FORTRAN77
语言和风格

〔美〕M.J. 麦钱特 著 王懋江 译

FORTRAN77的结构化入门

新时代出版社

FORTRAN77 语言和风格 FORTRAN77 的结构化入门

FORTRAN 77

语言和风格

FORTRAN 77的结构化入门

〔美〕M.J. 麦钱特 著
王懋江 译

新时代出版社

内 容 简 介

本书从结构程序设计的观点出发全面地讲述了 FORTRAN 77，全书共十章，分别叙述了FORTRAN 77的常数、变量、表达式、控制语句、字符数据、数组、辅程序、格式输入输出、文件等。各章节反复给出了结构流程图，强调了自顶向下的程序设计，所提出的风格模型指出了如何写出好的程序，并列出了该语言诸成分应遵从的规则。

全书叙述深入浅出，文字通俗。既适合于大专院校作FORTRAN语言的教材，也是一般工程技术人员学习和使用FORTRAN 77较好的参考书。

LANGUAGE AND STYLE
A Structured Guide to Fortran 77
MICHAEL J. MERCHANT
Wadsworth Publishing Company

FORTRAN 77
语 言 和 风 格
FORTRAN 77的结构化入门
〔美〕M. J. 麦钱特 著
王慧江 译

新时代出版社出版发行
（北京市海淀区紫竹院南路23号）
（邮政编码 100044）
新华书店经售
国防工业出版社印刷厂印刷
787×1092毫米 16开本 22印张 510千字
1985年6月第1版 1990年9月北京第2次印刷
印数：12,001—14,000册

ISBN 7-5042-0118-9/TP·4 定价：9.00元

译 者 的 话

这本书从结构程序设计的角度来讲解FORTRAN 77，叙述深入浅出，密切联系程序设计实际。该书把FORTRAN 77的精华，如字符型数据、字符表达式、格式输入输出、文件系统及其中的直接存取文件等，都通过具体应用实例清楚地予以叙述。此书是一本学习FORTRAN 77的很好的入门书，也是一本学习结构程序设计的教科书。

另一方面，正如书名所表明的，这本书一再强调了程序设计的风格问题，多次给出了程序设计的风格模型。程序设计的风格问题在国内可能还没有引起足够的重视，但确实是一个值得重视的问题。好的程序设计风格会加快软件的研制、交流和推广。一个软件工作者从设计第一个程序起就应坚持好的程序设计风格。因此译者认为这也是本书值得称道的地方。

本书的著者前言及第一章由程虎同志译出，其余部分由王懋江译出。在翻译中，对原书的个别错误作了订正，少数篇幅作了节译。译者曾就书中的个别难点请教了译审张泽渊同志，吴振益、吕汇川等同志也给了不少帮助，在此一并表示感谢。

前　言

作者头脑里是带着两个问题来写这本书的。第一，要教好新 FORTRAN 语言，什么方法最好？第二，使用 FORTRAN 来教结构程序设计，什么方法最好？

在解答第一个问题的过程中，经常是按照 1978 年 FORTRAN语言标准的特点来进行选择课题以及安排所要讲解的顺序的。例如，字符数据已和数值数据处于同等地位，所以字符数据在课文里早就谈到了。增加字符数据不仅扩展了语言的能力，而且教起来也比较容易，因为对学生来讲，字符数据的应用范围极广，并且在应用时，用不着高等数学就能阐明重要的程序设计概念。增加了表控输入输出，学生们无需精通 FORMAT 语句的非常复杂的技术细节，就能编写出程序。

在新标准之前，用 FORTRAN 来教结构程序设计是有困难的，因为这种语言缺乏必要的控制语句。现在增加了 IF-THEN-ELSE 结构，就能够教学生写出清晰的结构程序。

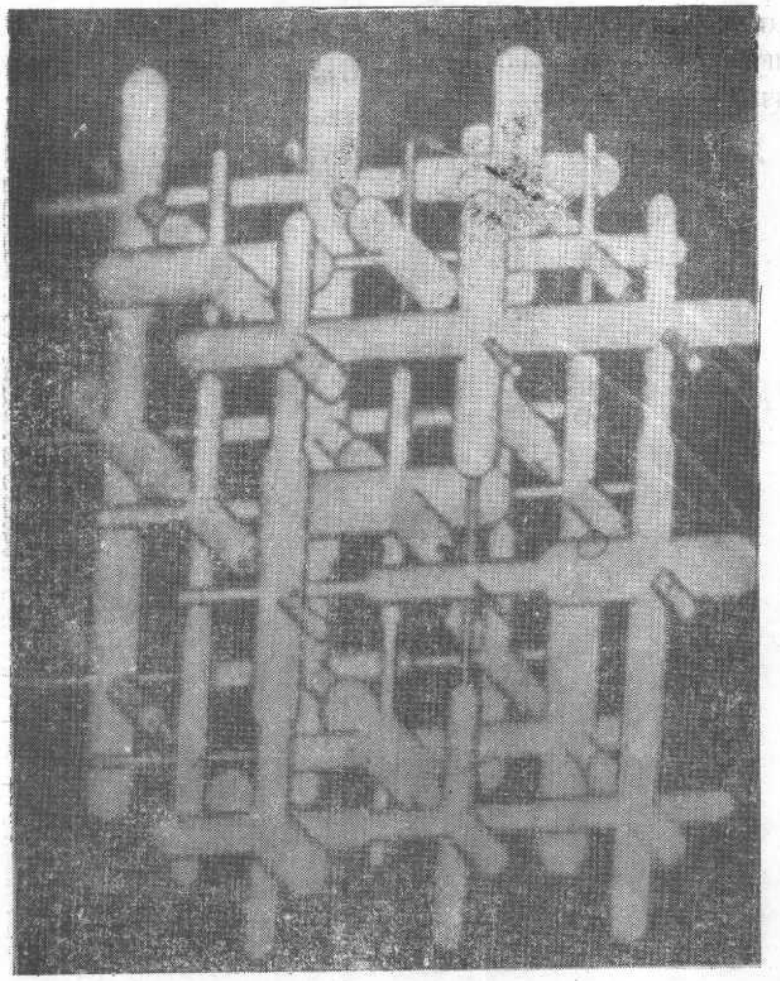
全书把程序设计语言的概念和程序设计风格的概念结合了起来。每章的风格模型节，实际上指出了怎样写好的程序和怎样应用 FORTRAN 语言。再三地强调了自顶向下设计和结构程序设计。

我们坚信程序设计这门课最好通过实践来学习。因此第一章和第二章所安排的内容是让学生从一开始就能正确地运行程序。在第一章，学生能看到从自顶向下设计到最后完成代码这样一个完整的过程。第二章指导学生弄懂在计算机上运行程序的细节。

用一学期的时间可把全书教完，介绍使用 FORTRAN 进行程序设计。假如学生们已经熟悉其它一种程序设计语言，则可以不必讲第一章和第二章，或者讲授时一带而过就可以了。如删去已标明为可选的节，则第一至第七章可作为 FORTRAN 的简明教程，或作为数据处理引论课程的补充教材。

书中任何不足之处，概由本人负责。我对那些阅读过我的原稿和提出了宝贵改进意见的人，J. H. 克伦肖（J. H. Crenshaw）、西肯塔基（Western Kentucky）大学；H. A. 埃特林格（H. A. Etlinger），罗彻斯特（Rochester）理工学院；K. 弗兰科夫斯基（K. Frankowski），明尼苏达（Minnesota）大学；K. 盖勒（K. Geller），德雷克塞尔（Drexel）大学；R. 弗赖伊（R. Frye），中央密执安（Central Michigan）大学；M. 卢克（M. Luker），明尼苏达-杜鲁斯（Minnesota-Duluth）大学；F. 奥本赫（F. Oppacher），康考尔迪亚（Concordia）大学；F. G. 沃尔特斯（F. G. Walters），密苏里-罗拉（Missouri-Rolla）大学，表示由衷的感谢。

M. J. 麦钱特



目　录

第一章　计算机、算法和程序设计

1.1 引　　言

在科学幻想小说、电影及动画片里，计算机常常被描绘得很象超智慧的人。在这种幻想作品里，计算机用流利的英语讲话，运用独立的判断力来解决问题，并能立即从存储库里取出任何问题的有关数据。这是一种令人兴奋的幻想。计算技术研究的一个激动人心的方面，就是有朝一日终会实现这种幻想——或许就在你的有生之年实现。

但是，要计算机具有真正的智慧，这还有待于将来。当今计算机速度之快是惊人的。它们能按照非常复杂的指令系统行事，但是它们毕竟是机器——在某些方面，只不过是相当简单的机器而已。计算机实际能做的，是按照简单的命令行事，而这些命令先要由程序员细心地设计出来，并用象 FORTRAN 这样的程序设计语言编写。

你从这本书中要学会两件事。第一件是怎样用计算机解决问题——即，怎样通过给计算机编制指令让它来做你要它做的工作。尤其是，要学会设计和描写**算法**，算法是计算机能执行的过程。

在这一章，我们要解释什么是算法，怎样用一种称为**流程图**的图来表示算法，以及怎样使用**自顶向下设计**方法来构造算法。

在设计好用来解决问题的算法以后，必须使用计算机能理解的语言来表示算法。**计算机程序**就是用象 FORTRAN 这样的程序设计语言编写的算法。所以，你要学会的第二件事就是怎样编写 FORTRAN 程序，这是与第一件事结合在一起的。

第一章讲的是 FORTRAN 语言入门。在第二章，将看到怎样在计算机上运行 FORTRAN 程序。

1.2 算　　法

人们彼此之间发命令时，常常是不精确的。诸如“去买一个面包”这样一条简单的命令，就要求做上百个判定：应该从前门还是从后门出去？应该左转弯还是右转弯？商店在哪一条路？面包在何处？是要全麦的、酸的、还是黄蒿黑麦的？因为人有理解力和推理能力，所以能够了解一般的、二义性命令的实际含意。而计算机跟人没有共同意识，故当给计算机编写要执行的过程时，每条指令都必须清楚明白。

假定你要用计算器解一个数学问题。你没有计算器，而一个朋友有，于是你就打电话给她，请她帮忙。可是她不在家，但她弟弟在，她弟弟不太懂数学，如果能精确地告诉他要做些什么，那么他就能使计算器工作。现在你必须说明怎样用一些指令来解你的问题，这些指令要如此地精确和无二义性，以致他不会误解。你可以说：“写入数56.2，按加法键，写入数475.3，按加法键，写入数11.63，按等于键，按照计算器顶部的指示灯读给我答数。”这是一个用自然语言表达的算法。

算法[1]是一步一步地解题的过程。正确的算法必须满足下列三个条件：

1．算法中的每一步必须是一条能执行的指令。

2．各步的顺序必须精确地确定。

3．算法最后必须能结束。

算法不必一定是为计算机编写的。例如，你可以用纸和铅笔来执行这些指令。烹调书可以对烘焙巧克力薄片小甜饼给出一个“算法”。“烘焙到做好”，这条指令对厨师是有意义的，因此它满足算法的第一个条件。

如果你只要告诉计算机“解下列问题……”就使得机器执行，那倒是方便的。然而，计算机仅有一个由一百条左右、能用电子线路执行的基本命令所组成的**指令系统**。这些命令类似于通过按键给计算器的命令。例如，你可以告诉计算机，让它加两个数。正确算法的第一个条件意味着，对一台计算机写算法时，每一步必须是计算机能够通过这些基本指令实现的某件事。

执行算法中的指令时，机器每次完成一条指令。正确算法的第二个条件意味着，算法必须精确地指明完成指令的顺序。

第三个条件意味着，不许算法永远进行下去。因此，下列过程不是一个算法。

计数过程

第1步　令N等于0。

第2步　N加1。

第3步　转向第2步。

如果你试图用一台计算机执行这个过程，那么，在理论上，这台机器将永远运行下去。另一方面，下列过程是一个算法，因为它最后将结束。

到一百万计数的算法

第1步　设N等于0。

第2步　N加1。

第3步　如果N小于1,000,000，则转向第2步；否则，停止。

1.3　自顶向下设计

当按照一个简单问题的算法进行工作时，只要你对那个问题想一会儿，就立即会得到它的解答。但是，实际的计算机应用很少这样简单。职业程序员写的程序，通常由成千条计算机指令组成。设计这样的程序和设计一个有成千个零件的机器一样复杂。为了使这样的程序能工作，所有各部分必须一起装配在一个有组织的体制中。

自顶向下设计是设计算法问题的一种途径。它是既能用来组织你的工作，也能组织算法的一种方法。

在某些方面，设计一个算法类似于写一篇散文。在写散文时，首先谈你对所要涉及事物的一般想法。然后将这些思想组织起来，并选择词汇，以便向读者表达你的意思。在设计一个算法时，对于要干什么，首先也要有一个总的想法。然后，必须组织这些想法，并给计算机选择一串正确的指令，使它执行你所要求的动作。程序设计和写作一样，

[1] 算法一词的英文 *algorithm* 来自波斯数学家的名字al-Khowarizmi（公元825年）。可参见D.E.克努特《计算机程序设计技巧》第一卷的第一页，国防工业出版社，1980。——译者

感到困难的是，常常不知从何处入手。

着手写散文的好方法是先列一个提纲。首先，陈述要涉及的主要论题，如下列例子中那样：

葛底斯堡（*Gettysburg*）演讲●

Ⅰ．国家的概念

Ⅱ．当前的南北战争

Ⅲ．我们今天在这里集会的目的

Ⅳ．我们对未来的决心

这个简单的提纲为散文提供了一个架子，然后把整个段落和句子充实进去。总的结构确定以后，余下的问题就是更详细地去着力阐述这些论题。你可以在每一个论题下面加小标题，例如：

葛底斯堡演讲

Ⅰ．国家的概念

A．美国是在八十七年前建立的。

B．它孕育于自由之中。

C．它信奉人人生来皆平等的宗旨。

Ⅱ．当前的南北战争

A．我们现在正处于南北战争之中。

B．战争检验由这个国家的创建者所表达的自由、平等宗旨能否天长地久地永存于世。

Ⅲ．我们今天在这里集会的目的

A．我们是在南北战争的战场上集会。

B．我们奉献这一战场的一部分作为墓地。

C．在这里战斗过的士兵，已经用他们勇敢的战斗，奉献给了这块土地。

Ⅳ．我们对未来的决心

A．我们必须献身于前人所未完成的工作。

B．我们必须献身于死者为之战斗过的事业。

C．我们必须保持国家孕育寓其中的自由观念。

当然，即使有最好的据以写作的提纲，我们也常常不能期望赶上林肯的不朽散文；但是，好的作品往往是经过很好地组织的，开始时拟一个提纲是很有用的方法。

程序设计象写作一样，必须很好地组织。在开始添加细节之前，最好有一个清楚而全面的计划。自顶向下设计的方法就象拟一个算法的提纲。第一步是制定一个一般计划，就象在提纲中写出主要论题。下一步，充实细节，就象在提纲中写出小标题，说明如何执行每个主要步骤。通过逐次地把这个计划弄精细，在研制过程的每一阶段增加进一步的细节，你就到达最后的目标：一个计算机能实行的由基本型指令表达的精确算法。

举例来说，假定要写一个算法（我们称它为算法X），来解代数课中的某个家庭作业问题。开始时，你只有算法应该做些什么的粗略想法，而很少想到具体指令。这时，你

● 美国林肯总统的一篇著名演说。可参见《英语学习》1981年第7～8期“葛底斯堡演说”浅析。——译者

把算法X想象为一个单个的实体（图1.1）。

当进一步思考这个问题时，你认识到解法实际上包括三部分，于是就对自己说："如果我能做第1部分，然后做第2部分和第3部分，则就会解决这个问题。"这就是**顶层设计**，如图1.2所示。

图 1.1

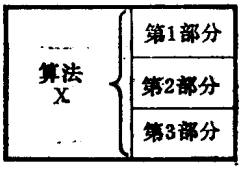


图 1.2

你可能还没有详细的算法，但是已经取得了一些进展；已经用三个较小的问题代替了一个大的问题。继续分析，也许发现第1部分能分成两个小问题——1A和1B，第2部分和第3部分也能类似地再分细。这就是第2层设计，如图1.3所示。如果算法复杂，可能还需要把这种分细的过程进行到另一个精细层次。最后，得到一个能够用计算机语言编写的详细计划。

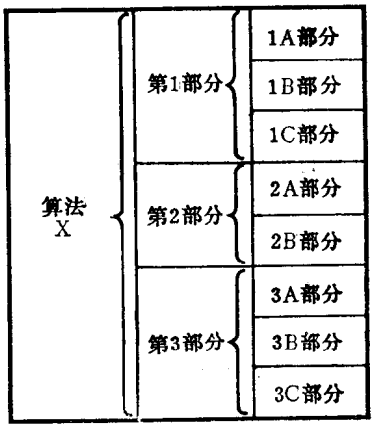


图 1.3

简而言之，自顶向下设计的基本原则是：

> 首先，集中精力于算法的总体设计。把它编写为一系列要执行的一般步骤。然后，用同样方法，对每一步骤充实其细节。

自顶向下设计中的一条重要原则是证实其**正确性**，它的意思是要对算法进行分析，使它确实正确。即使算法的第一草案是一个提纲，也必须是你试图要解的问题的有效解法。你应该这样问自己：如果能执行所提出的算法的每一步，真的就解决这个问题了吗？因此，自顶向下设计的第二条原则是：

> 在自顶向下设计的每一阶段，通过分析算法的正确性来证实算法。在确保整个计划稳妥以前，不要展开细节。

例1.1 求平均数。

假定给你一个任务，找五个数5.2、11.35、4.0、6.7和5.9的平均数。你将如何做？

如果使用计算器，也许你先写入第一个数，然后对它加上第二个数，再加第三个数，再加第四个数，最后加第五个数。一看有了五个数，就除以5，得到平均数。实际上，你所做的是执行求平均数的熟悉的算法的各步。

现在假定要你写一个一般算法，用来指示计算机找一组数的平均数，并印出结果。开始很简单，算法中有两个主要步骤。

求平均数的算法（叙述1）

Ⅰ．计算平均数。

Ⅱ．打印平均数。

这是顶层设计。

为了补充进一步的细节，你需要描述计算平均数的过程，就象用计算器做它那样。下面是修改过的叙述。

求平均数的算法（叙述2）

Ⅰ．计算平均数。

A．读入每一个数。当遍历这个数表时，计算这些数的和，并数一下它们一共多少个。

B．用步骤ⅠA中得出的和及计数，来计算平均数。

Ⅱ．打印平均数。

这是第二层设计。

为了检查你所做的，下一步应该证实算法。你应该问自己：是否确实能执行步骤ⅠA？是否确实能执行步骤ⅠB？这两步一起真是计算步骤Ⅱ所要求的平均数吗？理论上，你能使用这类推论，构造一个数学证明，证明算法是正确的。在很多实际场合，为了使自己相信没有错误，通过非正式的心算来证明就足够了。

在证实步骤ⅠB时，有一个小的“陷阱”要考虑。为了计算平均数，必须以计数除各数之和。如果计数是零会发生什么事？一个数除以零在数学上是无定义的，如果试图在计算机上这样做，就会出问题。如果是用计算器求平均数，就不会发生这个问题，因为人们知道不能除以零。然而，在为计算机写一个算法时，考虑象执行没有输入数据的算法这样一些例外情况是合理的。这种例外情况称为**边界条件**。稍作一点额外的努力，就能设计出对边界条件也能工作的算法。这样做将简化在计算机上检查算法的作业，当你开始运行自己的程序时，就会了解这一点。

现在，你能充实计算的细节而精化算法了。

求平均数的算法（叙述3）

Ⅰ．计算平均数。

A．计算并求各数的和。

1．令SUM和COUNT为0。

2．对要求平均数的每个数，做以下几点：

a．读入该数。

b．把它的值加到SUM。

c．COUNT加1。

B．计算平均数。

1．如果COUNT 不是0，则

a．令AVERAGE等于SUM除以COUNT。否则

b．令AVERAGE等于0。

Ⅰ．印出AVERAGE 的值。

这是一个完整的算法，它详细地告诉你如何精确地执行计算。如前所述，你应该证实这个算法，使自己相信，步骤ⅠA1和ⅠA2确实计了数并求了各数的和，步骤ⅠB确实计算了平均数（在COUNT是零的例外情况，也计算了平均数零）。

这个算法阐明几点共同的程序设计技术。词COUNT、SUM和AVERAGE是**变量**，它们表示计算中使用的一些数。与代数中一样，在程序设计中使用变量代表不知其值的数。

步骤ⅠA1是说SUM和COUNT 从等于零开始。这种做法称为**变量的初始化**——即，给它们一个初值。这有点象在开始新的计算之前，在计算器上按 CLEAR（清除）键。

步骤ⅠA2是一个**循环**的例子，是再三重复的过程。这个步骤中指令的每次执行称为循环的一次**迭代**（迭代的意思就是再做一次）。循环由一个**条件**控制。在这个例子中的条件是，是否还有任何数要被求平均。当条件假时，即所有数已读入以后，算法就出循环，进行步骤ⅠB。

变量COUNT 数读入值的个数。作这种用途的任何变量称为**计数器**。COUNT 加1的过程称为**计数器增值**。算法在循环的每次迭代中把计数器增值。

算法中的步骤ⅠB1是**条件指令**（常称为**if-then-else指令**），它根据条件“COUNT不等于0”成立与否（即真、假），或者执行步骤ⅠB1a，或者执行步骤ⅠB1b。

1.4 结构流程图

流程图是一种易读的算法图。把每一步写在框内，然后将这些框汇集在一起，表示要执行的指令的顺序。本书中的这类流程图称为**结构流程图**[1]。这些流程图便于使用第四章里解释的结构程序设计技术。

把简单的指令，象变量的初始化，做在一个长方形框中（图1.4）。

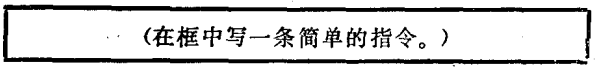


图 1.4

用包围循环中指令的L形框表示算法中的循环（图1.5）。外框中的条件就是控制循环的条件。这表示，当条件真时，要执行循环中的指令。可以在循环的开始〔图1.5(*a*)〕或循环的结束〔图1.5(*b*)〕检验该条件。例如，求平均数算法的步骤ⅠA2是一个循环，如图1.6所示。

用三角形表示条件指令（图1.7）。如果三角形中的条件真，算法就执行字“then”（则）底下的指令。如果条件假，算法就执行字“else”（否则）底下的指令。例如，求平均数的算法的步骤ⅠB1可以画表成如图1.8。

---

[1] 也称为纳萨-施内德曼（Nassi-Schneiderman）图，这种图是建立在I.纳萨（I.Nassi）和B.施内德曼（B.Schneiderman）创造的方法的基础上的〔ACM SIGPLAN Notices, Vol.8, No.8, August 1973〕。

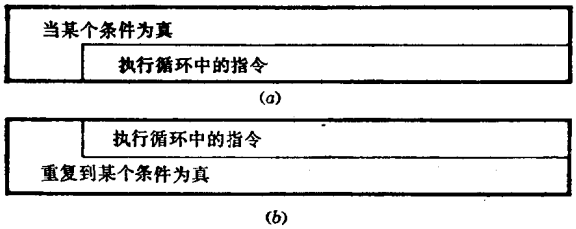


图1.5 表示循环的两种方式

（a）中条件是在循环的开始处检验；（b）中条件是在循环的结束处检验。

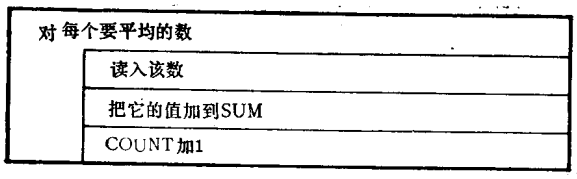


图 1.6

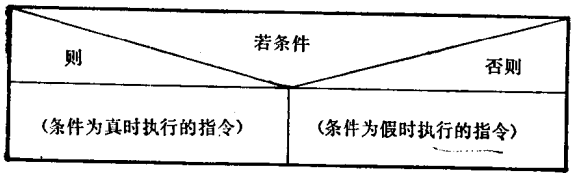


图 1.7

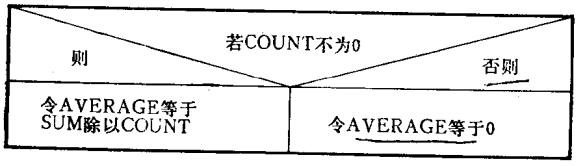


图 1.8

因此，你可以通过图 1.9 的结构流程图表示求平均数的整个算法。

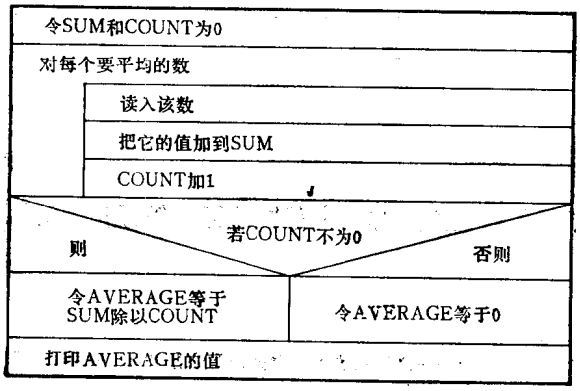


图 1.9

1.5 FORTRAN语言

在用计算机解题时，构造算法是第一步。你可以用如前所述的提纲形式● 写算法，或者用画结构流程图来表示算法。自顶向下设计的原则适用于随便哪一种情况。

下一步是编写程序。**程序**是用计算机语言写的算法。本书中，你将学习 FORTRAN 语言●。和平常的英语相比，FORTRAN 是一种简单的语言。FORTRAN 的词汇仅由几十个词组成（还要加上你自己建立的任何变量名字），如果知道句型的话，这些词也可以通过近十种句型得到。FORTRAN 中的句子称为**语句**。程序由一系列语句组成，每个语句写在单独的一行上。

虽然 FORTRAN 是简单的，但它是精确的。如果你写一句语法上不正确的英语句子，读者可能还会明白你的意思。在 FORTRAN 中就不是这样。每个语句有某种规定的形式。在 FORTRAN 中，在要求逗号的地方丢掉一个逗号，计算机就不认识这个语句，要不就产生误解。

因为FORTRAN 是精确的和无二义的，所以计算机可以读入你的程序，把它翻译为该机器指令系统中的命令，从而执行算法中的各步。事实上，名字 FORTRAN 就是 FORmula TRANslation（公式翻译）的缩写。我们现在称的“语句”，就是原来称的“公式”。因此，FORTRAN 是把程序中的语句翻译为机器可以执行的指令的语言。

以下各节介绍FORTRAN的基本概念和一些简单的 FORTRAN 语句，使你可以立即开始正确地运行程序。目前，我们将避开许多细节。就象通过听人们讲话而初学英语一样，你可以通过看一些完整的程序来学习有关FORTRAN 的一些概念。在以后的各章里，我们再涉及精确的规则。

1.6 计算机做什么

为了了解构成FORTRAN程序的指令，你需要有关计算机是什么和它做什么的基本知识。

图1.10是典型计算机系统的简化了的方块图。称它为系统，是因为它由几个部分组成。**中央处理机**是实在的计算机。这是执行程序指令的机器。**存储器**是存放指令和数据的地方。其它部分是用于输入输出的**外围设备**。

计算机虽然很快，实际上它是很简单的机器。它能执行的指令仅有下列四种类型。

1．存入存储器和从存储器取出

2．计算

3．输入和输出

4．程序控制

所有的程序，不管怎样复杂，都是由帮助计算机解释指令的说明语句，以及由这些类型的指令构成的。

● 提纲形式的另一个名字是伪码——之所以这样称呼是因为它象实际“代码”（即 FORTRAN 程序），但又不是一个完全的程序。

● 明确讲，你将学习 FORTRAN 77（见前言）。