前　言

1995 年 2 月，国家教委师范司制订了《高等师范学校计算机应用基础教学大纲》，根据《大纲》的要求，结合我们几年来为高师非计算机专业开设计算机应用基础课程的实践以及计算机技术的发展，我们把计算机应用基础知识分为两个部分：第一部分包括计算机的基础知识与操作系统、文字处理与制表、数据库管理系统等内容，这些称为“计算机文化基础”；第二部分包括程序设计语言与算法、计算机教育应用、多媒体技术等，这些称为“程序设计基础”。第一部分内容已在《计算机应用教程》(北京大学出版社，1995)中介绍，曲阜师范大学学生使用了此教材，在“山东省普通高校计算机等级考试(初级)”时成绩突出，此教材还在其他多所院校使用，同样取得了较好的成绩，现已印刷二次。第二部分内容就是我们编写的这部教材，本书取名为《程序设计基础教程》。我们编写本教材的目的是让学生在学习了“计算机文化基础”课程的基础上，进一步学习和掌握一种程序设计语言，具有程序设计和上机调试程序的能力；并了解计算机辅助教育的基本原理和方法，学习与其相关的多媒体技术，学会使用 Authorware 制作多媒体课件，为今后从事教育工作、适应教育的重大变革打好基础。

本书共分十三章。第一章介绍了程序设计语言的基本知识和计算机算法的基本概念。第二至八章讲述了 C 语言及其程序设计。C 语言是世界上广泛使用的计算机高级语言之一，它是一种结构化的程序设计语言，具有简单、高级、可靠的特点。通过这几章的学习，学生可掌握结构化程序设计的基本思想和方法，学会编制 C 语言的程序，能进行数值计算和非数值信息处理。第九至十一章介绍了计算机辅助教育的基本原理和教学软件设计的理论和方法，使学生具备运用计算机进行辅助教学和管理教学的能力。第十二章介绍了多媒体技术。多媒体技术近年来迅猛发展，已渗透到人类生活的各个领域，现代大学生对这门新技术应该有较深的了解。第十三章详细讲了多媒体开发工具 Authorware，学生通过该章内容的学习，学会制作多媒体课件的方法，为今后走向工作岗位后进行计算机辅助教学奠定基础。

本教材的计划授课时数为 72 学时，另外安排 36 小时以上的上机练习。

本书在编写过程中查阅了许多参考书，谨在此向这些作者们表示诚挚的谢意。本书的出版得到曲阜师范大学教务处和曲阜师范大学数学与计算机科学系领导的大力支持，也受到了国家教委世界银行贷款的资助。

由于编著者水平所限，难免会有错误之处，欢迎读者指正。

编　者

2000 年 6 月

第一章　算法与程序设计

§1.1　程序设计语言的概述

要使计算机按人的意图工作，就必须使计算机懂得人的意图，接受人向它发出的命令和信息，这就要解决一个“语言”的问题。计算机并不懂人类的语言，无论是汉语或英语，计算机都不能接受。因此要求人们用特定的语言与计算机交谈，这就是计算机语言。

从1946年第一台电子计算机“ENIAC”问世到现在，计算机语言经历了三个阶段：机器语言阶段，汇编语言阶段和高级语言阶段。现在已经出现了第四代语言。

机器语言就是计算机的指令系统。机器指令系统中的每条指令是一串二进制数代码。例如：1011011000000000，在某一类型的计算机中这条指令的作用是告诉计算机做一次加法运算。一般地，一条指令用来控制计算机进行一个操作，它告诉计算机应进行什么运算，哪些数参加运算，这些数据存放在什么地方，计算结果送到什么地方去等。用机器语言编写程序就是要写出由一条条指令组成的程序。计算机执行程序时，每条指令中的0和1通过线路变成电信号，使得计算机完成各种不同的动作。注意每种型号的计算机都有自己规定的指令系统，一般来说，不同型号的计算机的机器语言是互不通用的。我们用甲型机器的机器语言编写的程序在乙型机器上就不能用，若要在乙型机器用需要重新编程序。因此机器语言通用性差，这对计算机的推广应用造成很大困难。另外，用机器语言编写程序十分繁琐，首先需要背会各种代码和它的含义，然后再用0和1去组合程序。这种程序直观性差、易出错、难检查、难修改、难调试。

随着计算机的蓬勃发展和计算机速度的迅速提高，改变编写程序的方式显得尤为迫切。很快人们创造了“汇编语言”，它用一种特定的助记符号代表数字代码，例如机器语言中用“01”表示相加，汇编语言中用“+”表示加法运算，用汇编语言编出的程序较机器语言直观、易懂。汇编语言的一个语句与机器语言的一条指令一一对应。汇编语言编写的程序不能在机器上直接执行，需要用汇编程序将汇编语言的程序生成目的程序（机器语言程序），计算机直接执行目的程序。用汇编语言编程序与用机器语言编程序的步骤相似，仍然繁琐、工作量大，并且依赖于特定的计算机，通用性差。所以人们仍将汇编语言称为面向机器的语言。

从机器语言到汇编语言，使程序的直观性、易调试性有了较大的进步，但是仍然要求程序员必须知道计算机的工作原理和内部结构，只有训练有素的专业人员才能用好这种语言。为了克服一般用户使用计算机的困难，为了提高程序的易读性与程序设计的效率，人们花费很大的精力研究了“高级语言”，或称为“算法语言”或“程序设计语言”。有了高级语言之后，人们可以不必学习机器指令，也不需要懂得计算机的工作原理和内部结构，就能编出程序在计算机上解题，这就为计算机的推广使用扫除了极大的障碍。

用高级语言编写的程序既和人们习惯用的语言和数学公式相似，又能为计算机所接受，但是用高级语言编写的程序（称为源程序）计算机并不能直接接受和执行（它只能接受0和1组成的代码），因此必须要有“翻译”，把高级语言编写的程序翻译成计算机能接受的机器指令程序（称为目的程序），然后计算机再执行机器指令。这种“翻译”，通常有两种方式，即编译方式和

解释方式。

编译方式是：事先编好一个称为编译程序的机器指令程序，并放在计算机中。把用高级语言编写的源程序输入计算机，编译程序便把源程序整个地翻译成用机器指令组成的目的程序，然后执行该目的程序，得到计算结果，如图 1.1 所示。

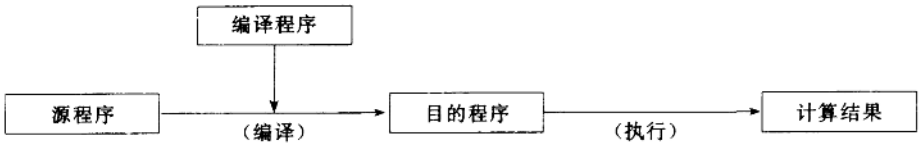


图 1.1

解释方式是：事先编好一个称为解释程序的机器指令程序，并放在计算机中。当把高级语言编写的源程序输入计算机后，可以马上执行这个程序，执行的过程是解释程序将源程序逐句地翻译，译出一句执行一句，即边解释边执行，如图 1.2 所示。

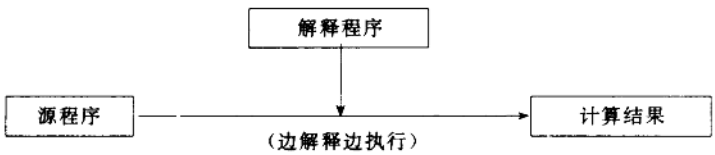


图 1.2

从 50 年代后期开始，各种各样的程序设计语言大量地涌现出来，到现在至少也有几百种，经过了长时间的“自然选择”，优胜劣汰，现在常用的有十几种，例如 FORTRAN，PASCAL，C，ALGOL，COBOL，BASIC，LISP，PL/1，Ada 等。从理论上讲，各种程序设计语言都能以编译方式处理，也能用解释方式工作。通常 FORTRAN，PASCAL，C 等高级语言总是采用编译执行方式，这些语言称为编译式语言；BASIC、LISP 等语言多是采用解释执行方式，被称为会话式语言。编译执行方式的好处是翻译的结果是一个完整的目的程序，将目的程序存储在外存储器中，以后运行时（不论执行多少次）就不必再翻译了，可以直接调用，从而节省了再翻译的时间，提高了执行速度。解释执行方式的好处是通过人机对话的方式进行程序的调试，在计算机上输入一段程序后马上执行，有错时机器立即显示出错信息，程序员修改后再马上执行，这样反复多次，直到将程序调试正确。

用高级语言编写的程序是由一系列的语句（或函数）组成的，其中的每一个语句是由编译程序（或解释程序）自动地转化为几条、几十条、甚至上百条机器指令。写一条这样的语句所需要的时间和写一条机器 指令所需的时间相差并不多，所以用高级语言编写程序可以大大提高编制程序的效率。另外，由于高级语言的书写方式更接近于人们的思维习惯，这样的程序更容易读和理解，不容易出错，有错时也更容易检查，这给程序的调试和维护带来了很大的方便，因而通常的程序都是选用某种高级语言来编写。

使用高级语言编写程序还有一个很大的优点，就是它可以适用于不同的计算机，或者说它具有通用性。用某一种高级语言编写的源程序几乎可以不加修改（有时只需作很小的修改）就能在不同的计算机上使用。应当指出的是不同的计算机执行的指令系统并不相同，所以不同型

号的计算机其编译程序(或解释程序)是不同的,只有这样,相同的源程序才能翻译出不同的计算机各自需要的目的程序。

高级语言虽然具有通用性,但这只是与机器语言和汇编语言相比较而言,而不是说各种高级语言的功能都是一样的。实际上人们总是根据自己工作中的需要和要求来编写程序,问题的领域不同,所属的专业不同,处理问题的技术和方法也会有所不同。所以各种计算机高级语言并不是真正对所有各个领域中的问题都同样有效和功能完全相同。例如 FORTRAN, C, BASIC 等语言更适用于科学计算,COBOL 语言主要用于商业管理,C 语言常常被用来编写系统软件,LISP 语言用在人工智能领域。

高级语言虽有众多优点,但是执行速度比不上同样功能的低级语言(通常人们称机器语言与汇编语言为低级语言),所以在像过程控制这样的实时系统中,至今仍存在使用汇编语言为主的现象。尽管程序设计语言不断发展,一些新出现的语言功能也不断加强,但是我们始终坚信,过去没有,现在没有,恐怕将来也不会有那样一种语言,对任何应用场合都是最好的。不同的语言各有千秋,互有长短,应用场合各异。作为一名计算机应用工作者,学习掌握很多种程序设计语言,既无太大的必要,也很困难。应该根据自己的工作特点选学一二种语言,熟练掌握和应用之后,如果需要学习新的语言时,应该将新语言与你所掌握的语言进行比较,可以发现程序设计语言纵然千变万化,其普遍规律是相同的,比如基本结构包括顺序、分支、循环,执行方式包括解释、编译等,通过比较语言的异同,易于记忆和掌握一种新的语言。

随着计算机科学的发展和软件开发技术的需要,新的程序设计语言层出不穷,一些老的程序设计语言也在不断地发展和更新,不断地推出新版本和新的系统语种。下面仅就最常用的 FORTRAN,BASIC,PASCAL,C 几种语言的发展给予简单的介绍。

1. FORTRAN 语言

FORTRAN 语言是目前国际上广泛流行的一种高级语言,主要用于科学计算。第一种 FORTRAN 语言是 1954 年推出的,1956 年开始实际使用。以后发展出许多不同的版本,例如 FORTRAN Ⅱ (1958 年), FORTRAN Ⅲ (1962 年), FORTRAN Ⅳ (1966 年,也称为 FORTRAN66),FORTRAN77(1977 年),这个发展过程使得 FORTRAN 从非结构化语言发展成为结构化程序设计语言。1990 年 3 月,国际标准化组织和美国国家标准化组织,双重批准了 FORTRAN 语言最新国际标准文本 FORTRAN90。目前 FORTRAN90 的某些内容仍处于变动之中。我国也积极参与了 FORTRAN90 的标准文本设计工作,将其国产化,是我国"八五"攻关项目之一,我国标准化组织也正在根据 FORTRAN90 确定我国的 FORTRAN 标准文本。FORTRAN90 的抽象数据类型和模块结构具有数据封装和隐藏机制,顺应了 90 年代面向对象和程序设计语言的发展趋势。

2. BASIC 语言

BASIC 语言是当前使用人数最多(包括中、小学生)、最为普遍的语言。BASIC 语言于 1964 年问世,其后不断地有各种各样的修改和扩充。1971 年建立了一个文本,1973 年建立了"标准 BASIC"文本说明,80 年代前期最流行的有 MS BASIC,PC BASIC,BASICA,GWBASIC。80 年代后期出现了第二代 BASIC(True BASIC,Turbo BASIC,Quick BASIC),使得 BASIC 从解释性的非结构化的语言发展成为结构化的程序设计语言,并且成为既有解释器又有编译器的两栖语言。90 年代的第三代 BASIC(GFA BASIC,Visual BASIC for Windows,Visual BASIC for DOS)正迎着面向对象的程序设计思想前进,初始的 BASIC 只有 17 种语句和 11

个内部函数，而 True BASIC 已经有 155 种语句和 43 个内部函数。Quick BASIC 共有 264 个语句和内部函数。数据类型也较第一代 BASIC 有了很大的改进，有了局部量和全局量之分，增加了用户自定义类型和动态数组功能。控制结构也吸收了其他语种的优点，还可以用 FUNCTION 定义递归的函数过程。到了第三代的 Visual BASIC 又增加事件驱动和可视界面等功能，引入了窗体(Form)和控件(Control)的概念。使过去需几天、几周甚至几月才能完成的编程任务，在几个小时之内就完成了。

3. PASCAL 语言

PASCAL 语言是第一个结构化的程序设计语言，也是一种系统设计语言。可以用它编写应用程序，也可以用来编写编译程序一类的系统软件。PASCAL 语言是瑞士苏黎世联邦大学的N·沃斯教授于 1968 年设计完成，1971 年正式发表，它的功能强、数据类型丰富、适用面广、编译程序简单，是 70 年代影响最大的一种算法语言。经过二十多年的发展，到 Turbo PASCAL 6.0 版本，具有了图形功能和面向对象的功能。

4. C 语言

C 语言具有简洁、紧凑、灵活的特点，有其他高级语言所具有的丰富的控制结构和数据结构，并且兼有低级语言的大部分功能。1972 年贝尔实验室的 D. M. Ritchie 设计了 C 语言，最初的 C 语言只是为了描述和实现 UNIX 操作系统提供一种工作语言而设计的。直到 1975 年 UNIX 第六版公布后，C 语言的实际优点才引起人们的普遍注意，1983 年美国国家标准化协会(ANSI)公布了 ANSI C，使之成为美国国家标准。目前，在微机上广泛使用的是 MS C，Turbo C 和 Quick C 语言。当前 C 语言的发展正方兴未艾，不仅系统软件设计人员而且应用软件设计人员也愈来愈多地使用 C 语言作为编程工具。另外 C 语言是结构化程序设计语言，数据结构和控制结构十分丰富，核心部分又很精炼，作为教学用语言是比较理想的。看来 C 语言在相当一段时期内，仍将保持编制软件的首选语言的地位。但 C 语言也有自己的缺点和不足，对于一般的数值计算、工程计算等方面的课题，它并没有突出的优势，还存在着数据类型分类不精确、求值二义性等缺点。此外，尽管 C 语言对于编制系统软件和应用软件很合适，充分发挥了其接近硬件、灵活高效、代码简洁等特点，但据统计，当 C 程序的规模达到 25000 行以上时，就难以维护，难以修改。这是因为传统语言中数据和程序是分离的。而新的面向对象的语言，把数据及其上的操作封装起来，成为核块。所以对于大型和超大型软件，用 C 语言来编制则感不便，而应使用面向对象的 C++语言了，“++”表示是对 C 语言的改进。从 C 到 C++是一个自然的平滑的过渡。C++与 C 兼容，可使大量 C 程序继续使用，在 CAD，CAM，CIMS，CASE，人工智能，专家系统等领域使用 C++后可以提高编程效率、缩短开发时间。

以上我们介绍的程序设计语言一般称为第三代语言，随着计算机科学的发展和计算机应用的普及，为了让广大非计算机专业用户方便地使用计算机以提高其工作效率，人们正在努力寻求一种比高级语言更简洁易学、效率更高的语言，这就是第四代程序设计语言 4GL。这类语言提供给使用者的友好界面，具有菜单驱动模式，要求用户记忆的东西很少，只需在菜单上做某些选择就能完成所需要的操作并获得结果。用 4GL 书写程序的行数应该比用其他高级语言书写的行数成倍的减少，运行效率则应成倍的提高。总之 4GL 是一种只需告诉机器做什么而无须具体指明运算的过程就可以自动执行的更易学易用的高级程序设计语言。目前虽有不少软件或语言自诩为“4GL”，如 1969 年诞生的 FORTH 语言，但达不到 4GL 的要求。倒是一些数据库查询语言及电子表格软件包已具备了 4GL 的主要特点，这类软件包既有非过程化的描

试读结束：需要全本请在线购买： www.ertongbook.com

述能力，又提供了过程化的编程能力。我们期待着真正公认的 4GL 尽早出现。

总之，程序设计语言正在快速的发展中。各种程序设计语言都力求支持结构化程序设计；面向对象的程序设计语言的设计思想已经渗透到各个高级语言中；简单易学、效率更高的第四代语言正在出现。社会的需要、科学的进步促使老语种不断的更新，新的语种层出不穷。

§1.2 计算机算法

学习程序设计，不仅要掌握所使用的一种计算机语言（如 C 语言），还要掌握解题的方法和步骤。语言只是一个工具，只懂得语言的规则并不能保证能编出有效的高质量的程序。我们这里要讲的计算机算法就是研究在计算机上解题的步骤和方法，对于实际给出的问题，经过对此问题的分析，确定解决问题的方法和步骤，即确定“算法”，然后根据算法编出计算机执行的程序，最后执行这个程序得出需要的结果。为了更清楚地理解“算法”这个概念，掌握设计算法的方法，下面举两个计算机算法的例子。

例 1 求解一元二次方程 $ax^2+bx+c=0$，其中 a,b,c 都是实数。

根据初等代数的知识，我们知道一元二次方程的解可以按照下面方法给出：

如果 $a\neq 0, d=b^2-4ac\geqslant 0$ 时，方程有两个实根：

$$x_1=\frac{-b+\sqrt{d}}{2a},\quad x_2=\frac{-b-\sqrt{d}}{2a}。$$

当 $d=b^2-4ac<0$ 时，方程有两个复根：

$$x_1=\frac{-b+\mathrm{i}\sqrt{-d}}{2a},\quad x_2=\frac{-b-\mathrm{i}\sqrt{-d}}{2a};$$

如果 $a=0$，但 $b\neq 0$，则原来的方程退化为一次方程 $bx+c=0$。这时方程有一个解：

$$x=-c/b;$$

如果 $a=b=0$，但 c 不为 0，则原来的方程是矛盾方程，无解；

如果 $a=b=c=0$，则 x 可以是任何数，原来的方程有不定解。

根据上述的分析，我们可以给出下面的算法：

① 输入方程系数 a,b,c；

② 如果 $a=0$，则转⑨；

③ 令 $d=b^2-4ac$；

④ 如果 $d\geqslant 0$，则转⑦；

⑤ 计算：

$$x_1=\frac{-b+\mathrm{i}\sqrt{-d}}{2a},\quad x_2=\frac{-b-\mathrm{i}\sqrt{-d}}{2a};$$

⑥ 输出方程的复根 x_1,x_2，转⑭；

⑦ 计算：

$$x_1=\frac{-b+\sqrt{d}}{2a},\quad x_2=\frac{-b-\sqrt{d}}{2a};$$

⑧ 输出方程的实根 x_1,x_2，转⑭；

⑨ 如果 $b=0$，则转⑫；

⑩ 令 $x=-c/b$；

⑪ 输出方程的根 x；转⑭；

⑫ 如果 $c=0$，方程有不定解；转⑭；

⑬ 方程无解；

⑭ 停止。

容易看出，这个算法可以用来处理所有实系数的一元二次方程的求解问题。

例 2 有雌雄一对兔子，假定过两个月可以繁殖雌雄一对小兔，小兔过两个月后又可以繁殖，问 n 个月时有多少对兔子？（假定 $n\geqslant 3$）

这是 Fibonacci 数列的例子，假定满 n 个月时兔子的对数为 $F(n)$，那么第 $n-1$ 个月的兔子对数为 $F(n-1)$，因为第 $n-2$ 个月所有的兔子到第 n 个月都有繁殖能力了，所以当月新生的兔子对数为 $F(n-2)$，则

$$F(n)=F(n-1)+F(n-2)。$$

按照例中的规定，$F(0)=0$，$F(1)=1$。根据上面的公式，我们可以计算出 $n=2,3,4,5,\cdots$ 时，$F(n)$ 分别为 $1,2,3,5,8,13,\cdots$。

下面给出计算 $F(n)$ 的算法：

① 输入月份数 n；

② 令 $F(0)=0$，$F(1)=1$，$x=2$；

③ 计算：$F(x)=F(x-2)+F(x-1)$；

④ x 增加 1；

⑤ 若 $x\leqslant n$ 转③；

⑥ 输出结果 $F(n)$；

⑦ 停止。

从以上两个例子可以看出算法应具有以下几个特点：

第一，有限性。当一个算法被执行时，必须在有限步以后停下来，结束该算法的执行，给出相应的结果，而不能无限地执行下去。尽管这里所说的有限步可能是几十步或者几百步，也可能是几万步或者几亿步，但它仍然是有限步。（严格说来，这里所说的“有限”只是说明了它和“无限”的区别。在实际工作中还有一个更严的限制，即实际工作所要求的时间界限的限制。算法和根据算法所编制的程序，必须在实际工作所规定的时间内给出相应的运算结果。）

第二，确定性。算法中的每个步骤都必须是确定的，在各种不同的情况下应该做什么都有明确的规定，而没有任何含糊不清的地方。（这里所说的确定性实际上也包含两方面的含义。第一是步骤的确定性，即算法中对各种不同情况下应该做什么都有明确的规定；第二是每一个步骤的执行结果都是惟一确定的，没有第二种可能。）

第三，通用性。一个算法总是针对某类问题来设计的，所以对于求解某类问题中的任何一个问题都应该是有效的。就是说，这类问题中的任何一个在使用该算法来处理时，都能正确地给出结果。而不是只能有效地处理这类问题中的一部分，其余的部分就不能处理。（从这个意义上来讲，它在加工处理这一类问题时具有普遍的意义。当然，这里所说的普遍意义仍有一定限制，即限于处理该类问题。）

正确地理解算法的概念是重要的，它是计算机科学中的一个基本概念，是了解计算机应用的基础，而且也是我们学习本课程的基础。

对于同一个问题，我们可能有若干种不同的算法，这些算法都可以用来处理这个问题。那么如何来评价这些不同的算法？对于算法的评价有许多标准。例如算法的清晰程度，算法是否容易读懂，算法的简练程度等等。但基本的标准是两个。一个是时间标准，一个是空间标准。所谓时间标准，简单说来，即执行这个算法需要多少时间，基本的原则是时间越短越好。对于同样的问题，如果用算法 A 和 B 分别进行处理，其结果是算法 A 所用的时间更短，那么我们就认为算法 A 更好一些。当然，这是一种比较粗糙的说法，详细的比较需要给出明确的指标和数量界线。所谓空间标准，即执行这个算法需要占用多少资源（可以理解为占用了多少计算机存储单元），基本的原则是资源的占用越少越好。对于同样的问题，如果用算法 A 和 B 分别进行处理，其结果是算法 A 占用的资源更少，那么我们就认为算法 A 更好一些。但是随着计算机技术的发展，硬件性能不断提高，以及软件研制水平和规模的发展，程序的规模越来越庞大，算法的清晰程度成了一个非常重要的问题。对于一个比较复杂的问题来说，如果所给出的算法让人无法读懂，那么这不能说是一个好的算法，因为按照这样的算法所编制出来的程序在软件的维护工作中将带来无数的麻烦。

为了提高算法的清晰度，需要选择一种合适的描述算法的工具。常用的描述工具有：自然语言，流程图，N-S 结构流程图，PAD 图，伪代码等。我们下面仅介绍最常用的程序流程图与 N-S 结构流程图。

1. 程序流程图

程序流程图也称为程序框图，人们学习程序设计语言编写程序很早就使用它，至今仍然是我国程序设计人员最普遍采用的一种工具，并且仍然为初学者掌握程序设计方法的辅助手段。程序流程图为人们普遍采用是因为它具有一些特有的优点。比如，它能把程序执行的控制流程顺序表达得十分清楚，看起来也比较直观易懂。许多计算机类的书用这种流程图描述算法，读者应熟练掌握流程图的用法。

需要指出的是，传统的流程图存在着一些严重的缺点，例如使用的符号不规范，表示程序控制流程的流线的使用灵活性太大，这些与现代软件工程化的要求是相背离的。为了消除这些缺点，我们对流程图所使用的符号做出严格的定义，对流线的使用适当限制，不允许人们随心所欲地画出各种不规范的流程图。

首先我们给出流程图中经常要用到的一些符号（参见表 1.1）。它是我国参照国际标准 ISO1028－73 制定的《信息处理流程图图形符号》国家标准 GB1526－79 的一部分，GB1526－79 规定了 30 多种图形的符号，这里所给出的是其中的一部分。以后我们画流程图时，就要使用这些标准符号。

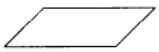
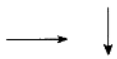
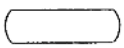
然后我们给出三种基本结构，这三种基本结构是 1966 年 Bobra 和 Jacopini 提出的，要求在画流程图时仅使用这三种基本结构，这是对使用流线的限制，目的是保证程序的结构化。并且人们已经证明，由这三种基本结构顺序组成的算法结构可以解决任何复杂的问题。这三种基本的结构是顺序结构、选择结构和循环结构。

（1）顺序结构。顺序结构含有多个连续的步骤，如图 1.3 所示。

在此结构中的各框（A 和 B）是顺序执行的，顺序结构是最简单的一种基本结构。

（2）选择结构。选择结构是由某个逻辑条件式的取值选择两个处理中的一个，如图 1.4 所示。

表 1.1　流程图中使用的符号表

符　号	名　称	意　义
	处理(框)	其他的规定符号定义所没有包括的处理功能都可以用这个符号表示，它有一个输入口和一个输出口
	判断(框)	表示需要在两个可以选择的路径上经过判断选择其中的一条路径，它有一个输入口和两个输出口
	输入/输出(框)	表示输入或者输出，即需要提供有关处理问题的数据，或者输出处理以后得到的信息
	文件(框)	表示文件，它可以是计算机内的，也可以是计算机外的
	流线	用来连接各个框的符号，通常它带有箭头，说明流程图的走向
	端点(框)	用来表示流程的开始或者终止

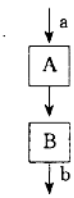


图　1.3

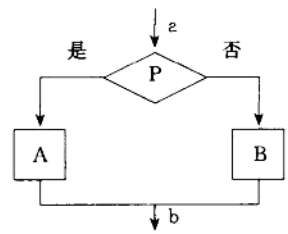


图　1.4

在此结构中有一个判断框，它有两个分支，根据 P 条件是否满足而分别执行“A”或“B”框。

(3) 循环结构。循环结构又称重复结构，有两种循环结构：

① 当型(WHILE-DO 型)循环结构，如图 1.5 所示。

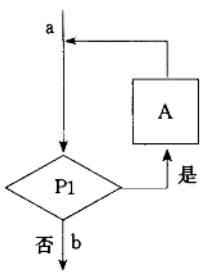


图　1.5

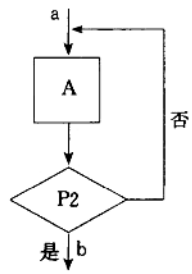


图　1.6

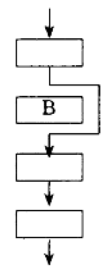


图　1.7

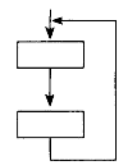


图　1.8

当 P1 条件满足时，反复执行 A 框。一旦 P1 条件不满足时就不再执行 A 框，结束执行本基本结构而执行它下面一个基本结构。如果在开始时，P1 条件就不满足，A 框一次也不执行。

② 直到型(UNTIL 型)循环结构，如图 1.6 所示。

先执行 A 框，然后判断 P2 条件是否满足，如 P2 条件不满足，则反复执行 A 框，直到某一时刻，P2 条件满足为止，此时停止循环，执行下面一个基本结构。可以看到，不论 P2 条件是否满足，至少执行一次 A 框。

请注意，以上两种循环结构的区别在于：① 当型循环结构是“先判断，后执行”，而直到型循环结构则是“先执行，后判断”。② 当型循环结构是当条件满足时执行循环，不满足时结束循环，而直到型循环结构则是条件不满足时执行循环，条件满足时结束循环。③ 直到型循环至少执行一次循环体(A 框)，而当型循环有可能一次也不执行循环体(A 框)。

这三种基本结构有以下共同的特点：

(1) 只有一个入口。图 1.3～图 1.6 的 a 点。

(2) 只有一个出口。图 1.3～图 1.6 的 b 点。

(3) 结构内的每一部分都有机会被执行到，也就是说，对每一个框来说，都应当有一条从入口到出口的路径通过它。图 1.7 中的 B 框是永远没有执行机会的，它不满足基本结构的要求。

(4) 结构内没有死循环(无终止的循环)。图 1.8 就是一个死循环，不属于三种基本结构的一种。

下面我们举例说明怎样用程序流程图描述算法。前面例 1 给出的求解一元二次方程的算法用流程图描述，如图 1.9 所示。例 2 给出的求 Fibonacci 数列的算法用流程图描述，如图 1.10 所示。

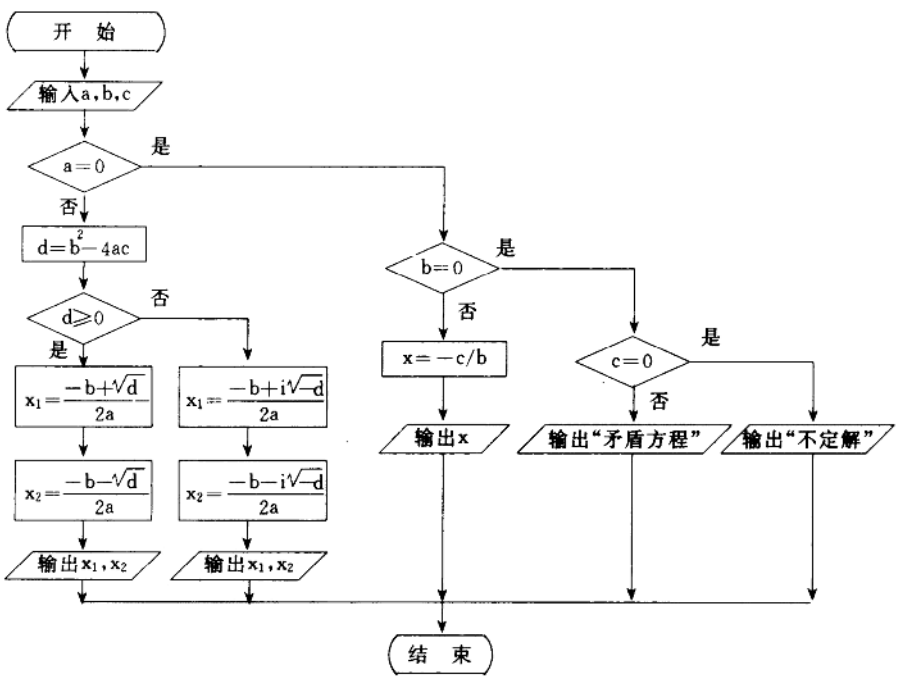


图 1.9

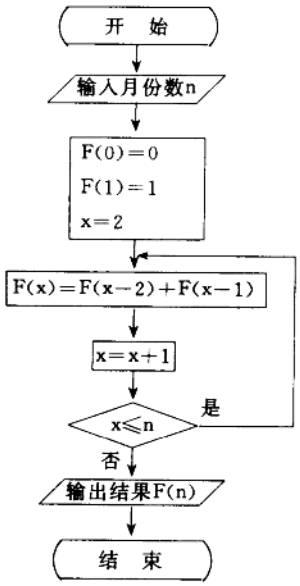


图 1.10

2. N-S 结构流程图

N-S 图的名称取自其创造者 Nassi 和 Shneiderman 两人名字的第一个字母。人们也称它为盒状图。实际上，N-S 图是流程图的一个变种。为了彻底地解决程序结构化的问题，N-S 图完全去掉了流程图中的控制流程的流线和箭头。因而完全排除了因任意使用控制转移对程序质量的影响。N-S 图把全部算法写在一个矩形框内，在框内还可以包含其他的框，换句话说，由一些基本框组成一个大的框，基本框就是代表三种基本结构的框。下面分别给予介绍。

(1) 顺序结构的 N-S 图

如图 1.11 所示，A 和 B 两个框组成一个顺序结构，它表示执行 A 后再执行 B。

(2) 选择结构的 N-S 图

如图 1.12 所示。它是一个整体，代表一个基本结构。它表示当条件 P 满足时执行 A，不满足时执行 B。

(3) 循环结构的 N-S 图

图 1.13 代表的是当型循环结构，它表示“当条件 P1 成立时反复执行操作 A”。

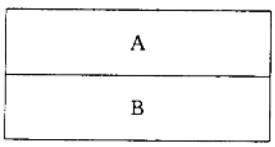


图 1.11

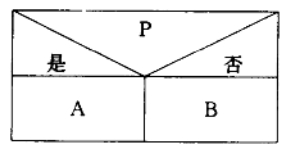


图 1.12

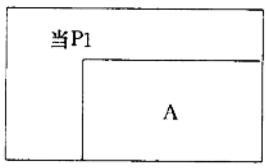


图 1.13

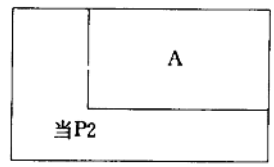


图 1.14

图 1.14 代表的是直到型循环结构，它表示“反复执行操作 A，直到条件 P2 满足时结束”。

用以上三种基本框可以表示任何复杂的算法，绘制出算法的 N-S 结构化流程图。仍以前面给出的例 1 和例 2 为例，说明怎样用 N-S 图表示一个算法。

求解一元二次方程的算法的 N-S 图，如图 1.15 所示。

求 Fibonacci 数列的算法的 N-S 图，如图 1.16 所示。

注意 N-S 图表达的算法随着层次的增加，从外向内逐步完成对算法的描述，图中每一个矩形都明确写出了应完成的功能。利用 N-S 图表示的算法必然是结构化算法。

另一方面，非结构化的算法用 N-S 图是无法表示的，可以考虑将非结构化的算法转换为结构化算法后，再用 N-S 图表达。例如图 1.17(a)所示的算法结构用 N-S 图是无法表达的，将此结构转换为功能相同的结构化的算法流程图 1.17(b)后，用 N-S 图表示为图 1.17(c)。

试读结束：需要全本请在线购买： www.ertongbook.com

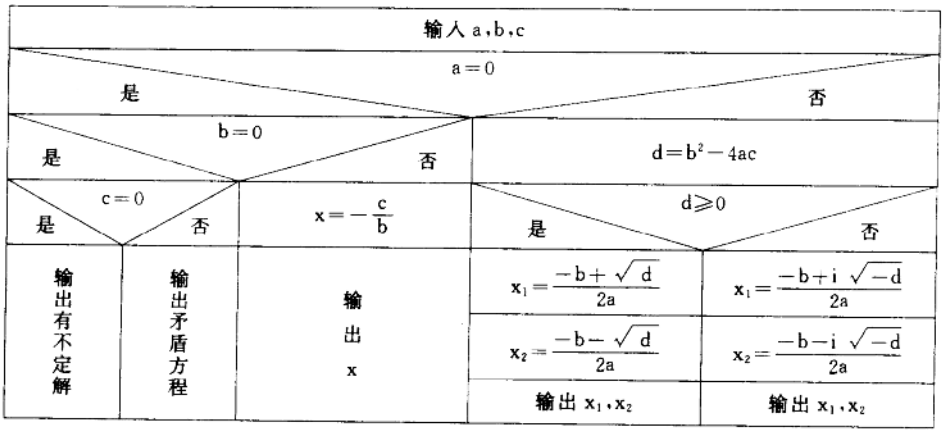


图 1.15

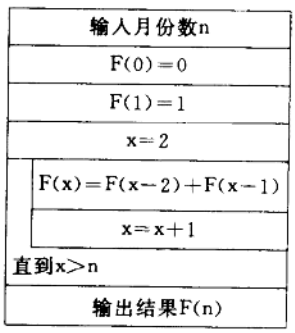


图 1.16

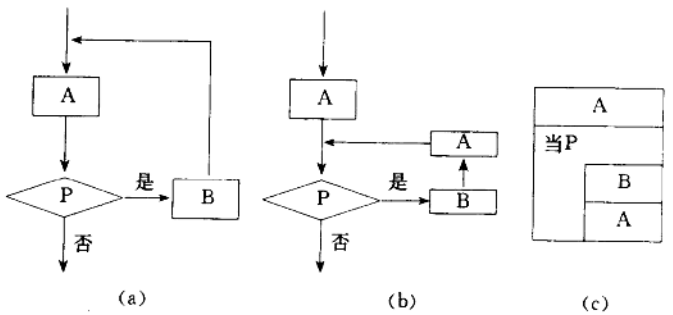


图 1.17

§1.3 结构化程序设计方法

结构化程序设计是60年代末至70年代中期形成的技术，它主要包括两个方面的内容，其一是对编写程序时使用控制结构的要求，强调使用三种基本控制结构，避免使用可能降低程序清晰性的转向语句(GO TO 语句)；其二是在软件开发的设计与实现过程中，提倡采用自顶向下和逐步细化的方法。

关于由三种基本结构组成结构化算法的内容在§1.2中已作了介绍，下面仅就“自顶向下，逐步细化”的方法给予简单介绍。

为了编写一个程序，有两种不同的思考方法，一种叫做“自顶向下”的方法，另一种叫做“自底向上”的方法。例如编著一部书时，往往先确定目标和主题，然后构思分成若干章，定出各章题目和大致内容，把每章分为若干节，再细分为若干段……即由粗到细，逐步具体化。这种方法是先有全局，进行整体设计后，再进行下一层的设计，逐步地实现精细化，这种方法称为“自顶向下，逐步细化”方法。日常生活中常使用这种方法。建筑师先设计鸟瞰图、平面图、立面图，在得到认可后再进行各部分的设计工作(由粗到细地逐步展开)，就是自顶向下的设计方法。

自底向上方法则与此相反，是先从具体局部入手，把若干个局部构成整体，建筑工人盖房

子从一砖一瓦入手，先完成局部后完成整体，就是一种“自底向上”的方法。

显然，“自顶向下”方法能做到胸有全局、通盘考虑，不致顾此失彼、头重脚轻，结构化程序设计提倡这种方法。过去不少程序设计人员采用的往往是“自底向上”的方法，拿到题目未作充分考虑就动笔一个语句一个语句地写程序，这种程序通常漏洞百出、质量较差，可读性低。

自顶向下，逐步细化，就是把一个复杂的任务分解成较小子任务，子任务还可能再分解为子子任务，整个复杂任务的完成就依赖于这些子任务、子子任务的逐个完成。这在程序设计工作中就是模块化程序设计的思想。

所谓模块化，是将一个大任务分成若干个较小的部分，每一部分承担一定的功能，称为“功能模块”，各个模块都可以分别由不同的人编写程序和调试，便于组织人力完成较复杂的任务。

自顶向下、逐步细化和模块化技术三者是紧密结合的，是结构化程序设计方法不可分割的三个要点，下面举一个简单的例子来说明如何实现结构化程序设计。

例 1 输入 100 个大于 2 的自然数，挑选出其中的素数打印出来。

图 1.18 是“顶层设计”，这是在最高层上的分解。

图 1.19 是将图 1.18 中的模块 A,B,C 分别细化后的结果。然后再将图 1.19 中的模块 D 和 E 细化，就得到图 1.20。

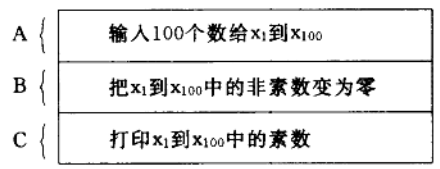


图 1.18

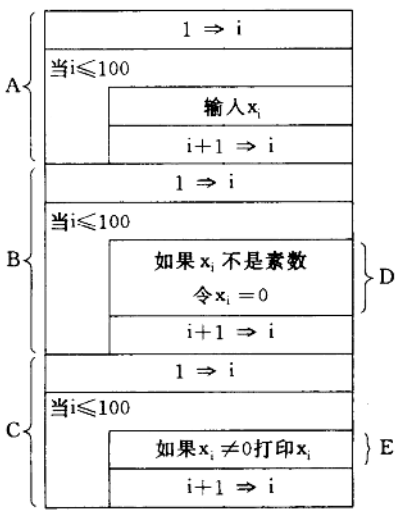


图 1.19

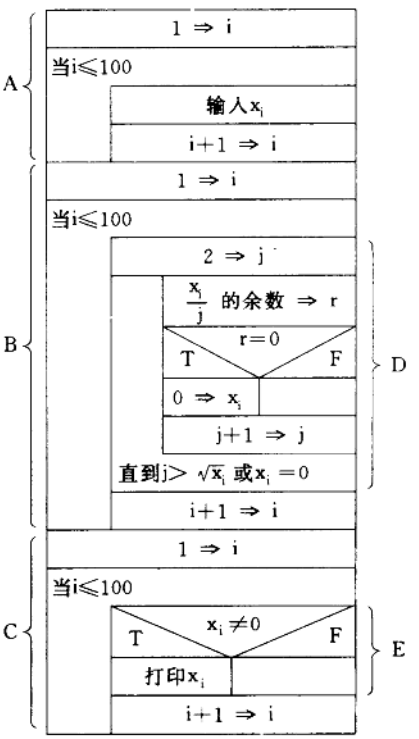


图 1.20

到此为止，已全部细化完毕。因为每一部分都能用高级语言的语句来表示。

如果遇到复杂的题目，在一张纸上画不完详细的 N-S 图，可以画出图 1.18 那样的粗框图，然后在另一张纸上画出某个框(A,B 或 C)的细化结果。看图时自顶向下一层一层地往下

找，直到了解整个算法。

从以上的例子可以看出自顶向下、逐步细化是结构化程序设计方法的核心，只用三种基本结构表示算法是保证设计出结构化程序的手段，我们应学会利用这种方法把一个总的任务逐步具体化，最终绘出结构化的框图。

§1.4 计算机解题过程

当我们准备用计算机来处理一个比较复杂的实际问题时，首先需要对这个问题进行分析，通过分析了解所开发的软件“做什么”的问题，即软件的功能、性能，还有技术可行性、社会可行性等，这也称为需求分析阶段。在分析的基础上制定出一个处理这个问题的完整的方案。这个方案必须是一个经过认真仔细考虑的、相当完整、相当合理的方案。这个方案中解决的是“怎样做”的问题。制定这个方案的过程也称为软件设计阶段，软件设计阶段解决的是软件的总体结构和一些处理的细节。通常把设计阶段的工作分为两步：概要设计和详细设计。

概要设计要完成程序结构的总体设计和数据结构的设计。程序结构的设计解决如何把整个任务划分成若干个部分的问题，这也是一个自顶向下、逐步细化的设计过程，设计出的程序结构最好是一个树状结构，如图 1.21 所示。图中的矩形代表实现某种特定功能的程序段，称为程序模块。树状结构中，有一个顶层模块，与其联系的有若干个下属模块，各下属模块可进一步引出更下一层的下属模块。例如报表加工的程序结构图如图 1.22 所示。从图中我们看出模块的层次关系十分清楚。

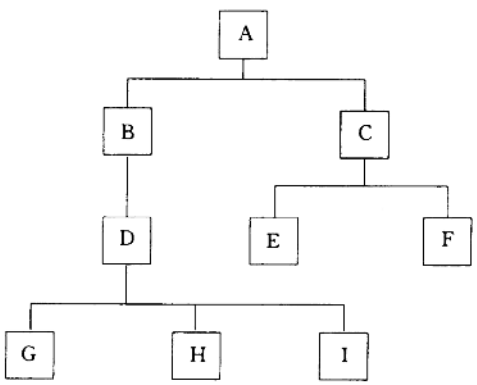


图 1.21

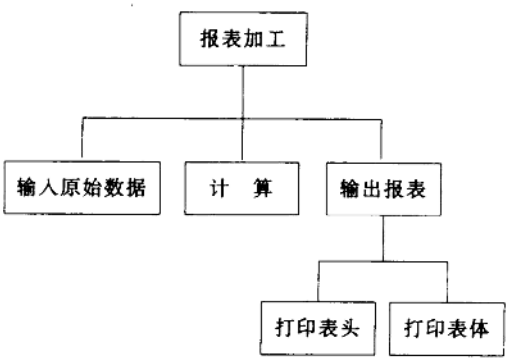


图 1.22

数据结构的设计是解决系统中用到的数据采用什么样的类型、什么样的组织形式以及数据的完整性、安全性的设计。

概要设计完成后进行详细设计，详细设计主要解决各个程序模块所用的算法以及各个模块的内部数据结构。确定算法以后，还要选定某种表达方法来描述各个算法。例如在一个工程计算的课题中，有一个模块是解一个 n 阶线性方程组，在详细设计时，首先要确定解方程组的算法，是用高斯主元消去法还是用塞德尔迭代法，这要根据实际问题的要求进行选择。选择算法后，可以用 N-S 图或程序流程图描述解方程组的过程。另外关于数据结构的确定，可以用一个 n 行 n+1 列的二维数组表示 n 阶方程组的系数增广矩阵。

把软件设计阶段的成果进一步转换，产生出某种程序设计语言的源程序，这一过程就是编码阶段。为了保证编码的质量，程序员必须深刻地理解、熟练地掌握并正确地运用程序设计语言的特性，做到编出的程序无语法错误和语义错误且节省存储空间、运行速度快。然而，对程序质量的要求，除了达到上述要求外，还要求源程序具有良好的结构性和良好的程序设计风格。只有这样编出的程序才能容易阅读，便于测试和排除程序差错。§1.3讲到的结构化程序设计方法是保证程序质量的有效方法。另外要想得到具有良好风格的程序，应主要解决好以下几个方面的问题：

(1) 符号名的命名采用具有实际意义的标识符，使其能够见名知意。例如表示和的量用sum，表示次数的量用times等。

(2) 在程序中尽量多地使用注释语句，让读程序的人更加容易理解程序。

(3) 恰当使用空格和空行，使编出的程序格式呈锯齿状(缩进格式)，层次清楚，段落分明。

编码阶段的成果就是源程序，源程序是否正确，需要进行测试。测试目的就是尽可能发现和消除程序中各种各样的错误。程序中常见的错误有语法错、名字符号错、初始化数据错、数据格式或文件格式错、循环次数错、逻辑性差错或纯属语义上的差错等等。语法错误在编译时就可以发现，某些逻辑性差错在连接成目的程序时可以发现，但大部分错误需要我们选择一批测试数据进行测试后才能发现。可以选择符合程序需要的典型的正确数据作为程序的输入，检验程序的输出是否是预期的结果。还可以选择错误的数据作为程序的输入，考验程序是否能发现错误并能进行相应的处理。注意，我们只能尽可能发现和排除程序中的绝大部分错误，保证一个复杂的程序百分之百的正确是不易做到的，所以测试工作要量力而行，适可而止。

经过测试我们得到了基本正确的程序，到此完成了软件开发的任务，下一步到了软件的运行和维护阶段。我们运行软件处理实际问题，在长期的使用中如发现程序中还存在着这样或那样的错误，就要及时给予改正；如果由于实际的需要，软件还应增加某些功能或对某些功能需要改进，就要对软件进行扩充和完善；如果由于外部的环境的改变，例如计算机硬件的改变或操作系统的版本变更，需要对程序作某些变化，这也要修改程序。以上所说的分别称为改正性、完善性和适应性维护工作。

以上我们介绍了用计算机解决一个实际问题的全部过程，也称为软件的生命周期，它包含了需求分析、软件设计、编码、测试、运行维护五个步骤。如果要详细了解这方面的知识请阅读软件工程方面的书籍。

§1.5 简单算法举例

我们以五个简单的典型算法为例，学习如何用N-S图描述算法。

例1 有两个杯子A和B，分别盛满酒和醋，要求将它们互换(即A杯原来盛酒，现改为盛醋，B杯则相反)。

根据常识，必增加一个空杯C作为过渡，算法如图1.23。

这个生活中的普通常识，应用于程序设计中，就是两个变量A和B的值的互换问题。需要引入一个工作变量C传递一下数据，描述算法的N-S图如图1.24。

例2 从10个数中挑选出最大的数。

解决这个问题的思路可用“打擂台”来比喻：先有一个人在台上(10个参赛者之一，不妨

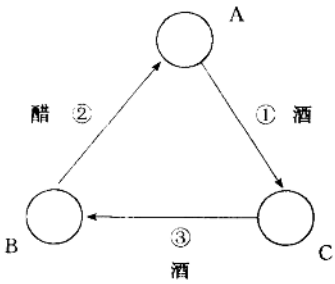


图 1.23

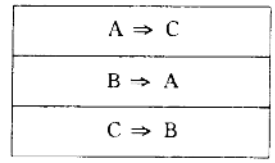


图 1.24

假设为第一个),然后第二个人与他比武,胜者留在台上,然后第三个与台上人比,胜者留在台上。如此继续比下去,直到第 10 个人比完为止(一共比 9 次),最后留在台上的为 10 个人中的胜者。

为描述算法,引入变量 A,设 A 为假设的最大数(在台上的人),第一次值为第一个数,参与比较的变量为 B(与台上人比武的人),若 B>A,则变量 A 的值变为变量 B 的值(冲掉原来 A 的值),即两两比较大者为 A,还需一个统计比较次数的变量 N,开始时值为 0,每比较一次做一次统计,即 N 的值在原来值的基础上加 1,直至比完 9 次(即 N≥9)为止,此时最大数在 A 中,最后将 A 的值输出。描述此算法的 N-S 图如图 1.25。

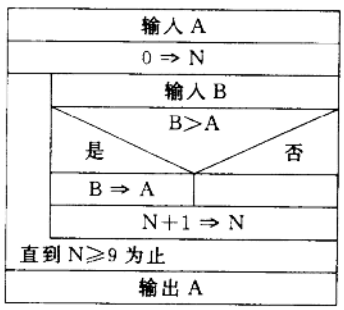


图 1.25

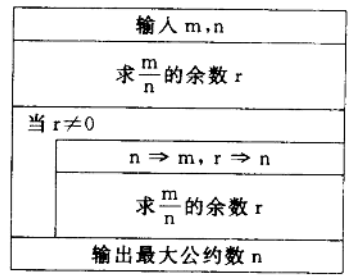


图 1.26

例 3 用辗转相除法求两个正整数 m 和 n 的最大公约数。

设 $m=120, n=28$,用辗转相除法求两个数的最大公约数的步骤如下:

(1) 用 120 作被除数,28 作除数(反之亦可),120 除以 28 的余数 $r_1=8$,由于余数 $r_1\neq 0$,做下一步。

(2) 原除数 28 作被除数,原余数 8 作除数,28 除以 8 的余数 $r_2=4$,由于余数不为 0,做下一步。

(3) 新除数 8 作被除数,新余数 4 作除数,8 除以 4 的余数 $r_3=0$,除数 4 就是 120 与 28 的最大公约数。

把此过程抽象出来:用 m 作被除数,n 作除数,其余数为 r,若 $r\neq 0$,则把除数 $n\Rightarrow m$,余数 $r\Rightarrow n$,重复以上过程,直至 $r=0$ 为止,除数 n 为原数 m 和 n 的最大公约数。

用 N-S 图描述上述算法,如图 1.26。

例 4 求 $n!=1\times 2\times 3\cdots\cdots\times n$。特别当 $n=5$ 时,即求 $1\times 2\times 3\times 4\times 5$ 的值。

描述算法的 N-S 图如图 1.27。

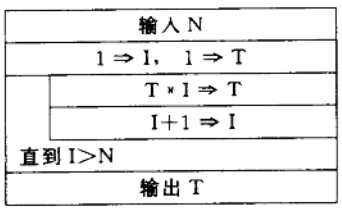


图 1.27

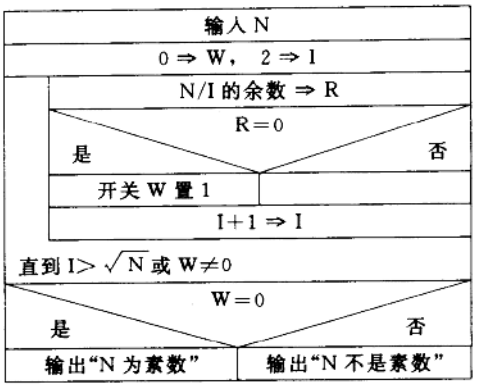


图 1.28

例 5 给定一个正整数 N，判断它是否为素数。

素数也叫质数，就是除去 1 和本身外，不能被其他任何整数整除的数。判断一个数 N 是否是素数，根据定义就是将 N 被 2,3,…,$(N-1)$去除，如果都除不尽，则 N 为素数，否则，若 2,3,…,$(N-1)$中有一个能整除，N 就不是素数。另外，根据数学知识，N 只需被 2,3,…,$\sqrt{N}$ 去除即可。描述算法的 N-S 图如图 1.28。

习 题 一

1. 叙述计算机语言大致经历了几代？
2. 程序设计语言和语言处理程序的作用是什么？
3. 编译程序和解释程序的工作方式有何不同？
4. 结构化程序设计的特点是什么？利用哪三种基本结构？用 N-S 图表示这三种基本结构。
5. 用 N-S 图描述下列各题的算法：

(1) 求 $\frac{1}{1\times2}+\frac{1}{2\times3}+\frac{1}{3\times4}+\frac{1}{n(n+1)}$，其中 $n=20$；

(2) 求 $s=\sum_{n=1}^{20} n!$。

试读结束：需要全本请在线购买： www.ertongbook.com