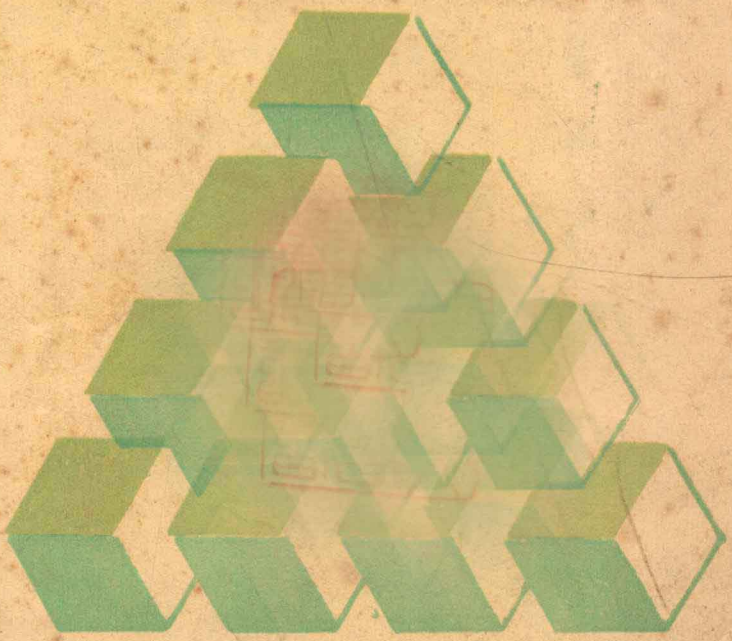
M. J. MERCHANT 著

FORTRAN 77
语言和风格



工程兵工程学院训练部译印

FORTRAN77语言和风格

FORTRAN77结构化入门

M.J.MERCHANT 著

苏挺英 王鸿德 张永康 译

钱士钧 徐永森 审校

本书是根据Michael J. Merchant著“FORTRAN77—Language And Style（A Structured Guide to Fortran77）”（Wadsworth Publishing Company, 1981）翻译的。第一、二、三、四章由苏挺英翻译，第五、六、七章由王鸿德翻译，第八、九、十章及附录由张永康翻译。全书由南京大学计算机科学系钱士钧、徐永森审校。

本书可供各类中、高等院校作为教材或参考书，也可供自学者学习之用。

前　言

本书编写时考虑了两个问题：讲授新的FORTRAN语言的最好方法是什么？讲授用FORTRAN语言进行结构程序设计的最好方法是什么？

为了解决第一个问题，在选择题目和确定叙述次序的时候，通常是依据FORTRAN语言1968标准。例如，字符数据被置于与数值数据相等的地位，并在书中较早地提出。增加的字符数据型（CHARACTER data type）不仅扩展了语言的功能，而且使之易于讲授，因为它给学生打开了一个应用方面的广阔天地，那里不需要高等数学就可以阐明程序设计方面的一些重要概念。增加的表制导输入和输出使学生不必去顾及格式（FORMAT）语句相当复杂的技术性细节。

在新标准出现之前，讲授用FORTRAN进行结构程序设计是困难的，因为缺乏必要的控制语句。增加了IF—THEN—ELSE 结构之后就有可能教会学生编写清淅的结构程序。

在整本书里，程序设计的语言与风格是溶合成一个整体的。各章中的文体风格部分给出了如何编写程序和利用FORTRAN语言的实践性指导。自顶向下设计及结构程序设计被不断地加以强调。

基于学习程序设计最好的方法是实践的坚定信念，第一章和第二章是使学生从一开始就能直接运行程序。在第一章里，学生将看到一个由自顶向下设计最终形成代码的完整程序。第二章则从细节上引导学生在计算机上运行程序。

作为用FORTRAN进行程序设计的介绍，整个课本可以在一学期内讲完。对于已经熟悉其他程序设计语言的学生可以略去第一和第二章，或很快地通过。在略去选读的章节之后，从第一章至第七章可以成为FORTRAN的一门小课或者作为数据处理入门课程的辅助材料。

虽然课本中的任何缺陷都是我个人的责任，我希望表达我对许多人衷心的谢意，他们阅读了原稿并提出了许多有价值的改进建议。（下面是致谢的人名）

目　录

第一章　计算机，算法，和程序设计

引　　言

在科学幻想小说、电影、和动画片中，计算机常常被描绘成如同有超等智慧的人一样。在这些幻想作品里，计算机用英语流利地交谈，解题时能独立地进行判断，从存储体内迅速检索出与任何问题有关的所有数据。这是一个动人的理想。而研究计算机的一个吸引人的前景是这个理想或许能在你的一生中得到实现。

但是，真正具有超等智慧的计算机乃是未来的事情。现代的计算机具有惊人的速度，它们还能按照非常复杂的指令系统动作。但它们仅仅是机器——从某些方面说，是相当简单的机器。一台计算机真正能做到的是执行经过程序员周密考虑并用程序设计语言（如FORTRAN）编写出来的简单指令。

从本书，你将学习两件事：一是如何使用计算机解题——就是如何编制指令以完成你的作业。特别是，你将学会设计和编写算法，它是一种计算机能够执行的过程。

在这一章里，我们将说明什么是算法，如何用一种称之为框图的图形来描述算法，并且如何运用自顶向下设计的方法去建立一个算法。

当解题的算法设计好之后，你需要用一种计算机能够理解的语言来表达你的算法。计算机程序是用程序设计语言（如FORTRAN）写成的算法。因此，你将学会第二件事——如何编写一个FORTRAN程序，这与第一件事同等重要。

在第一章里，将给出有关FORTRAN语言的介绍。在第二章里，将可以看到如何在计算机上运行FORTRAN程序。

算　　法

人们相互给出的指令往往是不精确的。一条简单的指令，例如“到商店去买块面包”，执行时需要作出成百个判定：你是从前门还是后门走出去？向左拐还是向右拐？商店在哪里？面包放在哪里？你要的面包是全麦的、酸面的还是黑麦面的？人是有智慧和具有推论能力的，能领会笼统的、含糊的指令的真正含意。但是，计算机是没有意识的。当你编写一个要计算机执行的过程时，每条指令都必须是明确的。

假设你要用计算器解一道数学题。但你没有计算器，你的朋友有，因此你打电话请求她的帮助。她不在家，她的弟弟，虽然只懂得简单的数学，答应如果你能准确地告诉他如何做法，他愿意使用计算器。现在你必须应用指令去详细说明如何解题，这些指令应是如此的明确和不含糊，使它们不致于被曲解。你必须说：“送入56.2，按一下加法键，送入475.3，按一下加法键，送入11.63，按一下等号键，将上边发光的数字告诉我。”这是一段用日常

语言表达的算法。

算法是步进式的解题过程。一个正确的算法应该满足三个条件：

1、算法的每一步应该是一条能被执行的指令。

2、步与步之间的顺序应该是明确的。

3、算法最后必须终止。

算法并不一定都是为计算机编写的。例如你可以用纸和铅笔来执行指令。烹调书上给出了焙制巧克力小甜饼的“算法”。对于一名厨师来说，指令“焙烧到好”的含意是明确的，所以它满足算法的第一个条件。

假如在告诉计算机“解下题……”之后，机器便能执行的话，在使用上将是十分方便的。然而，一台计算机的指令系统仅包含了一百条左右基本命令以实现电子化。这些命令与用按键的方式发给计算器的命令相似。例如，你可以告诉计算机进行两数相加。算法正确性的第一个条件含意是：在给计算机编写算法的时候，每一步的内容必须是计算机在执行这些基本指令之后所能实现的。

在执行算法中的指令时，机器每次执行一条指令。正确算法的第二个条件含意是：算法必须明确规定指令执行的顺序。

第三个条件的含意则是：算法不能是无休止的。下列过程不成其为算法。

计数过程

第1步　令N等于零。

第2步　将1加到N上去。

第3步　转向第2步。

如果让计算机执行这个过程，从理论上说，机器的运转是永不休止的。下列过程则可以成其为算法，因为它最后是终止的。

计数（到一百万）的算法

第1步　令N等于零。

第2步　将1加到N上去。

第3步　如果N小于1,000,000，则转向第2步；否则停止。

自顶向下设计

在执行简单问题的算法时，你只要经过短时间的思考就可以得出答案。然而，实际上在使用计算机时很少是这样简单的。专业程序员所编写的程序常常包含了成千条的指令。设计这样的程序与设计一个具有成千个部件的机器一样复杂。为了使它正常工作，所有的部件都必须恰当地组合在一个有机体内。

自顶向下设计是设计算法的一种方法。它可以用于组织你的工作和算法。

从许多方面说，设计算法和撰写文章相似。在写文章的时候，你是从所要表达的中心思想开始的。然后接着组织你的思路，并且用辞句向读者叙述你的思想。设计算法时，也是从你对完成任务的总的设想开始的。你必须组织思路，并且给计算机选择正确的指令顺序，使

它能执行预定的操作。但是和写作一样，在程序设计中，难以决定的常常是从哪里着手。

拟定提纲是着手开始撰写文章的好方法。首先，列出全文的主要纲目，例如：

Geffysburg　演讲

Ⅰ、国家的概念

Ⅱ、现时的内战

Ⅲ、此时此地我们的目标

Ⅳ、我们对未来的决策

这个朴实的提纲提供了一篇文章的框架，所有的段落和句子将填入这个框架。整体结构确定之后，余下的是精心地使纲目更详细。可以在纲目下增加细目，例如：

Geffysburg　演讲

Ⅰ　国家的概念

A、国家在八十七年前建立

B、它是自由的体现

C、它维护所有男人（人？）绝对平等的主张

Ⅱ　现时的内战

A、我们现在进行着内战

B、战争长期考验着这个国家建立者的能力

Ⅲ　此时此地我们的目标

A、我们在内战的战场上聚会

B、我们将把战场变成坟地。

C、在这里作战的士兵以他们的勇敢战斗献身于这块土地

Ⅳ　我们对未来的决策

A、我们将献身于我们面前尚未完成的工作

B、我们应该忠实于死难者为之战斗的主义

C、我们必须坚持我们国家所维护的理想——自由

当然，即使是从最好的提纲着手工作，任何人不可能指望经常模仿林肯的不朽论文；但是好的作品总是有好的结构，而拟定提纲是着手写作的很有用的方法。

与写作一样，在程序设计中，好的结构是最重要的。在开始充实细节之前，最好有一个清晰的整体计划。自顶向下设计的方法就像在拟定算法的提纲。第一步是简洁地叙述一个全面的计划——就像编写提纲中的纲目一样。其次，你填入细节，就像在提纲中写入小标题一样，用于指明如何执行每个主要步骤。在对每个进展阶段填入细节使之相继完善之后，你到达了最后的目的：一个能够用基本形式的指令表示，且能被计算机执行的正确算法。

假如你要编写一个求解某些代数课程家庭作业题的算法（称之为算法X）。当你开始着手的时候，你有一个有关于算法的粗略想法。但是对于具体的指令还想得很不具体。据此，你将算法X设想为一个单一的方框（图1.1）。

当你进一步思考问题，你认识到解题时存在三个部分。你自言自语：“如果我完成第一

图 1.1

当你开始着手编写算法的时候，将它设想成一个单一的方框

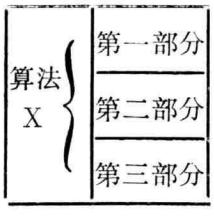


图 1．2

在自顶向下设计中，你首先确定算法的总体结构

部分，接着完成第二和第三部分，就可以解这个题。”这是你的顶层设计（示于图1.2上）。

你还没有得出详细的算法，但是你已经有了些进展；你用三个较小的题目去代替那个大题。继续你的分析，你可以发现，第一部分可以分解成两个分题——1A部分和1B部分——第二部分和第三部分都可以同样地分解。这是你的第二层设计（示于图1.3中）。如果算法比较复杂，就需要继续这个过程达到更为精细的层次。最后，你得到一个可以用计算机语言编写出来的详细计划。

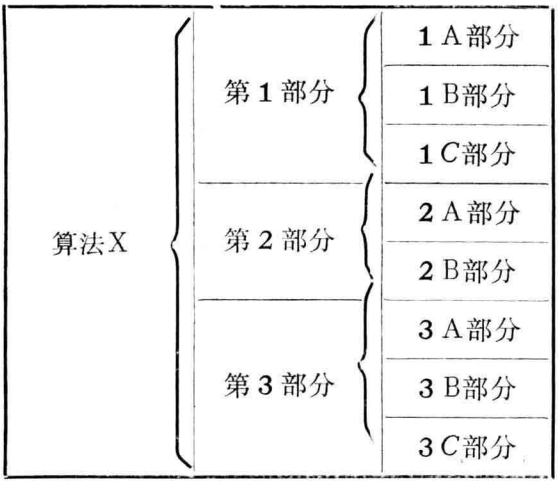


图 1．3

填入细节，使自顶向下设计精细化

总之，自顶向下设计的主要原则是：

> 首先将精力集中在算法的总体设计上，把它编写成一系列有待执行的步骤。然后，用同样的方法，给每一步填入细节。

在自顶向下设计中，确认是一条重要的原则，它意味着算法要经过分析以确认其正确性。即使算法的初次草稿还只是一个提纲，它必须是求解问题的有效解。你应该回答自己的是：如果执行了提出的算法的每个步骤，所得到的是否该问题的真实解。因此，自顶向下设计的第二条原则是：

> 在自顶向下设计过程中的每个阶段，要对算法进行分析，以确认其正确性。只有在你肯定总体计划是正确时才展开其细节部分。

例1.1 求平均值

设要计算数5.2，11.35，4.0，6.7和5.9的平均值，你将如何做法？使用计算器时，你可能先送入第一个数，然后加上第二个数，在其和数上加上第三个数，再加上第四个数，接着加上第五个数。当心中记下已有了五个数时，用5去除这个和数并算得其平均值。这里你所要进行的工作是用一个求平均值的算法，将这个步骤表达出来。

现在，如果你要编写一个能够用于指挥计算机求解一组数的平均值，并打印出结果的通用算法。开始很简单，算法中有两个主要步骤。

求平均值的算法（方案1）

Ⅰ、计算平均值。

Ⅱ、打印平均值。

这是顶层设计。

为了充实细部，你需要按照使用计算器的方法将过程描述出来。这是一个修正方案。

求平均值的算法（方案2）

Ⅰ、计算平均值

A、读入每个数，在送入数据表的过程中，计算这些数的和，并计数。

B、利用第ⅠA步中的和数与个数，计算平均值。

Ⅱ、打印平均值。

这是你的第二层设计。

接着，要确认这个算法以检验你的工作。你要回答自己：第ⅠA步是否的确可能执行？第ⅠB步是否的确可能执行？这两步相结合是否的确可能算出第Ⅱ步所需要的平均值？从理论上说，你可以采用这种方式的推理去建立算法正确性的数学证明。就实用目的说，对于大多数的情况，进行这样非形式的推理，已足以说明你没有犯什么错误。

在论证第ⅠB步时，有一个小“把柄”值得研究。在计算平均值时，需要将和数除以个数，如果个数为零时，将会出现什么结果呢？任一个数被零除在数学上是不确定的。如果你试图在计算机上进行这个计算，就会出现问题。如果你用计算器进行平均值的计算，问题就

不会产生。因为一个有一般常识的人是不会用零去除的。然而，当给计算机编写算法的时候，考虑一些异常情况是明智的，就像没有输入数据而要执行算法一样。这样的异常情况称之为边界条件。稍加补充，你可以设计一个同时考虑到边界条件的算法。当你开始运行你的程序时，你将会认识到，这个措施可以简化在计算机上验证算法的工作。

现在，你可以填入计算的细节，使算法精细化。

求平均值的算法（方案3）

Ⅰ、计算平均值

A、计数并求其和

1、令SUM与COUNT为零。

2、对于各数，执行下列操作

a、读入该数。

b、将其值加入SUM。

c、将1加入COUNT。

B、计算平均值

1、若COUNT不为零，则

a、令AVERAGE等于SUM除以COUNT。

否则

b、令AVERAGE等于零。

Ⅱ、打印AVERAGE的值。

这是一个完整的算法，它给出了足够的细节。严谨地告诉你如何去进行计算。如前所述，你要确认这个算法，说明第ⅠA1步和第ⅠA2步的确计了数并求得这些数的和。第ⅠB步的确计算了平均值（对于COUNT等于零的异常情况，它算得平均值为零）。

这个算法表明了一些程序设计的一般方法COUNT，SUM和AVERAGE等字称为变量，它们是计算中用到的数。与代数学里一样，在程序设计中你可用变量去替代其具体值尚属未知的数。

ⅠA1步提出算法从“令SUM和COUNT为零”开始，这项工作称之为变量的预置——即赋给它们以初值。这如同在开始新的计算之前，在计算器上按一下CLEAR键一样。

第ⅠA2步是循环的例子。它是一个多次反复的过程。在这一步中的指令组每执行一次称为循环的一次重复（重复意味着再执行）。循环由条件来控制。在这里，条件为“是否还有参与计算平均值的数。”当所有的数都输入之后即条件是“假”时，算法就从循环转出或退出，并进到第ⅠB 步。

变量COUNT用于计算读入数的个数；任何作此用途的变量均称之为计数器。将1加到COUNT上去的过程称之为计数器的增值过程，这个算法使计数器在循环的每次重复时加1。

算法中的第ⅠB1步称之为条件指令（通常称之为if－then－else指令）。它提出根据条件“COUNT不等于零”来决定执行第ⅠB1a步或第ⅠB1b步。

结 构 框 图

框图是算法的便于阅读的图表。你可以将每个步骤填入一个方框中，并将这些方框组合起来以表示指令执行的次序。本书所用的框图称之为结构框图。这些框图能使将在第四章里介绍的结构程序设计方法简易化。

你可以将一条简单的指令，例如预置一个变量，用一个方框来图示（见图1.4）。

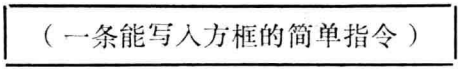
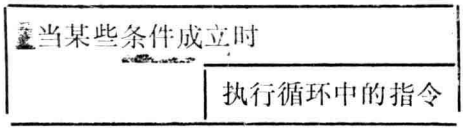
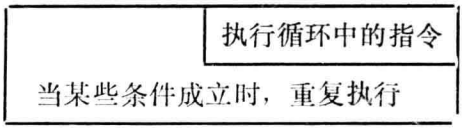


图 1.4

为了表明算法中的循环，可以用一个L形的方框将循环中的指令围起来（见图1.5）。方框外所标明的条件乃是循环的控制条件。它指明当条件成立时，循环中的指令应予执行。你可以在循环开始时（见图1.5(a)）或者结束时（见图1.5(b)）检验这个条件。例如，在计算平均值的计算中第ⅠA2步为一循环，可以用图1.6来表示。



（a）



（b）

图 1.5表示循环的两种方法

（a）中，在循环开始时检验条件；而（b）中，在结束时检验条件。

为了表明条件指令，可以采用一个三角形（见图1.7）。若三角形内的条件成立，算法执行“then”字下的指令。如果条件不成立，算法执行“else”字下的指令。例如，在求平均值的算法中ⅠB1步可以用图1.8来表示。

样这，你可以用图1.9上的结构框图来表示求平均值的完整算法。

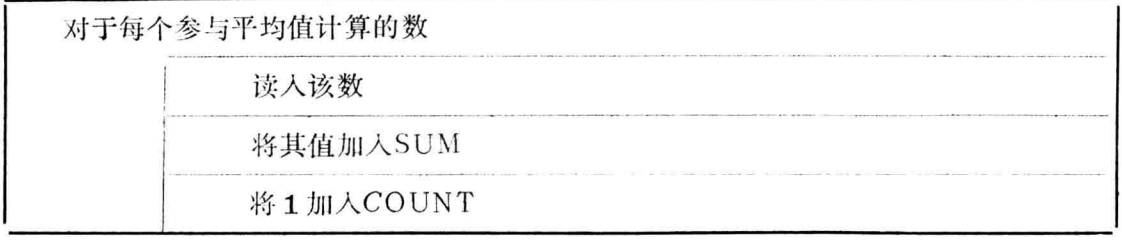


图 1.6

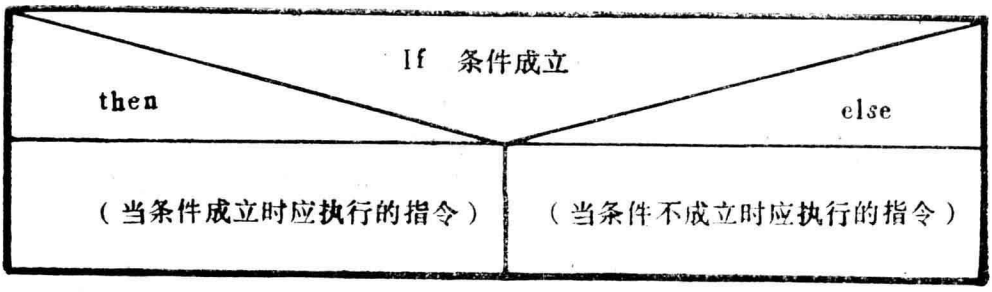


图 1.7

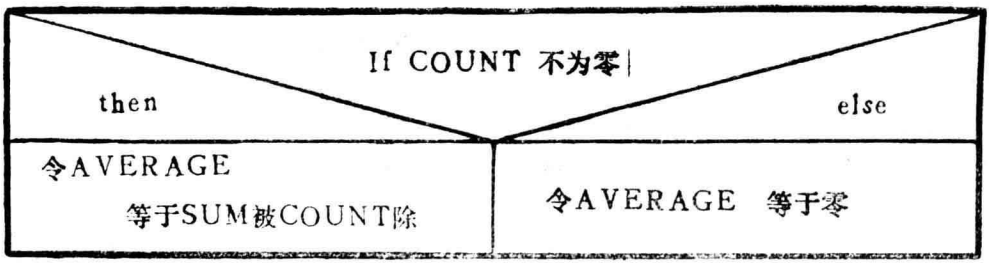


图 1.8

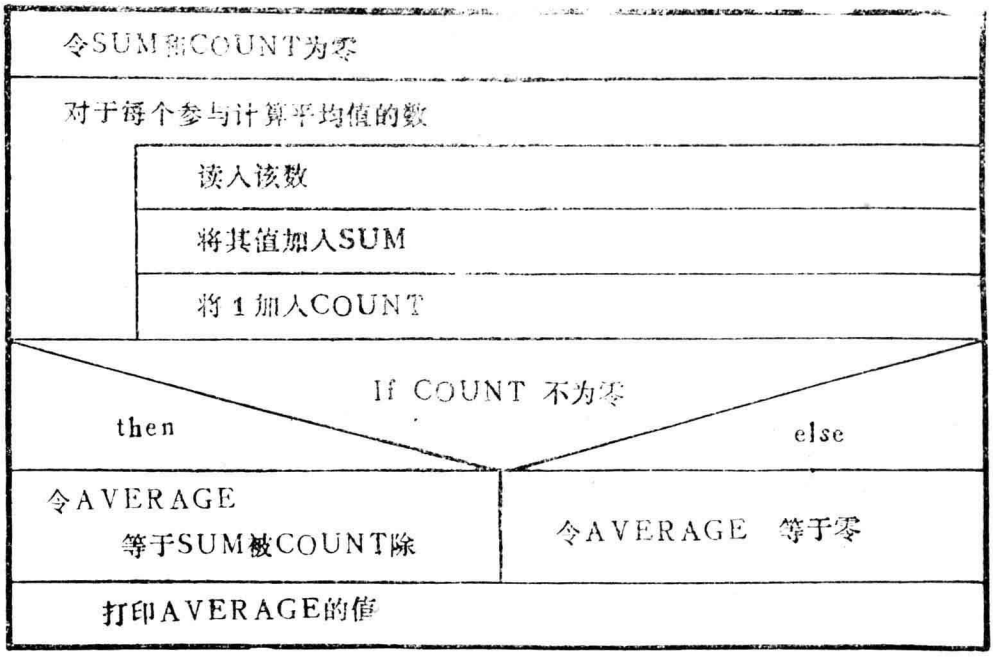


图 1.9