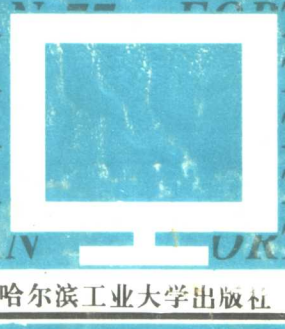
非计算机专业系列教材

FORTRAN 77
结构化程序设计

主编　蒋宗礼

主审　王义和



哈尔滨工业大学出版社

非计算机专业系列教材

FORTRAN 77结构化程序设计

主　编　蒋宗礼
副主编　王宇颖　潘侠　褚滨生
主　审　王义和

哈尔滨工业大学出版社

(黑)新登字第4号

内容提要

本书是根据计算机等级考试大纲编写的高等院校教材，主要内容包括：FORTRAN 77结构化程序设计的基本知识、各种数据类型、语句的语法、语义以及程序构成。在内容组织上，不仅讲授FORTRAN 77，更注重讲授程序设计的方法，使学生在实践中能够尽快编制出结构合理的程序。

本书为高等院校非计算机专业本、专科学生教材及非计算机专业等级考试学习和培训教材，也是一般工程技术人员实用的参考书。

非计算机专业系列教材

FORTRAN 77结构化程序设计

FORTRAN 77 JIEGOUHUA CHENGXU SHEJI

主　编　蒋宗礼

主　审　王义和

*

哈尔滨工业大学出版社出版

新华书店首都发行所发行

哈尔滨工业大学印刷厂印刷

*

开本787×1092　1/16　印张19.25　字数465千字

1994年12月第1版　1994年12月第1次印刷

印数　1—10 000

ISBN　7-5603-1070-2/TP·72　定价15.80元

前　言

FORTRAN 是一种高级程序设计语言，是用计算机完成数值计算的主要语言。自 1956 年正式使用以来，随着计算机技术的迅速发展，已先后推出一些不同版本。由于对结构化程序设计的良好支持及其它特点，FORTRAN 77 早已成为最流行的版本。

本书集作者多年之教学经验，参考了大量有关教材，并根据初学者的特点和国家教委考试中心制定的等级考试大纲的要求，依据由浅入深的原则，对内容进行了认真组织。不仅讲授 FORTRAN 77，而且还从算法入手，致力于程序设计的讲授。在讲述语言语法规则的同时，适当地介绍了导致这些规则的原因，使读者通过对系统的了解能更好地掌握语言和程序设计方法，提高编程水平。我们将程序设计能力看作是一种修养，力求将相关的问题融于全书，尤其是对易出错的问题，以不同的形式进行了反复的讲述。

由于高级语言程序设计是一门实践性很强的课程，所以，书中除有适量的精选例题外，还有大量的习题。希望读者在使用高级程序设计语言编写组织程序、解决实际问题方面尽可能多地得到训练。有条件的读者，可以先设计好算法，然后直接在机器上完成编程和调试。

全书共分十章，按 48 学时讲授。本书由哈尔滨工业大学计算机科学与工程系蒋宗礼副教授主编，王字颖、潘侠、褚滨生副教授任副主编，王义和教授主审。其中第一、二章由蒋宗礼编写，第三、四章由王字颖编写，第五、六、七章由潘侠编写，第八、九、十章由褚滨生编写。

由于我们水平有限，书中难免存在疏漏，恳请广大读者提出宝贵意见。

作　者

1994 年 12 月

目　录

第一章　计算简介

自 1946 年 12 月 15 日由以 Von Neumann 为首的一批学者在美国宾夕法尼亚大学莫尔分校研制出世界上第一台电子计算机“ENIAC”(Electronic Numerical Integrator and Computer)以来，在不到 50 年的时间内，电子计算机从第一代的电子管、第二代的晶体管、第三代的集成电路，已经发展到第四代的大规模集成电路电子计算机，而且系统的性能价格比有了极大的提高。目前，新一代的计算机的研制正在大力推进。

计算机的应用范围已从单纯的科学计算发展到生产与生活的各个方面，许多重要的地方已离不开电子计算机。我们相信，随着人类对计算机及其能力的不断深入的研究和了解，计算机的应用将会更为广泛。本书的主要目的就是引导读者学习使用计算机进行问题求解。

目前的计算机大部分仍需要用户告诉它如何进行问题求解，就是说，用户需要将解题的方法和步骤比较详细地以规定的形式和途径告诉计算机，在这个前提下，计算机才能帮助人们进行问题求解。所以，我们希望读者去努力学会根据计算机的特点，针对不同类型的问题，设计出(甚至只是较好地描述出)有效的解题方法和步骤。至于计算机语言，则是一种描述工具，而且计算机的高级语言一般都比较简单，易于掌握。这有点象我们撰写科技论文，要写的内容(新成果)是主要的，而用语言将它描述出来则相对处于次要地位。

解题的方法和步骤的精确描述叫算法。

有了正确的算法，就可以根据具体的计算机系统所“懂得”的较有效的一种语言来编制程序。所以，面对一个待解决的问题，第一步是确定解题的方法和步骤，设计出合适的算法，第二步才是用选定的语言编写程序。这一点，请读者从初学时就遵照去做。

本章除简单介绍计算机的软、硬件系统和简单的程序示例外，重点在于介绍算法的基本设计方法和描述方法。所以，如果读者是程序设计的初学者，请给予足够的重视；如果读者在程序设计方面已有良好的修养，可以跳过此章。

1-1　计算机系统

计算机系统由计算机硬件系统和计算机软件系统组成。

一、计算机硬件系统

一般来说，计算机都是指自动的、能够存储程序的计算机。计算机硬件系统指包括组成计算机系统的所有硬设备，也就是未配置计算机软件的计算机系统，我们称之为裸机。它能够以某种方式接受信息，并按照程序对信息进行处理，然后提供处理的结果。因此，它通常由输入设备、输出设备、存储器、中央处理机(CPU)组成。典型结构如图 1-1 所示。

存储器用来存储程序和数据，由许多存储单元组成。这里所说的存储器是指内存储器(简称内存)，一般的计算机系统还有比内存的存储容量(即存储单元的个数)大许多倍的外部存储器(简称外存)，如硬磁盘、软磁盘、磁带等。

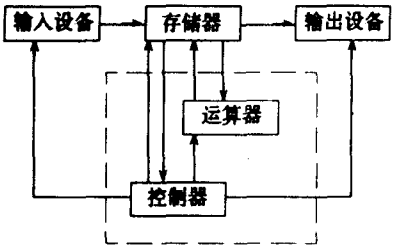


图 1-1

中央处理机(CPU)由运算器(ALU)和控制器(CU)组成。运算器的功能是实现系统中的算术和逻辑运算，控制器则是用来分析指令，并向各个部分发出具体的操作命令。

输入设备与输出设备是针对内存而言的。将外部信息送入到内存中的设备叫输入设备，如终端的键盘、卡片输入机、磁带读入机。把内存中的信息送到外部的设备叫输出设备，如终端的显示器、打印机等。输入设备和输出设备简称为 I/O 设备(Input/Output)。

二、计算机软件系统

计算机的硬件系统只能识别由 0、1 串表示的指令。对用户来说，用 0、1 串来表示问题求解的方法和步骤是非常困难的，对一般普通用户来说甚至是不可能的。要想充分发挥计算机硬件系统的作用，必须给它配备良好的软件系统。

所谓软件(Software)是相对硬件而言的，它包括计算机运行所需要的各种程序和有关资料。如汇编程序系统、编译系统、操作系统、数据库管理系统、诊断程序、应用软件包等。这些软件可以扩大计算机的功能和效率。一般地，人们将软件分成两大类：系统软件和应用软件。例如，操作系统就是一种系统软件，它实现有效地管理计算机系统中的软件和硬件资源，为用户提供方便的使用手段，并使系统可以高效、协调地工作。再如编译系统，它是在操作系统的支持下运行的，用于将高级程序设计语言编写的程序翻译成面向计算机的等价的代码。例如，FORTRAN 77 就是一种面向用户的适应于科学计算的计算机高级程序设计语言，我们用这种语言编写一个科学计算程序要比用面向计算机的 0、1 串组成程序容易得多，但计算机是不“懂”FORTRAN 77 语言的，这种程序必须通过 FORTRAN 77 的编译系统转换后，才能在计算机上执行。一般地，在选用计算机高级程序设计语言来编写程序进行问题求解时，要经过如下过程：

(1) 弄清待求解的问题；

(2) 设计适当的算法；

(3) 用一种适当的高级程序设计语言编制程序；

(4) 将此程序输入到计算机系统中；

(5) 用系统提供的该语言的编译系统将该程序翻译成等价的可运行目标程序；

(6) 命令计算机系统执行这个目标程序，取得问题求解的结果。

三、程序设计语言

语言是人们用于交流信息的工具。通常，中国人用汉语，因为大家都“懂得”这些汉语句子所表达的意思；美国人用英语，法国人用法语，其道理都一样。计算机是人们用来进行“计算”的工具，人们要让计算机帮助他完成问题求解，就需要用人和计算机都能理解的语言，把

问题求解的方法和步骤以及问题本身告诉计算机。这样，计算机才能按照人的指令去工作。计算机语言可以分成三个级别：机器语言、汇编程序设计语言（简称汇编语言）、高级程序设计语言（简称高级语言）。

1. 机器语言

每台计算机硬件系统都有自己的语言，我们称之为机器语言。所谓机器语言，是指计算机直接使用的程序代码（称为指令代码）。这种代码无需进行翻译，可直接被机器执行。机器语言程序中对存储单元的访问使用绝对地址和相对地址。

不同的计算机一般都具有不同的机器语言，它们之间的差别甚至是很大的。而且，每台计算机的硬件系统也只"懂得"它自己的语言。机器语言均是用 0、1 代码组成的。例如，在 Z8000 机中，计算 7＋5 的程序可写成

单元地址	机器指令
1000000000000000	1011110100000111
1000000000000010	1011110100010101
1000000000000100	1000000000000001

其中第一条指令的意思是将整数 7 送入寄存器 R0；第二条指令的意思是将整数 5 送入寄存器 R1；第三条指令的意思是将 R0 的内容加上 R1 的内容，结果送到 R1 中。而在 DJS-130 中，7＋5 的计算程序为

单元地址	机器指令
0000000000110000	0000000000000111
0000000000110001	0000000000000101
0000000000110010	0010100000110000
0000000000110011	0011000000110001
0000000000110100	1011011000000000

上述第一个单元中存放的是整数 7，第二个单元中存放的是整数 5，第三个单元中存放的指令是将第一个单元中存放的内容取到累加器 AC_1 中，第四个单元中存放的指令是将第二个单元中存放的内容取到累加器 AC_2 中，最后一个单元中存放的指令是将 AC_1 中的内容加上 AC_2 中的内容，结果存放在 AC_2 中。

由这个例子我们就可以看到这两个机器语言的差别。这两种机器的差别在这一简单计算程序中也有反映。如 Z8000 使用寄存器参加运算，而 DJS-130 是使用累加器进行运算的，前者以八个二进制位为一个编址单元，后者则是以十六个二进制位为一个编址单元。可见用机器语言编程序的困难程度。我们不仅要用这些既难记又难懂从而又非常容易出错的代码去编程，而且在编程中还要对机器内部的结构有非常清楚的了解。在这种情况下，我们能有多大的精力放在问题求解方法的研究上，一天又能编写出多少条指令组成的程序呢？为了解决其中的一些问题，人们创造了汇编语言。

2. 汇编语言

在 50 年代初，Wilkes 等人引进了子程序、伪指令、操作码、地址码、十进制数表示、相对地址功能以及十一二转换的概念，使用了助记符，在 IBM 650 上构造出第一个汇编程序，使得人们可以用汇编语言来编写程序，极大地提高了工作效率，大大地降低了程序的出错率。

汇编语言仍是一种面向机器的程序设计语言，通常是为特定的机器专门设计的，与机器

语言非常接近。实际上，它是引人一些助记符来代替二进制代码，甚至用助记符来代表一组指令。用这种语言编写的程序需经过汇编程序翻译成机器语言程序。一般来说，汇编语言指令与机器语言指令是一一对应的，当助记符代表的是一组指令时，也可以方便地用这一组指令替换之，所以汇编语言的翻译程序—汇编程序的构造是比较容易的。如果汇编语言中允许用一个助记符代表一组指令，那么这种汇编语言叫做宏汇编语言。

由于汇编语言中引人了良好的助记符，所以用起来比机器语言方便了许多。如 7+5 的 Z8000 汇编语言程序可写成

指令	意义
LDK R0,#7	把常数 7 取人寄存器 R0
LDK R1,#5	把常数 5 取人寄存器 R1
ADD R0,R1	寄存器 R0 的内容加上寄存器 R1 的内容，并将结果送到 R1 中。

使用 DJS-130 的汇编语言则可写成

标号	操作指令	意义
	LDA 1,A	取标号 A 所指单元的内容(7)送到累加器 AC_1
	LDA 2,B	取标号 B 所指单元的内容(5)送到累加器 AC_2
	ADD 1,2	累加器 AC_1 的内容与 AC_2 的内容相加，结果送 AC_2
A:	7	常数 7
B:	5	常数 5

显而易见，有了汇编语言，人们便从 0、1 代码中解脱出来了。现在可以详细地研究让机器如何去干好一件事，而翻译工作留给了机器。尽管使用汇编语言是一大进步，但是它仍要求用户对所用的计算机的逻辑结构及其性能有较深人的了解。由于所编写的符号指令仍是与机器指令大多都一一对应，这使得编起程序来仍是一件不容易的事。为了进一步减轻人们的编程负担，使得编程的人无需去了解计算机的逻辑结构，而只关心如何去解一个题，人们又创造了高级语言。

3. 高级语言

第二代计算机问世以后，计算机的速度和处理能力不断得到提高，计算机的高速度与人工编程的低效率之间的矛盾越来越突出。1956 年前后，在美国出现了 FORTRAN(Formula Translation)高级程序设计语言，这是世界上最早出现的高级语言，是程序设计语言发展中的又一个重要的里程碑。它同时吸收了机器语言和数学语言的特点，编出的程序具有很强的直观性。例如，前面讲的 7+5 的计算，就可以按习惯写成 7+5；要计算式子

$$\sin x+\sin y$$

的值，则可写成

$$SIN(X)+SIN(Y)$$

FORTRAN 语言作为一种适应于进行科学计算的语言的成功实现，证明了高级语言的

可行性与生命力。它引进了许多新的概念，如：变量、表达式、数组、语句、输入输出格式等，使用户在编程时不再依赖于机器的结构，也正是由于这个原因，使得用户编写出来的FORTRAN语言程序可以在不同的机器上得到执行。

事实上，我们可以把人与计算机看成交往的双方，语言是这一交往的中间站，在计算机刚出现时，这个中间站紧靠计算机而远离人。到了汇编语言时代，这个中间站向人这边移动了一些，而到了高级语言时代，这个中间站就更靠近人了。可见，计算机语言的发展是从面向计算机逐渐地变得面向问题、面向用户，让计算机本身做越来越多的工作，而不断地减轻人的负担。

FORTRAN语言的极大成功，大大地激发了人们设计高级语言的积极性。这些年来，人们已根据不同的应用要求，设计出了上千种计算机语言。当然其中大多数是专用语言，通用语言较少，如：FORTRAN、PASCAL、C、ALGOL、COBOL、BASIC、LISP、ADA、PL/1等。

四、FORTRAN语言的发展

FORTRAN是Formula Translation的缩写，意思是"公式翻译"。它主要是为进行科学和工程中的计算而设计的，所以，它具有很强的数值计算功能。从1956年开始正式使用到现在，一直在数值计算领域中占据主要工具的地位。

自从FORTRAN语言问世以来，根据实际应用的需要和计算机科学技术的不断发展，先后推出了一些不同的版本。最流行的有：FORTRAN Ⅱ、FORTRAN Ⅳ、FORTRAN 77。其中FORTRAN Ⅱ的第一个编译程序是1955年在IBM 7040计算机上实现的。在随后的几年中，FORTRAN又相继出现在其它计算机上。1962年初，FORTRAN Ⅳ的编译程序被构造出来。FORTRAN Ⅳ不仅扩大了FORTRAN Ⅱ的功能，而且对FORTRAN Ⅱ还做了一些改动，这使得原来用FORTRAN Ⅱ写的程序不经修改难以在FORTRAN Ⅳ的编译程序的支持下运行。这种不兼容性引起了人们的重视。1966年3月，美国国家标准化协会(ANSI，即American National Standard Institute)公布了两个标准文本：

(1) 美国标准基本FORTRAN(ANSI，X3.10－1966)，它相当于FORTRAN Ⅱ；

(2) 美国标准FORTRAN(ANSI，X3.9－1966)，它相当于FORTRAN Ⅳ。

由于FORTRAN语言在国际上得到了广泛的使用，1972年7月，国际标准化组织(International Standard Organization，简称ISO)发表了ISO FORTRAN标准。即《程序设计语言FORTRAN ISO 1539－1972》，ISO的FORTRAN分为三级：

(1) 完全FORTRAN，相当于FORTRAN Ⅳ；

(2) 中间FORTRAN，介于FORTRAN Ⅱ与FORTRAN Ⅳ之间；

(3) 基本FORTRAN，相当于FORTRAN Ⅱ。

有人称FORTRAN Ⅳ为FORTRAN 66，它流行了十几年(甚至有人至今还在用)，几乎统治了整个数值计算领域，许多的应用程序和程序库都是用FORTRAN语言编写的。但是，自从结构化程序设计被提出来以后，人们越来越感到FORTRAN Ⅳ的不足。美国标准化协会通过对ANSI FORTRAN(X3.9－1966)的修改，在1978年重新发表了FORTRAN文本，为了与FORTRAN 66相区别，定名为FORTRAN 77，即FORTRAN(X3.9－1978)，同时宣布撤消FORTRAN(X3.10－1966)。FORTRAN 77与FORTRAN 66基本上兼容，这主要是考虑到不使目前大量流行的FORTRAN 66程序作废。1980年，FORTRAN 77被接受为国

际标准，即《程序设计语言 FORTRAN ISO 1539－1980)。该标准分为全集和子集，从而使FORTRAN 语言的发展进入了一个新时期。

随着大规模并行处理技术的发展，在 80 年代后期，人们又致力于并行 FORTRAN 语言的研究，目前已提出一些版本，如：FORTRAN 90、FORTRAN D 等。可以预料，FORTRAN 语言必将在数值计算领域中继续发挥其无法取代的作用。

1-2 算法及其表示

计算机科学是研究信息表示和信息处理的科学。当我们要用计算机进行问题求解时，需要用适当的方式把该问题表示出来，并研究如何通过对问题的这种表示的处理来求得问题的解。这里所说的适当的方式就是适当的数据结构。一般来说，选择问题的合适的数据结构表示与设计具体的处理方法和步骤是互相影响的。对科学计算而言，在已建立了问题的数学模型的前提下，数据结构基本上是比较固定的，所以，主要的任务是根据给定的数学模型(包括选定的数据结构)，考虑充分利用计算机系统提供的方便手段，去设计算法。

一、算法的概念

所谓算法，就是对解题方法的精确描述。即对解决问题而采用的方法和步骤的描述。应该指出，我们这里讲的算法，并不仅仅指加、减、乘、除……等等算术运算所涉及的计算方法。

例 1-1 描述一次课的内容的学习过程。

一般地，大家都会采用如下步骤：

1. 预习；
2. 听课；
3. 复习。

对这三步，每一步都有不同的几件事要做，因此，我们可以给出它的更详细的描述，这个过程叫作求精。

1. 预习：
   1.1 阅读教材中关于本次课内容的部分；
   1.2 思考有关问题；
   1.3 记下需要重点注意的问题。
2. 听课：
   2.1 走进上课教室；
   2.2 选择适当的座位坐下；
   2.3 听老师讲解并记好笔记。
3. 复习：
   3.1 重新阅读教材和课堂笔记，回忆教师的讲解思路；
   3.2 阅读参考书；
   3.3 如果存在需与同学讨论的问题，则与同学讨论；
   3.4 如果存在需要请教师答疑的问题，则请教师答疑；
   3.5 检查是否还有疑问，如果有，则解决之；

3.6 完成各类练习。

大家可以发现求精后的描述比第一个描述更具体了，但它还是第一个描述所指的内容。我们可以对这个求精了的描述进行更进一步的求精。实际上，这就是一个算法的设计，这里用的是算法设计中逐步求精的方法。它的特点是，考虑问题时先整体，后局部，先抽象后具体。所以又称这种方法为自顶向下的设计方法。例如，在上述算法的设计中，我们第一步的工作是把一次课的内容的学习划分成三阶段来完成，在第二步的求精过程中，我们又分别考虑每个阶段的任务如何完成。

这种算法设计方法安全可靠，最终得到的程序的总体结构清晰、含义明确，易读性，易理解性好，也利于程序的维护和程序的正确性证明。此外，这种方法对初学者尤其适应。因为对程序设计的初学者而言，开始的第一大难关是编制一个问题的求解程序时，往往不知道从何处下手，一会想了这样一条语句，一会想起那样一条语句，东拼西凑，难以写成，往往当被问及某条语句写在那个地方起什么作用时，却说不清。如果使用这种自顶向下，逐步求精的方法，则可以先考虑解决此问题总体上需要几大步，每步需要解决什么样的问题。一般地，只要你一下子不去考虑得过细，这个总体框架是比较容易得到的，对一个科学计算问题尤其这样。在这个基础上，再分别研究每一步又需要如何去做。如此下去，不难得到一个适当的算法，从而编出一个“好”的程序。

例 1-2 求 100 以内的素数的算法。

素数是只有 1 和本身这两个自然数为其因数的，大于 1 的整数。显然，除 2 之外，所有的素数都是奇数。

求[1，100]范围内的素数，有许多方法。例如，可以按递增的顺序依次检查每一个满足条件 $2 \leqslant n \leqslant 100$ 的自然数 n，看它是否为素数，也可以对这个指定范围中的自然数按各种条件进行一次次的筛选。我们选用第一种方法。首先我们想到的是 2 是素数，小于 100 的其它素数可在 3 到 100 中的奇数中找。从而有最初的(也是最粗的)算法：

1. 2 是素数；

2. 选出 3～100 中的素数。

第一次细化。为了方便，引入变量 n 表示被检查的数，我们让 n 从 3 开始取值，每次增加 2，分别检查，看 n 是否为素数，是，就输出它，不是，就检查下一个，如此重复，直到 n 的值大于 100。这样 2 就可以细化为

2.1 n 取 3；

2.2 n 是素数吗？是则输出它，不是则继续；

2.3 n 取下一个被检查数；

2.4 如果 n 未超过 100，则转 2.2 继续检查。

在以上各步中，步骤 2.2 需要进一步地细化，也就是说，对一个自然数 n，我们怎样才能回答“n 是素数?”这个问题。为此，我们看 n 是否可以被 2～n－1 的数所整除。这样用 m 作为除数，用 P 作为素数的标志，2.2 被细化为

2.2.1 m 取 2；P 取“真”；

2.2.2 检查 n 是否可被 m 整除，如果可以，说明 n 不是素数，则让 m 取 n，P 取假；

2.2.3 m 取下一个数；

2.2.4 如果 m 小于 n，则转 2.2.2 继续进行测试；

2.2.5 如果 P 为真，则输出素数 n。否则，继续。

下面我们把这个算法完整地写出来，同时引用一些符号，使得算法更易读。

1. 输出素数 2；

2. 选出 3～100 中的素数：

2.1 n＝3(表示 n 取 3，下同)；

2.2 判断 n 是否素数；

2.2.1 m＝2；P＝“真”；

2.2.2 如果 mod(n，m)＝0，则 m＝n，P＝“假”；

2.2.3 m＝m＋1；

2.2.4 如果 m＜n，则转 2.2.2；

2.2.5 如果 P 为“真”，则输出 n。

2.3 n＝n＋2；

2.4 如果 n≤100 则转 2.2；

3. 结束。

注意：上述算法中 m＝m＋1，n＝n＋2 分别表示变量 m 和 n 分别取 m＋1 和 n＋2 的值，即在原来值的基础上加 1 或 2。另外，在没有注明向某个地方转移时，则顺序执行下一条。如 2.2.4 的条件 m＜n 被满足时，则转 2.2.2 执行，否则就继续执行下一条(2.2.5)。2.2.2 中的 mod(n，m)表示 n 除以 m 的余数，mod(n，m)＝0 不是 mod(n，m)取 0，而是判定 mod(n，m)是否为 0，即 m 是否可以除尽 n。所以符号“＝”有时表示“等于吗?”有时表示取某值。根据上下文就可以区别它的这两种意义。

算法被细化到这一步，就可以编程了。编写程序时，要根据程序设计的要求(如结构化)，利用语言提供的各种语句对设计好的算法进行描述。

这里给出的是求素数的一个简单的算法，其效率是不太高的，读者可以根据自己掌握的有关素数的知识选用其它的方法来选素数。如当判断 n 是否为素数时，并不需要从 2 除到 n－1，只须除到 $\sqrt{n}$ 就可以了。此外，我们给的这个算法将选素数的范围限制在 100 以内，请你考虑一下，如何修改此算法，使它能选出 1000 以内的素数来。进一步地，如何修改它，使之能选出任意指定范围内的素数。例如，所有小于 50 的素数，所有小于 505 的素数，所有大于 91 小于 1001 的素数等等。

构造一个好的算法，就象写作一样，应该好好地加以组织。在开始，必须先有一个总体计划，然后再增加细节。自顶向下逐步求精的方法就象拟一个算法的提纳。第一步设计总体计划就象写作时先列出的主要论题，在后续的设计中就相当于写出提纲中的小标题，然后是一些具体内容的概括。通过逐步地把这个计划细化，最后达到可以直接指导编程的目的。

例如，我们要写一个算法 A 来求解一个问题 P。开始时，我们只有关于如何做的很粗略的想法，例如这个问题的求解过程实际包含三个部分：A1、A2、A3。这相当于最上层的设计，我们用 A1、A2、A3 代替了 A。继续分别对 A1、A2、A3 进行分析，也许会发现它们分别可以分成更小的问题，这可看成是第二层的设计。如此下去，最后我们就可以得到一个可以用计算机语言编写程序的算法了。

简而言之，这种设计方法的基本思想是：先完成算法的总体设计，把它编写成一般的执行步骤序列，然后用同样的方法充实每一个步骤的细节。

例 1-3 求一组数的平均数。

现在我们来考虑如何构造一个算法，让计算机找出这一组数的平均数。开始时，想得比较简单，算法分两大步：

1. 计算平均数；

2. 输出此平均数。

为了补充细节，就要考虑这组数的平均数的计算过程，也就是对第一步的细化：

1.1 读入每一个数，并计算这些数的和和这批数的个数；

1.2 用和除以个数，得平均数。

为了保证设计的正确性，对你设计的每一步都应问一下：它的作用能达到否？它能否被顺利执行？某一步的几个子步是否确保这一步的功能被实现？当然，一般我们在设计小型计算程序时还不需要对这些问题给出严格的理论证明，只用进行一些必要的验证就可以了。验证中尤其要对一些例外情况多加注意。如本例中在计算平均数时，用到了除法，而当除数是零时，除法是无法进行的。所以，在除之前应该验证一下除数是否为零。

我们再对 1.1 进行细化，描述出如何读数、计算和、计算这批数的个数。为此，引入表示这批数的和的变量 SUM 和个数的变量 COUNT，为方便起见，我们用变量 A 表示每次读入的数。我们的思想是逐个地读入每一个数，把计数加 1，并把它加到和中，这样 1.1 可以改成：

1.1.1 使 SUM 和 COUNT 取 0；

1.1.2 对这批数中的每一个，完成如下操作：

a. 读入该数 A；

b. 把它的值加到 SUM 中；

c. 个数 COUNT 加 1。

对整个算法，可以完整地描述如下：

1. 计算平均数：

1.1 计算这些数的和与个数：

1.1.1 SUM=0;COUNT=0；

1.1.2 对这批数中的每一个数：

a. 将此数读入 A 中；

b. SUM=SUM+A；

c. COUNT=COUNT+1。

1.2 求平均数：

1.2.1 如果 COUNT=0 则 AVERAGE=0；

否则 AVERAGE=SUM/COUNT。

2. 输出 AVERAGE；

3. 结束。

二、算法的特性

算法作为对解题方法和步骤的精确描述，具有五个重要特性：有穷性、确定性、有效性、有输入性和有输出性。先看如下例子然后参考这个例子看这五个特性。

例 1-4 给定两个正整数 m、n，求它们的最大公因数。

最大公因数就是能够同时整除 m 和 n 的最大正整数，下面是著名的欧几里德算法(简称算法 E)：

E1：[输入参数]输入正整数 m、n(m≥n)；

E2：[求余数]以 m 除以 n，并令 r 为所得的余数(显然有 0≤r<n)；

E3：[余数为 0?]如果 r=0 则结束，输出结果 n；

E4：[交换]m=n，n=r，即将 m 与 n 的值分别置成 n 与 r 的值，转 E2。

1. 有穷性

一个算法总是在执行有穷步之后结束。所以，一个算法是一个有穷的动作序列。算法 E 满足这个条件：因为在步骤 E2 之后，r 的值小于 n，所以，如果 r≠0，步骤 E4 被执行，它把 m 的值改成 n 的值，把 n 的值改成 r 的值，由于 r<n，所以当再次执行 E2 时，n 的值已减小了，而对于任意给定的正整数 n，从它开始的递减的正整数序列必然是有限的，并且该序列的长度不大于 n。所以对于任意给定的正整数 m、n，E2、E3、E4 只能被执行有穷次，而 E1 只被执行一次。

事实上，有穷性应该是有一定限度的，我们通常要求算法能在适当的时间内执行结束。对一个执行时间比较长的算法，比如说要连续执行一个小时以上，我们建议它能以一定的时间间隔输出计算的中间结果或表示算法在正常被执行的信息，以便使用户对算法的运行情况了如指掌，避免不必要的疑虑和无效的运行。

2. 确定性

算法的每一步必须有明确的意义。例如，算法 E 的第一步 E1 明确指出输入的是两个正整数 m、n，并且 m≥n。实际上，这是保证算法总能正常运行的条件。E2 则明确指出了是 m 除以 n，得余数 r，而这里说的“除以”和“余数”的意义则认为是众所周知的。此外，关于确定性，我们还要求构成算法的动作序列有一个初始动作，且序列中的每一个动作仅有一个后继动作，结束动作再无后继动作。

3. 有效性

算法的每一步都能被有效地执行，并得到确定的结果。例如算法 E 中，我们是在确认 r 不为 0 时，才将它的值给 n，这时 n 才能作为除数参加运算，否则，n 是不能作为除数参加运算的，因为一个数除以 0 是无意义的。

算法的有效性保证了算法在结束时，能对求解的问题给出一个适当的回答。算法 E 就能给出要求的最大公因数。

4. 输入

一个算法有 0 个或多个输入。即在算法运行中，能够从外界获得必要的参数，然后根据这些参数给出计算结果。算法 E 的输入参数就是 m 和 n。当然，如果我们将此算法的功能限制在只计算某两个数，比如 12 与 6 的最大公因数，则可将 E1 改为：取 m 为 12，取 n 为 6，这样，算法的输入就是 0 个。

算法的输入相当于函数的自变量，所以又称之为输入参数或入口参数。

在例 1-2 中，我们给出的算法的输入是 0 个，若要将算法改得能求小于任一给定正整数的所有素数，则需将所给算法中的 100 改为一个输入参数，并在算法的开始读入这个参数。即在所给算法的第一步之前加进读入这个参数的动作。若要将算法改得能求大于一个给定

的正整数，小于另一个给定的正整数的所有素数，则要改成有两个输入参数的，另外，算法的其它部分也需加以修改，但主要的算法思想是不变的。

5. 输出

一个算法有一个或多个输出。算法的目的是进行问题求解，输出就是将求解的结果通知“用户”。没有输出的算法是无意义的。如算法 E 输出的是 n 的最终值。

算法的输出又叫输出参数或出口参数。

在现代的算法设计中，对一个算法，一般要知道的信息是功能、输入参数、输出参数，这在模块化方法中尤被强调。

例 1-5 设计求 $\sum_{i=1}^{N} i$ 的算法($N \geqslant 1$)。

这是很一般的一个求和计算，所以我们直接给出如下算法：

算法一：

1. S=0;
2. 输入 N;
3. I=0;
4. 当 I<N 时重复如下操作：
   4.1 I=I+1;
   4.2 S=S+I。
5. 输出 S;
6. 结束。

算法二：

1. S=0;
2. 输入 N;
3. 对 I=1 到 N，重复如下操作：
   3.1 S=S+I;
4. 输出 S;
5. 结束。

算法三：

1. S=0;
2. 输入 N;
3. 如果 N≤0 则转 5;
4. S=(1+N) * N/2;
5. 输出 S;
6. 结束。

这三个算法都是题目所要求的求和算法。算法二不总是有效的，当读入的数 N 不满足题目要求时，它就不能正常工作。另外我们还可以看到两点：首先，一个问题的求解可以有不同的算法，一个功能的完成也可以用不同的实现方法。如算法一中的 3、4 与算法二中的 3 在正常的情况下完成的功能是一样的，因此，请读者在设计中选取“好”的形式。其次，算法三比算法一、算法二要简洁的多，而且它用一个乘法和一个除法代替了一系列加法及其辅助操