第一章

程序设计基础

本章重点介绍程序设计的基本理论、基本知识和基本方法，为读者今后更好地从事程序设计和软件开发打下良好的基础。本章首先介绍计算机的基本组成，其次对程序设计和程序设计语言的基本概念及发展作一概述，着重介绍结构化程序设计和面向对象程序设计。

1.1 计 算 机 基 础

计算机被人类称为是 20 世纪最伟大的发明之一。问世几十年来，已经经历了电子管计算机、晶体管计算机、集成电路(IC)计算机、大规模集成电路(LSI)计算机“智能”化计算机五个发展阶段。计算机的应用很广泛，涉及到国民经济和社会生活的各个领域、各行各业，主要用于科学与工程计算、数据处理与信息管理、工业生产自动化、计算机辅助系统、人工智能等方面。

计算机技术与通信技术相结合，出现了计算机网络通信，尤其是 Internet 的快速发展，使得世界各地的人们可以互相交流，缩短了彼此的距离，使我们足不出户，就可以了解各国的国情。同时，随着 Internet 三大应用：远程教学、远程医疗和电子商务的发展，将使我们的生活方式和生活环境发生很大的变化，尤其是电子商务的发展，很可能使我们可以在网上生存，从网上购得自己所需要的一切东西。当今社会已步入 21 世纪的信息时代，了解计算机，学会使用计算机是时代对我们的要求。

计算机系统实际上是一个很复杂的系统，主要由两大部分组成：硬件系统和软件系统。硬件系统是计算机的物质基础，软件系统是计算机得以运行的保障。

1.1.1 计算机硬件系统

计算机硬件系统实际上是由各种物理部件组成的，直观地看，计算机硬件系统就是一大堆物理设备。一台计算机从硬件系统看主要由四大部件组成：中央处理器、存储器、输入设备和输出设备，如图 1.1所示，图中实线表示数据线，虚线表示控制线。

1. 存储器

存储器是计算机用来存放程序和数据的记忆设备。根据存储器和中央处理器的关系，存储器可分为主存储器(简称主存,又称内存)和外存储器(简称外存,又称辅存)。

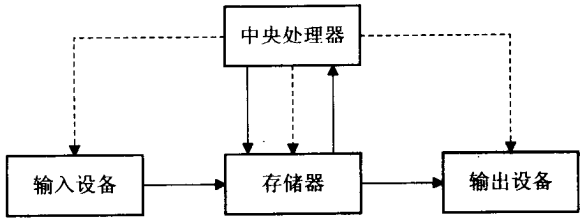

图 1.1 计算机的基本部件

主存储器通常设置在主机内部，它的基本功能是按指定的位置（地址）写入或读出（也称访问，access）信息。主存储器可以看成是由一系列存储单元组成的，每个存储单元都有一个特定的编号，即地址，地址指出了该单元在主存储器中的相对位置，便于中央处理器进行访问。每个存储单元可以存一个数据代码，该代码可以是指令，也可以是数据。计算机计算之前，程序和数据通过输入设备送入主存储器，计算开始后，主存储器不仅要为其它部件提供必需的信息，也要保存运算中间结果及最后结果，总之，它要和各个部件打交道，进行数据传送。主存储器是用半导体器件构造成的，其存取速度比较快，但由于价格上的原因，容量较小。

外存储器设置在主机外部，用来存放相对不经常使用的程序和数据。外存储器的容量一般较大，而且可以移动，便于计算机之间进行信息交换，价格较低，但存取速度较慢。常用的外存有磁盘、磁带和光盘，磁盘又分为软盘和硬盘。

2. 中央处理器

中央处理器简称 CPU(Central Processing Unit)，它是整个计算机的核心，计算机发生的所有动作都是受 CPU 控制的。CPU 主要包括运算器、控制器和寄存器三个部分。

运算器也称为算术逻辑部件 ALU(Arithmetic Logic Unit)，主要完成数据的算术运算(如加、减、乘、除)和逻辑运算(如与、或、非运算)。控制器负责从内存中读取各种指令，并对指令进行分析，根据指令的具体要求向计算机各部件发出相应的控制信息，协调它们的工作，从而完成对整个计算机系统的控制。因此控制器是计算机的指挥控制中心。CPU 内部的寄存器主要用来存放经常使用的数据。

3. 输入设备

输入设备的主要功能是把要输入的程序和数据信息通过输入接口顺序地送往计算机存储器中。常用的输入设备有键盘、鼠标、扫描仪、手写笔等。

4. 输出设备

输出设备的主要功能是将计算机操作的结果，转换为人们能识别和接收的信息形式，通过输出接口输送出来。常用的输出设备有显示器、打印机、绘图仪等。

磁盘既是输入设备，又是输出设备。输入/输出设备，又称为 I/O 设备。

1.1.2 计算机软件系统

仅有硬件系统的计算机（常称为裸机）是无法工作的，必须为它编制出由一条条指令组成的各种程序，它才能正常工作。计算机软件系统就是为了运行、管理和维护计算机而编

制的各种程序的总和。软件系统是计算机的灵魂，主要包括系统软件和应用软件。

1. 系统软件

系统软件是为了发挥计算机各部件的功能，方便用户使用而编制的。能实现系统功能的软件主要包括操作系统 OS(Operating System) 、数据库管理系统、各种语言的编译系统和解释系统以及各种工具软件等。

操作系统是对裸机的第一层扩充，是管理和控制计算机资源，如 CPU、I/O 设备及存储器等，使应用程序得以自动执行的程序。最常用的操作系统有 DOS、WINDOWS、UNIX 等。数据库管理系统是指用来帮助用户在计算机上建立、使用和管理信息系统的系统软件。各种语言的编译系统和解释系统是指程序设计语言的翻译系统。工具软件有时又称服务软件，是开发和研制各种软件的工具。常见的工具软件有诊断程序、调制程序、编辑程序等。

2. 应用软件

应用软件是用户利用计算机及其提供的系统软件，为解决实际问题而编制的计算机程序。由于计算机的日益普及，各行各业、各个领域的应用软件越来越多，常用的应用软件有以下几种：

(1) 信息管理软件，如 MIS 管理信息系统)；

(2)办公自动化系统,如 OFFICE 2000、WPS 2000；

(3) 辅助设计软件以及辅助教学软件，如 AUTOCAD；

(4)图像处理软件,如 PHOTOSHOP、CORELDRAW；

(5) 数值处理软件，如 MATHLAB。

用户可以根据需要解决的各种实际问题，选择不同的应用软件。

1.1.3 计算机的发展

真正作为世界上第一台计算机的是 1946 年美国研制成功的全自动电子数字计算机 ENIAC。这台计算机共用了 18 000 多个电子管，占地 170 m^2,总重量为 30 kg,耗电 140 kW,每秒能作 5000 次加减运算。这台计算机虽然有许多明显的不足之处，它的功能还不及现在的一台普通微型计算机，但它的诞生宣布了电子计算机时代的到来，其重要意义在于它奠定了计算机发展的基础，开辟了一个计算机科学技术的新纪元。

在短短的半个多世纪中，计算机的发展突飞猛进，经历了电子管、晶体管、集成电路和超大规模集成电路四个阶段，使计算机的体积越来越小，功能越来越强，价格越来越低，应用越来越广泛。按照电子元器件的发展阶段，可将计算机分为四代：

1. 第一代计算机

第一代计算机是指以第一台计算机 ENIAC 问世开始到 20 世纪 50 年代末这一阶段的计算机。这一时期的主要特征是使用电子管作为电子器件；软件还处于初始阶段；使用机器语言与符号语言编制程序。

第一代计算机是计算机发展的初级阶段，其体积比较大，运算速度也比较低，存储容量不大，并且，为了解决一个问题，所编制的程序很复杂。这一代计算机主要用于科学计算。

2. 第二代计算机

第二代计算机是指从 50 年代末到 60 年代初期的计算机，其中 1958 年与 1959 年是这一代计算机的鼎盛时期。这一时期的主要特征是使用晶体管作为电子器件，在软件方面开始使用计算机高级语言，为更多的人学习和使用计算机铺平了道路。

这一代计算机的体积大大减小，具有重量轻、寿命长、耗电少、运算速度快、存储容量比较大等优点。因此，这一代计算机不仅用于科学计算，还用于数据处理和事务处理，并逐渐用于工业控制。

3. 第三代计算机

第三代计算机是从 60 年代中期到 70 年代初期的计算机。这一时期的主要特征是使用中、小规模集成电路作为电子器件，并且，操作系统的出现，使计算机的功能越来越强，应用范围越来越广。

使用中、小规模集成电路制成的计算机，其体积得到了进一步的减小，功能得到了进一步的增强，可靠性和运算速度等指标也得到了进一步的提高，并且为计算机的小型化、微型化提供了良好的条件。这一时期中计算机不仅用于科学计算，还用于文字处理、企业管理、自动控制等领域，出现了计算机技术与通信技术相结合的信息管理系统，可用于生产管理、交通管理、情报检索等领域。另外，微型计算机得到了飞速的发展，对计算机的普及起到了决定性的作用。

4. 第四代计算机

第四代计算机是指用大规模与超大规模集成电路作为电子器件制成的计算机。这一代计算机在各种性能上都得到了大幅度的提高，对应的软件也越来越丰富，其应用已经涉及到国民经济的各个领域，并且在办公室自动化、数据库管理、图像识别、语音识别、专家系统等众多领域中大显身手，同时也已进入了家庭。

1.1.4 计算机的发展方向

计算机的广泛应用有力地推动了国民经济的发展和科学技术的进步，同时也对计算机技术提出了更高的要求，从而促进了计算机的进一步发展。以超大规模集成电路为基础，未来的计算机将向巨型化、微型化、网络化与智能化的方向发展。

1. 巨型化

巨型化并不是指计算机的体积大，而是指计算机的运算速度更高，存储量更大，功能更强。为了满足如天文、气象、宇航、核反应等科学技术发展的需要，也为了满足计算机能模拟人脑学习、推理等功能所必需的大量信息记忆的需要，必须发展超大型的计算机。目前正在研制的巨型计算机，其运算速度可达千亿次每秒，内存容量可达几百兆字节，而外存的容量将更大。这样的巨型计算机，其信息存储的能力可超过一般大型图书馆所需要的信息存储量。

2. 微型化

超大规模集成电路的出现，为计算机的微型化创造了有利条件。目前，微型计算机已进入仪器、仪表、家用电器等小型仪器设备中，同时也作为工业控制过程的心脏，使仪器

设备实现‘智能化”,从而使整个设备的体积大大缩小,重量大大减轻。自 20 世纪 70 年代微型计算机问世以来，大量小巧、灵便、物美价廉的个人计算机为应用的普及做出了巨大的贡献。随着微电子技术的进一步发展，个人计算机将发展得更加迅速，其中笔记本型、掌上型等微型计算机必将以更优的性能和更低的价格受到人们的欢迎。

3. 网络化

随着计算机应用的深入，特别是家用计算机越来越普及，一方面希望众多用户能共享信息资源，另一方面也希望计算机之间能互相传递信息进行通信。个人计算机的硬件和软件配置一般都比较低，其功能也有限，因此，要求大型与巨型计算机的硬件和软件资源以及它们所管理的信息资源应该为众多的微型计算机所共享，以便充分利用这些资源。基于这些原因，促使计算机向网络化发展，将分散的计算机连接成网，组成计算机网络。在计算机网络中，通过网络服务器，一台台计算机就像人类社会的一个个神经单元被联系起来，从而组成信息社会的一个重要的神经系统。

计算机网络是现代通信技术与计算机技术相结合的产物。所谓计算机网络，就是把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件、软件、数据信息等资源。计算机网络技术是在 20世纪 60年代末、70年代初开始发展起来的，由于它符合社会发展的趋势，因此其发展的速度很快。目前，已经出现了许多局部网络产品，应用也比较普遍，尤其是在现代企业的管理中网络发挥着越来越重要的作用。实际上，像银行系统、商业系统、交通运输系统等单位，要真正实现自动化，具有快速反应能力，都离不开信息传输，离不开计算机网络。

随着社会及科学技术的发展，对计算机网络的发展提出了更高的要求，同时也为其发展提供了更加有利的条件。计算机网络与通信网的结合，可以使众多的个人计算机不仅能够同时处理文字、数据、图像、声音等信息，而且可以使这些信息四通八达，及时地与全国乃至全世界的信息进行交换。

4. 智能化

最初,计算机主要用于计算。但是,现代计算机早已突破了‘计算’这一初级含义。

计算机人工智能的研究是建立在现代科学基础之上的。计算机智能化程度越高，就越能代替人的作用。因此，智能化是计算机发展的一个重要方向。现在正在研制的新一代计算机，要求它能模拟人的感觉行为和思维过程的机理，使计算机不仅能够根据人的指挥进行工作,而且可以‘看’、‘听’、‘说’、‘想’、‘做’,具有逻辑推理、学习与证明的能力。这样的新一代计算机是智能型的，甚至是超智能型的，它具有主动性，具有人的部分功能，不仅可以代替人进行一般工作，还能代替人的部分脑力劳动。

现在，世界上许多国家都在积极开展智能型计算机的研制开发工作，这是人类对计算机技术的一种挑战，也是对其它有关领域和学科发起的挑战，它必将促进其它众多学科的进一步发展。

1.2 程序设计基础

当我们需要计算机来解决一个实际问题时，必须依靠一定的工具（人机界面）编写一个指令（语句）清单，一旦指令（语句）清单提交给计算机，计算机就可以执行这些指令（语句），产生结果。程序就是计算机可以执行的指令序列或语句序列。而设计、编制、调试程序的过程就称之为程序设计。编写程序所使用的语言即为程序设计语言，它为程序设计提供了一定的语法和语义，所编写出的程序必须严格遵守它的语法规则，这样编写出的程序才能被计算机所接受，所运行，才能产生预期结果。

1.2.1 程序及算法

一个程序应包括以下两方面的内容。

(1) 对数据的描述。在程序中要指定数据的类型和数据的组织形式，即数据结构（data structure）

(2) 对数据操作的描述，即操作步骤，也就是算法(algorithm)。

数据是操作的对象，操作的目的是对数据进行加工处理，以得到期望的结果。打个比方，厨师做菜肴，需要有菜谱。菜谱上一般应包括：①配料，指出应使用哪些原料；②操作步骤，指出如何使用这些原料按规定的步骤加工成所需的菜肴。面对同一些原料可以加工出不同风味的菜肴。作为程序设计人员，必须认真考虑和设计数据结构及操作步骤（即算法)。因此,著名计算机科学家沃思(Nikiklaus Wirth）提出一个公式：

数据结构+ 算法 =程序

实际上，一个程序除了以上两大要素之外，还应当采用结构化程序设计方法进行程序设计，并且用某一种计算机语言表示。因此，程序还可以这样表示：

程序 = 算法+ 数据结构+ 程序设计方法+ 语言工具和环境

也就是说，以上四个方面是一个程序设计人员所应具备的知识。在设计一个程序时要综合运用这几方面的知识。在这四个方面中，算法是灵魂，数据结构是加工对象，语言是工具,编程需要采用合适的方法。算法是解决‘做什么’和‘怎么做’的问题。程序中的操作语句，实际上就是算法的体现。显然，不了解算法就谈不上程序设计。下面用例子来引入算法的概念。

例 1.1 给定两个正整数 p 和 q，求 p 和 q 的最大公约数 g。

如何求得两个正整数的最大公约数呢？对于这样一个数学问题，数学家欧几里德(Euclid)给出了一个解决方案，这个方案由三个步骤完成，如下所述。

求解最大公约数的欧几里德算法：

步骤 1：如果 p < q，则交换 p 和 q。

步骤 2：令 r 是 p/q 的余数。

步骤 3：如果 r = 0，则令 g = q，结束算法，g 即为求得的最大公约数；否则令 p = q，q = r，转向步骤 2。

按照以上的步骤，就可以逐步地计算出任意两个正整数的最大公约数。例如求 24 和 16 的最大公约数，即 p = 24，q = 16。从步骤 1 开始执行，显然 p > q，不满足条件 p <

q，因此不用交换 p和 q 执行步骤 2，24/16 的余数是 8，即 r = 8 执行步骤 3，r≠0 ，则 p = 16，q = 8，转向步骤 2，16/8 的余数是 0，即 r = 0 接着执行步骤 3，因为 r = 0，则 g = 8，算法结束。最后算得的 g = 8 即为 24 和 16 的最大公约数。计算工具可以使用笔和纸，也可以使用计算机。

我们把这种将问题归结为有规律的操作步骤，并且用有限多个步骤来表示的具体过程就称之为算法。对同一个问题，可以有不同的解题方法和步骤。

例 1.2 求 1+2+3+…+100=?

算法 1

步骤 1：1+2 = 3

步骤 2：3+3 = 6

步骤 3：6+4 = 10

步骤 99：4950+100 = 5050

算法 2

100+(1+99)+(2+98)+ +(49+51)+ 50

= 100+49 *100 +50

= 5050

算法 3

步骤 1：k = 1，s = 0

步骤 2：如果 k>100，则算法结束，s 即为求得的和；否则转向步骤 3

步骤 3：s = s+k，k = k+1

步骤 4 转向步骤 2

对于这个问题，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。一般地说，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。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。用计算机实现算法，就是用计算机语言编写程序去执行。

1.2.2 算法的特征和描述

一、算法的特征

由上面的例子可以看出一个算法具有如下五个特点。

1. 有穷性

一个算法必须总是（对任何合法的输入值）在执行有限步骤之后结束，即一个算法必须包含有限的操作步骤，而不能是无限的，并且每一个步骤都必须在可接受的有穷时间内完成。

如在例 1.1 中，p 和 q 取任意的正整数，经过有限的步骤，总可以使 r = 0，算法都可以在执行有限的步骤之后结束。

2. 确定性

算法中的每一个步骤的含义都必须具有确定的意义，而不应当是含糊的，使读者在理解时产生二义性，并且，在任何条件下，算法只有惟一的一条执行路径，即对于相同的输入只能得出相同的输出。

3. 可行性

每一个算法是可行的，即算法中的每一个步骤都可以有效地执行，并得到确定的结果。因此，算法的可行性也称为算法的有效性。

例如，若 b = 0，则执行 a/b 是不可行的，即不能有效执行。

4. 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。一个算法可以有零个或多个输入。如在例 1.1 中，需要输入两个正整数的值，然后根据算法进行计算。一个算法也可以没有输入，如例 1.2，在执行算法时不需要输入任何信息。

5. 输出

算法的目的是为了求解，“解”就是输出。一个算法可以有一个或多个输出。没有输出的算法是没有意义的。

如在例 1.1 中，算法输出的结果 g 就是要求的 p 和 q 的最大公约数。

二、算法的描述

为了描述一个算法，可以用不同的方法。常用的有自然语言描述法、N-S 图描述法、伪代码描述法三种。

1. 自然语言描述法

所谓自然语言，就是人们日常使用的语言，可以是汉语、英语或其他语言。如例 1.1 就是用自然语言来描述求解最大公约数的算法。用自然语言来描述和表示算法通俗易懂，但文字过于冗长，容易出现歧义性。因此，除了简单的问题外，一般不用自然语言描述算法。

2. 伪代码描述法

伪代码描述法就是用介于自然语言和计算机程序设计语言之间的文字和符号来描述算法，即计算机程序设计语言中具有的关键字用英文表示，其他的可用汉字，以便于书写和阅读。用伪代码表示算法，并无固定的、严格的语法规则，只要求把意思表达清楚，但书写的格式要写成清晰易懂的形式。

3. N-S 流程图描述法

N-S 流程图是在 1973 年，由美国学者 I. Nassi 和 B. Shneiderman 提出的一种新的流程图形式。在这种流程图中，将全部算法写在一个矩形框内，在该矩形框内还可以包含其他的从属于它的框，或者说，由一些基本的框组成一个大的框。这种流程图被称为 N-S 结构化流程图（N 和 S 是两位美国学者的英文姓名的第一个字母）。这种流程图适合于结构化程序设计，因而很受欢迎。

N－S流程图用以下的流程图符号：

(1)顺序结构：用图 1.2 形式表示。A和 B两个框组成一个顺序结构，表示先执行 A操作，然后执行 B操作。

(2)选择结构：用图 1.3表示。当 P条件成立时，执行 A操作；P不成立则执行 B操作。注意图 1.3是一个整体，代表一个基本结构。

(3) 循环结构：有两种，当型循环结构用图 1.4形式表示。直到型循环结构用图 1.5形式表示。

图 1.4表示当给定的 P条件成立时，反复执行 A操作，直到 P条件不成立为止，跳出循环。

图 1.5表示先执行 A操作，然后判断 P条件是否成立，如果 P条件不成立，则再执行 A，再判断 P，直到给定的 P条件成立为止，然后跳出循环。

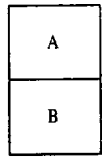

图 1.2

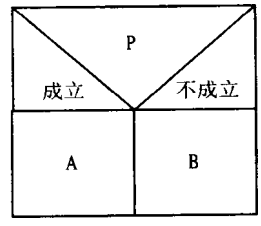

图 1.3

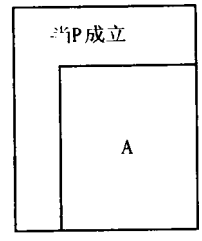

图 1.4

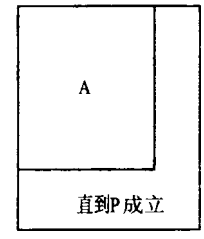

图 1.5

用以上三种 N－S流程图中的基本框，可以组成复杂的 N－S流程图，以表示算法。

需要说明的是，在图 1.2、图 1.3、图 1.4、图 1.5中的 A框或 B框，可以是一个简单的操作(如读人数据或打印输出等)，也可以是三个基本结构之一。例如，图 1.2所示的顺序结构，其中的 A框可以是一个选择结构，B框可以是一个循环结构，即如图 1.6所示。由 A框和 B框这两个基本结构组成一个顺序结构。

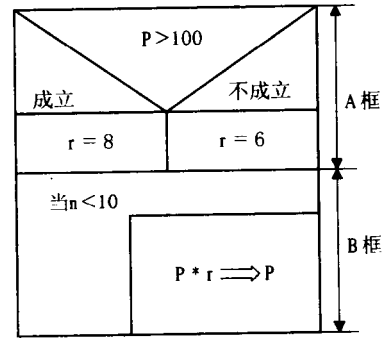

图 1.6

用 N－S流程图表示算法直观易懂，但画起来比较费事，在设计一个算法时，可能要反复修改，而修改流程图是比较麻烦的。因此，流程图只适宜于表示一个算法，在设计算法的过程中，其不是很理想的方法。

以上三种算法的表示方法各有长短。自然语言只适用于表示简单的算法，用伪代码表示，书写格式比较自由，可以随手写下去，容易表达出设计者的思想。同时，用伪代码写的算法很容易修改，便于算法的设计。而使用 N－S流程图则便于算法的表示。在程序设计过程中，可以根据需要和习惯任意选用。

1.2.3 算法与程序设计

现实生活中的许多简单问题也可以用算法来表示，常常将这些简单问题归结为三种情况：顺序、选择、循环。

1. 关于顺序

例如，客户到银行去取款，首先需要填写取款单，然后等待业务员处理，输入密码，拿到取款。而对于业务员来说，要经过以下几个步骤：

步骤 1：输入客户帐号；

步骤 2：提示客户输入密码；

步骤 3：输入取款金额；

步骤 4：等待系统处理，输出取款金额和新余额；

步骤 5：经核实后，将取款金额交于客户，结束本次操作。

这种算法的步骤是顺序地一步一步进行的，不需要跳转，也不需要判断条件。该算法是用自然语言来描述的。

2. 关于选择

例如，要判断某一年是否是闰年，需要考虑两个条件： ① 能被 4整除，但不能被 100整除的年份都是闰年，如 1992年、2004年是闰年，而 1900年、2100年就不是闰年；②能被 100整除，又能被 400整除的年份是闰年，如 2000年是闰年。不符合这两个条件的年份不是闰年。算法可用自然语言描述如下：

步骤 1：输入需要判断的年份；

步骤 2：判断，若该年份不能被 4整除，则输出“不是闰年”，然后转到步骤 5，结束；

步骤 3：判断，若该年份能被 4整除，但不能被 100整除，则输出“是闰年”，然后转到步骤 5，结束；

步骤 4：判断，若该年份能被 100整除，又能被 400整除，则输出“是闰年”，否则输出“不是闰年”；

步骤 5：结束本次判断。

这种算法需要判断条件，根据不同的条件，产生分支，分别去处理不同的需求。下面分别用伪代码和 N－S 流程图描述本算法。

用伪代码描述如下：

```
开始
输入一个需要判断的年份 year;
if(year不能被 4整除)
    输出“不是闰年”;
else if(year不能被 100 整除)
        输出“是闰年”;
    else if(year 能被 400整除)
            输出“是闰年”;
        else 输出“不是闰年”;
结束
```

用 N－S流程图描述如图 1.7 所示。

3. 关于循环

例如，一个工人师傅要单独作 100 个零件，那么相同的工作就要作 100 次。又如，求和 1＋2＋3＋…＋99＋100，算法用自然语言表示如下：

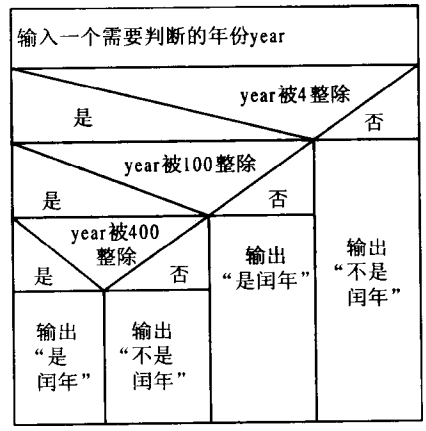

图 1.7

步骤 1 初始化和 sum ＝ 0；

步骤 2：令 i＝1；

步骤 3：计算 sum＋i，和仍放在 sum 中，可表示为 sum＋i⇒sum；

步骤 4：使 i 的值加 1，即 i＋1⇒i；

步骤 5：如果 i 不大于 100，返回重新执行步骤 3，以及其后的步骤 4 和步骤 5；否则，执行步骤 6；

步骤 6 输出结果 sum。

下面分别用伪代码和 N－S流程图的方法来描述本算法：

(1) 用伪代码描述如下：

```
开始
令 sum ＝ 0;
for(i ＝ 1; i<＝100; i＋＋)
    sum ＋ i⇒sum ;
输出结果 sum ;
结束
```

(2) 用 N－S流程图描述如图 1.8 所示。

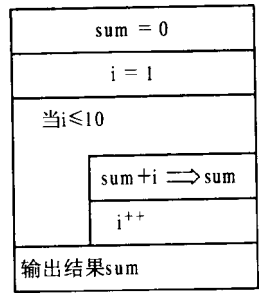

图 1.8

实际生活中的大部分问题是以上三种情况的综合。

例 1.3 判断，并输出 3 到 100 之间的素数。(所谓素数，是指除了 1和它本身之外，不能被其他任何整数整除的数，即数学意义上的质数，如 5、13、47 等。)

分析：判断一个数 n(3≤n≤100)是否为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。实际上，n 不必被 2 到(n－1)的所有整数除，只需被 2 到 n/2 之间的整数除即可，甚至只需被 2 到 $\sqrt{n}$ 之间的整数除即可。例如，判断 23 是否为素数，只需将 23 被 2、3、4 除即可。如果都除不尽，n 必为素数，这是一个数学定理。算法用自然语言表示如下：

步骤 1：令 n＝3；

步骤 2：使 m＝2；(m 作为除数)

步骤 3：n 被 m 除，得余数 r；

步骤 4：如果 r＝0，表示 n 能被 m 整除，则不是素数，执行步骤 7，否则执行步骤 5；

步骤 5：m＋1⇒m ；

步骤 6：如果 m≤$\sqrt{n}$，返回步骤 3 ；

步骤 7 如果 $m \leqslant \sqrt{n}$，则表示不是素数，否则表示从 2 到 $\sqrt{n}$，都不能整除 n，即 n 为素数，则输出 n；

步骤 8：n+1⇒n；

步骤 9 如果 n 不大于 100，返回重新执行步骤 2 以及其后的步骤，否则算法结束。最后输出的结果就是 3 到 100 之间的素数。

这个例子是一个相对比较复杂的问题，需要用两层循环来实现，其中也包括一些条件判断。下面分别采用伪代码和 N－S 流程图的方法来描述本算法。

(1) 用伪代码描述如下：

```
开始
for(n=3; n<=100; n++)
    { for(m=2; m<=√n ; m++)
      { n/m 的余数⇒r ;
        if(r == 0) break;
        }
      if(m>=√n + 1) 输出 n ;
    }
结束
```

(2) 用 N－S 流程图描述如图 1.9 所示。

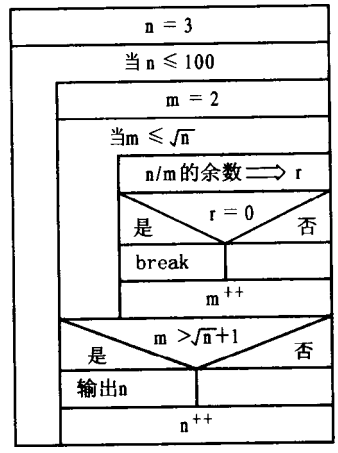

图 1.9

在计算机中算法是用程序设计语言进行描述的，算法的描述过程就是程序设计，如上面的算法都是用 C 语言来描述的。实际上，当算法确定了以后，有很多种语言可以选择，如 Pascal 语言和 Basic 语言。表 1.1 列举了三种语言的比较。

表 1.1　C 语言、Pascal 语言和 Basic 语言的比较

语言 / 语句	C 语言	Pascal 语言	Basic 语言	含　义
说明语句	{ }	BEGIN END	无	复合语句
	int x;	VAR X: INTEGER	DIM X	定义一个整型变量
	int y[5];	VAR Y: ARRAY[0..4] OF INTEGER	DIM X(5)	定义一个整型数组
顺序语句	x+=3;	x=x+3	X=x+3	赋值语句
	i++;	i=i+1	I=i+1	自增语句
	scanf( )	read	input	输入语句
	printf( )	write	output	输出语句
分支语句	if (条件) 语句 1 else 语句 2	IF 条件 THEN 语句	IF 条件 THEN 语句	条件语句

续表

语言 / 语句	C 语言	Pascal 语言	Basic 语言	含　义
循环语句	for(i=1;i<=20;i++) 语句	FOR I:=1 TO 20 DO 语句	FOR I=1 TO 20 STEP 1 语句序列 NEXT I	循环语句
	while(条件) 语句	WHILE 条件 DO 语句		
	do 语句 while(条件)	REPEAT 语句序列 WHILE 条件		
函数	int f( )	FUNCTION f( ): INTEGER	SUB F	定义返回值为整型的函数
指针	int *p	VAR P: ^ INTEGER	无	定义一个整型指针

对于现实生活中比较大的问题，是分模块进行处理的，每一个模块完成一定的功能。模块在不同的语言中，用不同的方法来实现， C 语言中用函数来实现，Pascal 语言中用函数和过程来实现，而 Basic 语言中用子程序来实现。

从这三种语言的比较可以看出，各种语言在本质上都是相同的，都为相应的功能提供相应的语句形式。在编写程序时，所采用的方法和思路也是相同的，不同的只是所使用语言的语法和语句形式的差异。因此，对于程序设计语言的学习是相通的，熟悉了一种语言，就很容易移植到另外一种语言上来。

1.2.4　程序设计语言

在计算机问世的几十年中，随着计算机硬件的更新换代，计算机程序设计语言也随之有很大发展，至今已经历了三代。

1. 第一代——机器语言

由于计算机只能接收以二进制形式，即有 0 和 1 组成的代码，因此早期的程序设计语言就是要写出由一条条这样的机器指令组成的程序。一条机器指令用来控制计算机执行一个操作内容，它告诉计算机应该进行什么运算，哪些数参加运算，这些数放在什么地方，结果应送到什么地方，等等。

由于这种程序计算机可以直接识别，执行速度快，但这种程序编写却很困难，易出错，难记，难检查，难维护，因此只适于专业人员使用，让所有用户掌握几乎是不可能的。

2. 第二代 ——汇编语言

汇编语言是一种面向机器的低级程序设计语言，是一种符号语言，语句中用助记符代替操作码和地址码。操作码表示执行何种操作，地址码指示操作的对象。语句基本上与机器指令一一对应。这样，用汇编语言写程序较之机器语言易于阅读和修改，且编写出的程序短，执行速度快，所以时至今日，汇编语言仍在使用，特别是用在实时控制系统中。但由

于它是一种面向机器的语言，不同机器所配的语言各不相同，不能通用。

3. 第三代——高级语言

高级语言不依赖于具体的机器，有严格的语法规则，接近于人们习惯使用的数学用语和自然语言。高级语言的出现，给计算机应用开辟了广阔的道路。由于高级语言易学易用，因此迅速得以推广和使用；由于高级语言独立于机器，编程者可以不了解具体计算机的细节情况，就可以编出像样的程序来；另外，其良好的可读性也为程序的修改和调试带来了很大的好处。但由于高级语言的运行需要编译、链接，因此程序的执行速度较低级语言要慢。当今世界上的高级语言多种多样，有上百种之多，本书主要以 C语言作为典型的程序设计语言来介绍程序设计的思想。

1.2.5 C语言

C 语言是国际上广泛流行的、很有发展前途的计算机高级语言。它适合于作为系统描述语言，即用来写系统软件，也可用来写应用软件。

以前的操作系统等系统软件主要是用汇编语言编写的（包括 UNIX 操作系统在内）。由于汇编语言依赖计算机硬件，程序的可读性和移植性都比较差。为了提高可读性和移植性，最好改用高级语言，但一般高级语言难以实现汇编语言的某些功能（汇编语言可以直接对硬件进行操作，例如，对内存地址的操作、位操作等）。人们设想能否找到一种既具有一般高级语言特性，又具有低级语言特性的语言，集它们的优点于一身。于是，C 语言就在这种情况下应运而生了。

C 语言是在 B 语言的基础上发展起来的，它的根源可以追溯到 ALGOL 60。1960 年出现的 ALGOL 60 是一种面向问题的高级语言，它离硬件比较远，不宜用来编写系统程序。1963 年英国的剑桥大学推出了 CPL(Combined Programming Language)语言。CPL 语言在 ALGOL 60 的基础上接近硬件一些，但规模比较大，难以实现。1967 年英国剑桥大学的 Matin Richards 对 CPL 语言作了简化，推出了 BCPL(Basic Combined Programming Language)语言。1970 年美国贝尔实验室的 Ken Thompson 以 BCPL 语言为基础，又作了进一步简化，设计出了很简单的而且很接近硬件的 B 语言(取 BCPL 的第一个字母),并用 B 语言编写了第一个 UNIX操作系统，在 PDP－7上实现。1971 年在 PDP－11/12 上实现了 B 语言,并编写了 UNIX 操作系统。但 B 语言过于简单，功能有限。1972 年至 1973 年间，贝尔实验室的 D. M. Ritchie 在 B 语言的基础上设计出了 C 语言(取 BCPL 的第二个字母)。C 语言既保持了 BCPL 和 B语言的优点(精练,接近硬件),又克服了它们的缺点(过于简单，数据无类型等)。最初的 C 语言只是为描述和实现 UNIX 操作系统提供一种工作语言而设计的。1973 年，K. Thompson 和 D. M. Ritchie 两人合作把 UNIX 的 90%以上用 C 语言改写(即 UNIX 第 5版。原来的 UNIX 操作系统是 1969 年由美国贝尔实验室的 K. Thompson 和 D. M. Ritchie 开发成功的，是用汇编语言编写的）。

后来，C 语言多次作了改进，但主要还是在贝尔实验室内部使用。直到 1975 年 UNIX 第 6 版公布后，C 语言的突出优点才引起人们普遍注意。1977 年出现了不依赖于具体机器的 C语言编译文本《可移植 C 语言编译程序》,使 C 移植到其它机器时所做的工作大大简化，这也推动了 UNIX 操作系统迅速地在各种机器上实现。例如，VAX，AT&T 等计算机系统都相继开发了 UNIX。随着 UNIX 的日益广泛使用，C 语言也迅速得到推广。C语言和

UNIX 可以说是一对孪生兄弟，在发展过程中相辅相成。 1978 年以后，C 语言已先后移植到大、中、小、微型机上，已独立于 UNIX 和 PDP 了。现在 C 语言已风靡全世界，成为世界上应用最广泛的几种计算机语言之一。

以 1978 年发表的 UNIX 第 7 版的 C 编译程序为基础，Brain W. Kernighan 和 Dennis M. Ritchie（合称 K & R）合著了影响深远的名著《The C Programming Language》，这本书中介绍的 C 语言成为后来广泛使用的 C 语言版本的基础，它被称为标准 C。1983 年，美国国家标准化协会（ANSI）根据 C 语言问世以来的各种版本对 C 的发展和扩充，制定了新的标准，称为 ANSI C。ANSI C 比原来的标准 C 有了很大的发展。K & R 在 1988 年修改了他们的经典著作《The C Programming Language》，按照 ANSI C 标准重新改写了该书。1987 年，ANSI 又公布了新标准——87 ANSI C 。目前流行的 C 编译系统都是以它为基础的。

目前广泛流行的各种版本 C 语言编译系统虽然基本部分是相同的，但也有一些不同。在微机上使用的有 Microsoft C，Turbo C，Quick C 等，它们的不同版本又有差异。在使用一个系统之前，要了解所用计算机系统的 C 编译的特点和规定。

1.3 程序设计发展史

程序设计初期，由于计算机硬件条件的限制，运算速度与存储空间都迫使程序员追求高效率，编写程序成为一种技巧与艺术，而程序的可理解性、可扩充性等因素被放到第二位。随着计算机硬件与通信技术的发展，计算机应用领域越来越广泛，应用规模也越来越大，程序设计不再是一两个程序员可以完成的任务。在这种情况下，编写程序不再片面追求高效率，而是综合考虑程序的可靠性、可扩充性、可重用性和可理解性等因素。正是这种需求刺激了程序设计方法与程序设计语言的发展。

1. 早期程序设计

早期出现的高级程序设计语言有 FORTRAN、COBOL、ALGOL、BASIC 等语言。FORTRAN 是 FORmula TRANslator 的缩写，是最早广泛使用的高级语言之一，主要应用在科学计算领域；COBOL 是 COmmon Business Oriented Language 的缩写，主要应用在商业事务处理领域，其中的许多指令是为了会计或工资一类应用而设计的； ALGOL 是 ALGOrithmic Language 的缩写，是一种通用的算法语言； BASIC 是 Beginner's All - purpose Symbolic Instruction Code 的缩写，主要面向初学者，BASIC 语言简单易学，但不是很先进。

这一时期，由于追求程序的高效率，程序员过分依赖技巧与天分，不太注重所编写程序的结构，这时期可以说是无固定程序设计方法的时期。一个典型问题是程序中的控制随意跳转，即不加限制地使用 goto 语句，这样的程序对别人来说是难以理解的，程序员自己也难以修改程序。

2. 结构化程序设计

随着程序规模与复杂性的不断增长，人们也在不断探索新的程序设计方法。 Bohm 和 Jacopini 证明了只用三种基本的控制结构（顺序、选择、循环）即可实现任何单入口 /单出口的程序；Dijkstra 建议从一切高级语言中取消 goto 语句的大辩论；MillS 提出程序应该只有一个入口和一个出口。这些工作导致了结构化程序设计方法的诞生。

Pascal 语言是由 Nikiklaus Wirth 根据结构化程序设计方法开发出来的语言，它是因纪念 17 世纪法国数学家 Blaise Pascal 而命名的。其特点是提炼出程序设计共同的特征并能将这些特征编译成高效的代码，因而成为结构化程序设计的有力工具，在教育界深受欢迎。

C 语言也是一种广为流行的结构化程序设计语言，它具有灵活方便、目标代码效率高、可移植性好等优点。

由美国国防部支持开发的 Ada 语言是结构化程序设计的总结，它是为了纪念第一位程序员 Ada Augusta 而命名的。Ada 语言包括了许多新的机制，如模块化、数据抽象、类属参数、异常处理、并发处理等。

3. 面向对象程序设计

到今天，结构化程序设计已无处不在，几乎每种程序设计语言都具备支持结构化程序设计的机制。然而，随着程序规模与复杂性的增长，程序中的数据结构变得与这些数据上的操作同样重要。在大型结构化程序中，一个数据结构可能被许多过程处理，修改此数据结构将影响到所有这些过程。在由几百个过程组成的成千上万行结构化程序中，这相当麻烦，并且容易产生错误。

面向对象程序设计建立在结构化程序设计的基础上，最重要的改变是程序围绕被操作的数据来设计，而不是围绕操作本身。面向对象程序设计以类作为构造程序的基本单位，具有封装、数据抽象、继承、多态性等特点。

Simula 语言是面向对象程序设计的先驱，它首先提出类的概念； Smalltalk 语言及程序设计环境的成功导致了面向对象程序设计的兴起； Eiffel 语言是完全根据面向对象程序设计方法开发出来的纯面向对象语言，具有强类型、多重继承、类属机制、断言机制、异常处理、自动内存管理等特点，进一步完善了面向对象程序设计。

C++语言则是目前应用最广泛的面向对象程序设计语言。 C++语言是对 C 语言的扩充(或称为 C 语言的超集),它继承了 C 语言高效、灵活的特点，完善了 C 语言的类型检查、代码重用、数据抽象机制，扩充了面向对象程序设计的支持。

1.4 结构化程序设计

1.4.1 结构化程序设计的发展

结构化程序设计的思想是在 20 世纪 60 年代末、70 年代初为解决‘软件危机’的需要而形成的，这一思想已被广泛接受。多年来的实践证明，结构化程序设计策略确实使程序执行效率提高，并且由于减少了程序的出错率，而大大减少了维护费用。正因为如此，结构化程序设计这一名词的应用比比皆是，结构化 FORTRAN、结构化 COBLE 等屡见不鲜。

结构化程序设计是按照一定的原则与原理，组织和编写正确且易读的程序的软件技术。结构化程序设计的目标在于使程序具有一个合理结构，以保证和验证程序的正确性，从而开发出正确、合理的程序。

结构化程序设计的提出对计算机科学产生了巨大影响。这里列举几点：

1. 对程序设计语言的影响

Pascal 语言是基于结构化程序设计思想而设计的，并且是系统体现这一思想的第一个

程序设计语言。这是程序设计语言发展的一个里程碑。结构化程序设计思想也深深地影响到其后的各种语言之设计。至于早先存在的语言则纷纷研制它们的结构化版本，形成所谓的结构化 FORTRAN 等。C 语言也是一个比较典型的结构化的程序设计语言。

2. 推动了程序设计方法学的发展

Software－Practice and Experience 杂志 1975 年第 1 期社论把结构化程序设计看作是在程序实践所依据的概念中的一场革命。结构化程序设计的提出让人们看到，与其它很多科学一样，程序设计也有自身的规律与原理，从而产生出一种新的程序设计原理与方法，并促使人们进一步去探索与研究，特别是探索研究大规模程序设计的特点与规律，创造研制正确的程序并证明其正确性的有效方法，进而实现程序自动构造的方法与工具。

3. 对计算机程序设计教学的影响

结构化程序设计的提出向传统的程序设计教学方法提出了一个挑战，这就是说，讲授程序设计，不应是仅仅讲授一种程序设计语言，告诉学生可以用它写出怎样的程序，而是应该讲授怎样用它来编写易读易理解的程序。也就是说，教师不仅仅要用仔细挑选的一些例子去解释阐明程序成分直到整个程序，还应该传授一种系统的有纪律性的思考方式，从而能对程序进行组织和编码，使程序容易理解与修改，能保证其正确性。

结构化程序设计方法的主要技术是自顶向下、逐步求精。具体地说，就是在接受一个任务之后，纵观全局，先设想好整个任务分为几个子任务，每一个子任务又可以进行细分，直到不需要细分为止。这种方法就叫做“自顶向下、逐步求精”。比如，设计房屋就是采用了自顶向下、逐步求精的方法，即先进行整体规划，然后确定建筑物方案，再进行各部分结构的设计,最后进行细节的设计(如门窗、楼道、给排水等)。

采用这种方法考虑问题比较周全，结构清晰，层次分明。用这种方法也便于验证算法的正确性。在向下一层细分之前应检查本层设计是否正确，只有上一层是正确的才可以继续细分。如果每一层设计都没有问题，则整个算法就是正确的。由于每一层向下细分时都不太复杂，因此容易保证整个算法的正确性。检查时也是由上而下逐层检查，这样做思路清晰，可以有条不紊地一步一步地进行，既严谨又方便。

我们应当掌握自顶向下、逐步求精的设计方法。这种设计方案的过程是将问题求解由抽象逐步具体化的过程。自顶向下是一种分解问题的技术，与控制结构无关；逐步求精是结构化程序的连续分解，最终使其成为顺序、选择和循环这三种基本控制结构的组合。结构化程序设计的结果是使一个结构化程序最终由若干个过程组成，每一过程完成一个确定的功能，并且每一过程都是由顺序、选择和循环这三种基本控制结构组成的。

1.4.2 结构化程序设计的特征与风格

结构化程序设计的主要特征与风格如下所述。

(1) 一个程序按结构化程序设计方式构造时，一般地总是一个结构化程序，即由三种基本控制结构：顺序结构、选择结构和循环结构构成。这三种结构都是单入口/单出口的程序结构。已经证明，一个任意大且复杂的程序总能转换成这三种标准形式的组合。

(2) 有限制地使用 goto 语句。鉴于 goto 语句的存在使程序的静态书写、顺序与动态执行顺序十分不一致，导致程序难读难理解，容易存在潜在的错误，难于证明正确性，有人