)是否素数的方法是很简单的：将N作为被除数，将2到(N-1)各个整数轮流作为除数，如果都不能被整除，则N为素数。

算法可以表示如下：

S1：输入N的值。

S2：I=2

S3：N被I除，得余数R。

S4：如果R=0，表示N能被I整除，则打印N“不是素数”，算法结束。否则执行S5。

S5：I+1⇒I

S6：如果I≤N−1，返回S3。否则打印N“是素数”。然后结束。

实际上，N不必被2到（N−1）的整数除，只需被2到$\frac{N}{2}$间整数除即可，甚至只需被2到$\sqrt{N}$之间的整数除即可。例如，判断13是否素数，只需将13被2，3除即可，如都除不尽，N必为素数。S6步骤可改为：

S6：如果I≤$\sqrt{N}$，返回S2，否则算法结束。

通过以上几个例子，可以初步了解怎样写一个算法。

1.3 算法的特性

一个算法应该具有以下特点：

1. 有穷性。一个算法应包含有限的操作步骤，而不能是无限的。例如上一节中例1.4的算法，如果将S8步骤改为“若deno>0返回S4”，则循环永远不会停止。这不是有穷的步骤。

事实上，“有穷性”往往指在合理的范围之内。如果让计算机执行一个历时1000年才结束的算法，这虽然是有穷的，但超过了合理的限度，人们也不把它视作有效算法。究竟什么算“合理限度”，并无严格标准，由人们的常识和需要而定。

2. 确定性。算法中的每一个步骤都应当是确定的，而不应当是含糊的，模棱两可的。例如，有一个健身操的动作要领，其中有一个动作：“手举过头顶”，这个步骤就是不确定的，含糊的。是双手都举过头？还是左手？或右手？不同的人可以有不同的理解。算法中的每一个步骤应当不致被解释成不同的含义，而应是十分明确无误的。如例1.5中的S3步骤如果写成“N被一个整数除，得余数R”，这也是“不确定”的，它没有说明N被哪一个整数除，因此难以执行。也就是说，算法的含义应当是唯一的，而不应当产生“歧义性”。所谓“歧义性”是指可以被理解为两种（或多种）的可能含义。

3. 有零个或多个输入。所谓输入是指在执行算法时需要从外界取得必要的信息。例如，在执行例1.5算法时，需要输入N的值，然后判断N是否素数。也可以有二个或多个输入，例如，求两个整数m和n的最大公约数，则需要输入m和n的值。一个算法也可以没有输入，例如，例1.1在执行算法时不需要输入任何信息，就能求出5！。

不要把指定算法误认为是“输入”。输入是指在执行指定的算法时需要从外界获取的信息。

4. 有一个或多个输出。算法的目的是为了求解，“解”就是输出。如例1.5求素数的算法，最后打印出的“N是素数”或“N不是素数”就是输出的信息。没有输出的算法是没有意义的。

5. 有效性。算法中的每一个步骤都应当能有效地执行，并得到确定的结果。例如，如果B=0，则执行A/B是无法有效执行的。

对于那些不熟悉计算机的人来说，他们可以只使用别人已设计好的现成算法，只需根据算法的要求给以必要的输入，就能得到输出的结果。对他们来说，算法如同一个“黑箱子”一样，他们可以不了解“黑箱子”中的结构，只是从外部特性上了解算法的作用，即可方便地使用算法。例如，对一个“输入三个数，求其中最大值”的算法，可以用图1.2表示，只要输入a,b,c三个数，执行算法后就能输出其中最大的数，但对于程序设计人员来说，必须会设计算法，并且根据算法编写程序。

图 1.2

1.4 怎样表示一个算法

为了表示一个算法，可以用不同的方法。常用的有：自然语言；流程图；结构化流程图；伪代码；PAD图等。

1.4.1 用自然语言表示算法

我们在1.2节中介绍的算法是用自然语言来表示的，自然语言就是人们日常使用的语言，可以是汉语或英语或其它文字。用自然语言表示通俗易懂，但文字冗长，容易出现“歧义性”。自然语言表示的含义往往不大严格，要根据上下文才能判断其正确含义。而且用自然语言来描述包含分支和循环的算法，不很方便(例1.5的算法)。因此，除了那些很简单的问题以外，一般不用自然语言描述算法。

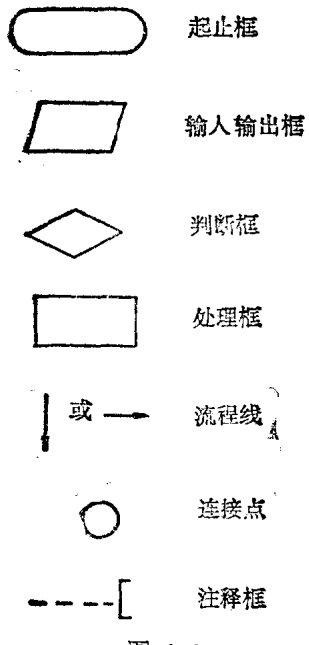

图 1.3

1.4.2 用流程图表示算法

流程图是用一些图框来表示各种类型的操作。用图形表示算法，直观形象，易于理解。

美国国家标准化协会ANSI (American National Standard Institute)规定了一些常用的流程图符号(见图1.3)，已为世界各国程序工作者普遍采用。图1.3中菱形框的作用是对一个给定的条件进行判断，根据给定的条件是否成立决定如何执行其后的操作。它有一个入口，二个出口。见图1.4示意。连接点（小圆圈）是用于将画在不同地方的流程线连接起来。如图1.5中有两个以1标志的连接点（在连接点圈中写上“1”），它表示这两个点是连接在一起的，相当于一个点一样。用

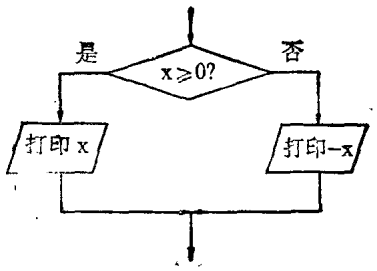

图 1.4

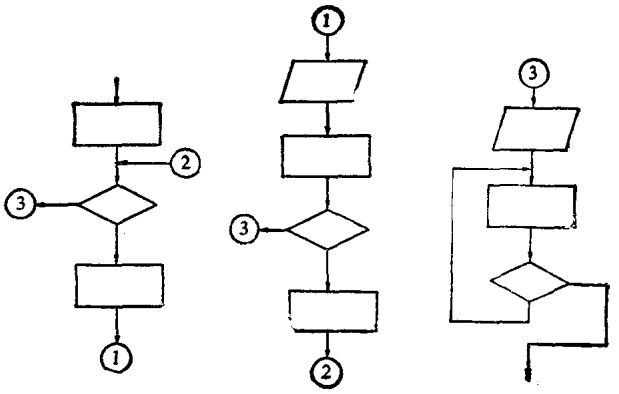

图 1.5

连接点，可以避免流程线的交叉或过长，使流程图清晰。注释框不是流程图中必要的部分，不反映流程和操作，只是为了对流程图中某些框的操作做必要的补充说明，以帮助阅读流程图的人更好地理解流程图的作用。

我们对1.2中所举的几个算法例子，改用流程图表示。

【例 1.6】 将例1.1求5!的算法用流程图表示，流程图见图1.6。

如果需要将最后结果打印出来，可以在菱形框的下面再加一个输出框，见图1.7。

【例 1.7】 将例1.2的算法用流程图表示。将50名学生中成绩在80分以上者的学号和成绩打印出来。

见图1.8，菱形框两侧的“Y”和“N”代表“是”(yes)和“否”(no)。在此算法中没有包括输入50个学生数据的部分。如果包括这个输入数据的部分，流程图如图1.9所示。

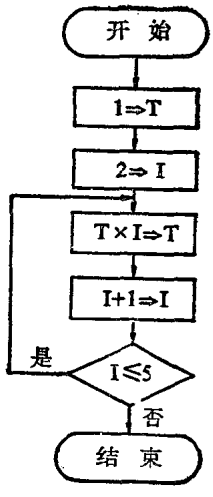

图 1.6

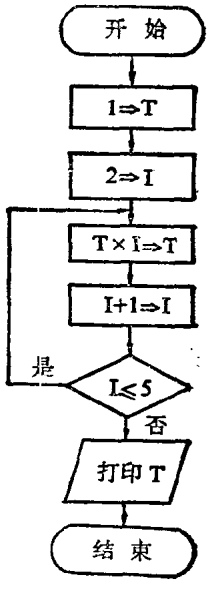

图 1.7

【例 1.8】 将例1.3判定闰年的算法用流程图表示。

见图1.10，显然，用图1.10表示算法要比用文字描述算法逻辑清晰、易于理解。

请读者考虑，如果例1.3所表示的算法中，S2步骤内没有最后“转到S5”这一句话，而只是：

S2：若Y不能被4整除，则打印Y“不是闰年”。

这样就意味着执行完S2步骤后应执行S3步骤，而不论S2的执行情况如何。请读者画出相应的流程图。请思考这样的算法在逻辑上有什么错误？从流程图上，是很容易发现逻辑上的错误的。

【例 1.9】 将例1.4的算法用流程图表示。即：求$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}\cdots+\frac{1}{99}-\frac{1}{100}$

见图1.11。