大学计算机程序设计基础（C++）

景 红 主编

DAXUE JISUANJI CHENGXU SHEJI JICHU（C++）

西南交通大学出版社
Http://press.swjtu.edu.cn

大学计算机程序设计基础（C++）

景 红 主 编

西南交通大学出版社
·成 都·

图书在版编目（CIP）数据

大学计算机程序设计基础：C++ / 景红主编. —成都：西南交通大学出版社，2008.3
ISBN 978-7-81104-828-5

Ⅰ. 大… Ⅱ. 景… Ⅲ. C 语言－程序设计－高等学校－教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2008）第 021507 号

大学计算机程序设计基础（C++）

景　红　主编

*

责任编辑　万　方
封面设计　本格设计
西南交通大学出版社出版发行
成都二环路北一段 111 号　邮政编码：610031　发行部电话：028-87600533
http: //press.swjtu.edu.cn
四川锦祝印务有限公司印刷

*

成品尺寸：185 mm×260 mm　　印张：13.125
字数：328 千字　　印数：1—4 500 册
2008 年 3 月第 1 版　　2008 年 3 月第 1 次印刷
ISBN 978-7-81104-828-5
定价：22.00 元

前　言

计算机程序设计课程从20世纪70年代末起步发展至今，发生的变化很大，课程的地位从计算机专业的基础课发展到非计算机专业的公共基础课，教学语言从Basic，Pascal，Fortran发展到C，C++，并且已经奠定了它作为高校一门重要的计算机基础课程的地位。但受到传统教学理念的影响，课程教学一般围绕语言自身的体系展开，忽略了培养学生的程序设计能力。一方面，由于语言涉及的概念多和语法比较复杂，使很多学生感到难以入门；另一方面，显现出熟知语法知识的学生在编程解题时仍感无从下手的教学弊端。

近年来，西南交通大学软件学院为了满足本校各专业人才培养的需求，在计算机基础教育的理论和实践等方面进行了一系列改革，计算机程序设计基础课程的建设也不例外。我们改革的思路是：课程体系突出程序设计而不是语法；以C++语言作为教学语言，由面向过程到面向对象，逐步深入；将基础教学与实践教学相结合，突出编程能力的培养。

本书是计算机程序设计基础课程的配套教材，是在西南交通大学本科“计算机程序设计基础与VC++”胶印教材的基础上，根据教学效果对其内容进行更新和调整之后编写的。与传统教材相比，本书最大的特点就是：以应用为主线组织教学内容，采用案例教学策略，通过问题引出/案例分析，重点讲解程序设计的思想和方法，辅以语言知识的介绍，用学生容易理解的方式讲清基本概念和方法。因此，全书各章节都有针对性地设计了大量的编程实例，并从思路分析、算法描述、源程序清单到编程技巧等方面做了细致和全面的讨论与介绍。

与本书配套的还有《实验指导》教材，可引导学生通过大量的上机练习，包括基本训练和综合训练，在实践中掌握知识，培养程序设计的能力，逐步理解和掌握程序设计的思想和方法，进而具备利用计算机解决实际问题的初步能力。

参加本书编写的人员（按姓氏笔画排列：王国强、刘军、刘霓、辛勤、景红）都是在高校长期从事计算机教学工作的一线教师，有着丰富的教学经验，力图编写出一本使教与学“C++”更容易的教材。但由于时间仓促，错误难免，敬请读者不吝赐教。

编　者

2008年1月于西南交通大学

目　录

第1章

引 论

教学目标

- 了解编程求解问题的全过程；
- 了解算法基础知识；
- 掌握结构化算法的表示方法；
- 理解常用算法设计思想。

教学内容

- 软件开发和程序设计的概念；
- 算法的概念；
- 算法的表示方法；
- 算法设计策略；
- 算法复杂性分析。

§1.1 软件开发和程序设计

所谓**软件**泛指程序和相关文档的集合。**文档**是为方便了解程序所需要的资料说明，这些资料并不一定要装入计算机；而**程序**作为一种具有逻辑结构的信息，是计算任务的处理对象和处理规则的描述，这一描述必须通过相应的实体才能体现。从用户的角度来看，软件决定着计算机做什么和如何做，是用户与硬件之间的交互界面；从商业的角度来看，当程序作为商品以有形介质为载体进行交易时即为软件。

随着计算机技术的迅猛发展，软件开发商们为用户设计编写了大量的应用软件，使得用户在日常工作中遇到的任务多数都可以借助现有的软件来完成，如文字处理软件 Word、表格处理软件 Excel、网页制作软件 FrontPage 等。但往往需要解决的问题繁杂多变，特别是在工程应用领域，很难设计出一个包罗万象的通用软件；同时面对大量的具体问题，使用通用软件来解决其效率会很低，甚至可能无法完成任务。在这种情况下，由用户**自行开发**具有针对性的应用软件就成为唯一的解决办法。

1.1.1 软件开发过程

从开始研制软件到废弃该软件的整个期间，称为软件生命期。

传统的软件开发过程可划分为七个阶段，见图 1.1。

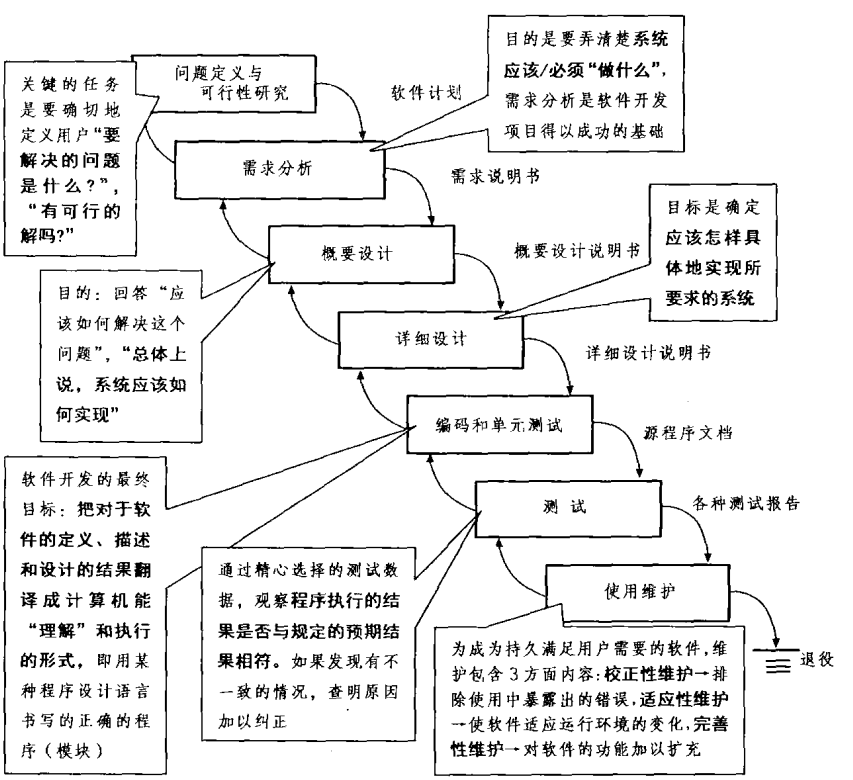


图 1.1 传统的软件开发过程

在这种方式下的软件开发过程中，程序设计员凭借自身的经验和认识，对系统功能进行分解和抽象，每一阶段的工作建立在前一阶段工作的基础上。有时程序设计员由于受到认识深度的限制，某个阶段的程序开发工作可能有误，而且这种错误通常只有到后面的工作中才能显现出来。由于开发工作一环套一环，因此一旦出现这种情况，要更改前期工作就十分困难，有时甚至要把全部工作推翻重来。

基于上述原因，应该使分析、设计和实现一个软件系统的过程与人们认识这个系统的过程同步，做到这一点的可能途径之一就是使用与人类认知规律相近的方法和方式去分析、设计和实现一个软件系统，即所谓面向对象技术。

1.1.2 程序设计方法

程序设计是一个将人类思维转化为计算机思维的过程，可分为面向过程的程序设计和面向对象的程序设计两大类。

1. 面向过程的程序设计

为了得到问题的解而执行的一步一步地操作，称为**过程**。面向过程的程序设计是一种基于功能分析及每个功能由计算机的一个操作过程实现的程序设计方法，又称为**传统的程序设计**。面向过程程序设计的关键是规划算法和数据结构。

2. 面向对象的程序设计

面向对象程序设计模拟自然界认识和处理事物的方法，将数据和对数据的操作方法组织在一起，形成一个相对独立的整体，称为**对象**。对象是活动的，对象行为靠消息触发而激活。面向对象程序设计的关键是确定对象并对其分类。

1.1.3 程序设计过程

传统的程序设计过程主要包括以下几个阶段：

1. 分析问题

通过原始资料，取得对问题的一个清晰的理解，进而确定解决问题的目标（称为输出）以及实现该目标所需要的条件（称为输入）。

2. 设计算法与数据结构

数据结构描述了问题涉及的对象之间的联系和组织结构，算法描述了求解问题的步骤或规则。设计合理的数据结构往往可以简化算法，而好的算法又使程序具有更高的效率。

3. 检查算法

使用多组样本数据，通过手工计算，对方案的正确性进行证明和验证。

4. 编码实现

选用一种程序设计语言（如 C++语言）将算法转换成计算机能够理解的程序（称为编程）。良好的编程风格是程序具备可靠性、可读性和可维护性的基本保证。

5. 测试和调试程序

“测试”是在计算机上用样本数据运行程序，测试代码的正确性。“调试”就是查找和排除程序错误，直到能够得到正确的运行结果为止。通常将程序中的错误称为 bug，它可能是

语法错误，也可能是逻辑错误。大多数语法错误容易找到和改正，但逻辑错误就很难找到，因为导致逻辑错误的原因很多。

§1.2 计算机算法

1.2.1 算法的概念

1. 计算机算法

简单地讲，**算法**就是解决某类具体问题的方法，是解决问题的步骤序列。在我们的日常生活中，这样的事例随处可见。例如看电影，通常需要通过如下的步骤来实现：买电影票→持票按时到指定电影院进场→按指定座位坐下→看电影→退场。

计算机算法是指利用计算机解决问题的方法，简称算法，分为数值算法和非数值算法两大类。其中非数值算法应用范围十分广泛，最常见的是事务管理领域，例如图书检索、人事管理、行车调度管理等。

2. 算法主要特征

一般利用计算机解决问题的算法应具有以下几个特征：

① 有效性：算法中的每一步骤都应当是可执行的，并能得到确定的结果；

② 有穷性：算法应该在执行有限的步骤后结束，而不能是无限的；

③ 确定性：组成算法的每一步骤应该具有确定的语义，而不能有任何歧义；

④ 有零个或多个输入：部分数据在操作之前需要通过外界赋值，称为算法的输入，是算法加工的对象，一个算法可以没有输入，也可以有多个输入；

⑤ 有一个或多个输出：算法执行过程中对外界产生的任何影响（算法的运算结果）是算法的输出，一个算法可以有一个或多个输出。

3. 算法主要因素

为了有效地解决问题，不仅需要保证算法正确，还要考虑算法的质量。要设计一个“好”的算法，通常需要考虑以下因素：

① 正确性：算法应满足具体问题的要求；

② 可读性：一个可读性好的算法有助于对算法的理解，易于调试和修改；

③ 健壮性：当输入非法数据时，算法也能够做出适当的反应或处理，而不会产生一些莫名其妙的结果，更不会出现死机、系统崩溃等情况；

④ 高效性：算法的效率与算法占用计算机设备资源成反比，而最突出的就是计算机的运行时间和计算机的存储空间。占用时间越短效率越高，占用内存空间越少效率越高。

1.2.2 算法的表示

描述算法的方法有很多，常见的方法有自然语言、传统流程图、结构化流程图（三种基本结构的流程图、N-S 流程图）、伪代码等。

1.2.2.1 自然语言表示

自然语言是指人们日常使用的语言，如汉语、英语等。很显然使用自然语言描述算法，通俗易懂；但叙述文字冗长，含义不尽严格，容易出现“歧义性”。

【例 1.1】 用自然语言描述求解“5!=？”的算法。

步骤 1：先求 1×2，得到结果 2；
步骤 2：将步骤 1 得到的结果 2 再乘以 3，得到结果 6；
步骤 3：将步骤 2 得到的结果 6 再乘以 4，得到结果 24；
步骤 4：将步骤 3 得到的结果 24 再乘以 5，得到最终结果 120；
步骤 5：输出结果 120。

【例 1.2】 用自然语言描述求解“100!=？”的算法。

说明：

此时不能采用【例 1.1】介绍的方法求解，因为需要描述 99 个步骤而不可取。我们通过分析可以发现，求阶乘实际上就是一个循环求积的处理过程，因此可采用如下方法求解：

S1：输入 n 值
S2：将 1⇒t
S3：将 1⇒i
S4：将 t×i⇒t
S5：将 i+1⇒i
S6：若 i≤n 成立，返回重新执行 S4，以及其后的步骤 S5；否则执行 S7；
S7：输出 t，算法结束。

设一个变量（设为 t）代表被乘数，另外一个变量（设为 i）代表乘数，直接将每一步的乘积放在被乘数 t 变量中

1.2.2.2 传统流程图表示

美国国家标准化协会 ANSI（American National Standard Institute）规定使用图符表示特定的操作，称为流程图（又称为传统流程图），见图 1.2，并已为世界各国程序设计者普遍采用。

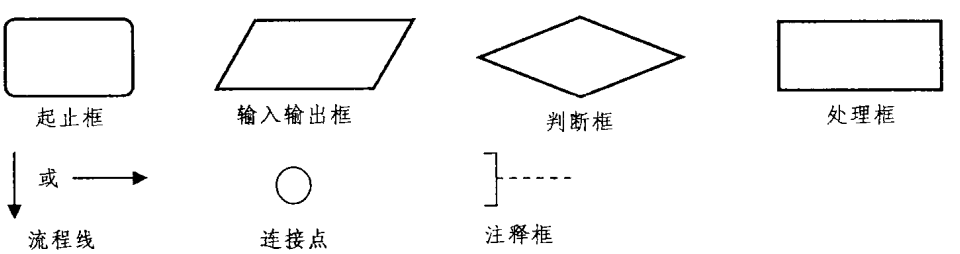


图 1.2 流程图符号

【例 1.3】 用传统流程图描述求解“5!=？”的算法（见图 1.3）。

【例 1.4】 用传统流程图表示“判定一个大于或等于 3 的正整数是否是素数？”的算法（见图 1.4）。

说明：

所谓素数（质数），是指除了 1 和该数本身以外不能被其他任何整数整除的数。因此，

判断一个数 n（n≥3）是否是素数的方法为：将 n 作为被除数，并用 2 到（n－1）之间的各个整数轮流作为除数，若都不能被整除，则 n 为素数。

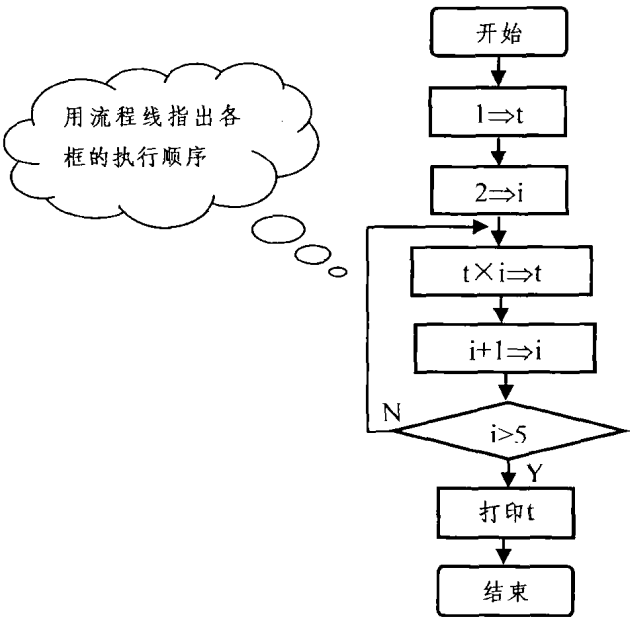


图 1.3 【例 1.3】的算法流程图

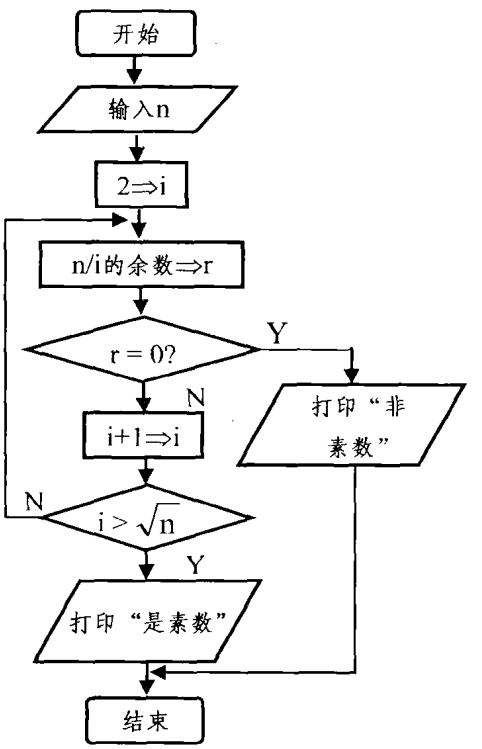


图 1.4 【例 1.4】的算法流程图

相对自然语言来看，流程图更直观形象、易于理解。传统流程图的缺陷是由于允许流程随意转移，造成算法的逻辑常常难以理解，进而降低了算法的可靠性和可维护性。

1.2.2.3 结构化流程图表示

1. 三种基本结构的流程图

1966 年，Bohra 和 Jacopini 提出了以下述三种基本结构作为表示算法的基本单元。可以这样认为，一个算法无论多么复杂，都可以由这三种基本结构组合而成。

(1) 顺序结构（见图 1.5）。从 a 点（入口）进入，自上而下顺序地先执行 A 操作，然后执行 B 操作，最后从 b 点（出口）脱离该结构。

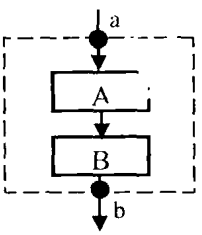


图 1.5 顺序结构

(2) 选择结构（又称选取结构），见图 1.6 (a)。从 a 点（入口）进入，首先对给出的条件 P 进行判断，如果 P 成立（Y），则执行 A 操作；P 不成立（N），则执行 B 操作；最后从 b 点（出口）脱离该结构。

如果 A 或 B 有一个为空操作(即不执行任何操作)，如图 1.6(b)(设 B 为空操作)，如果 P 成立（Y）则执行 A 操作，否则不执行任何操作，最后从 b 点（出口）脱离该结构。

需要注意的是：无论条件 P 是否成立，只能执行 A 或 B 之一，不可能既执行 A 又执行 B。

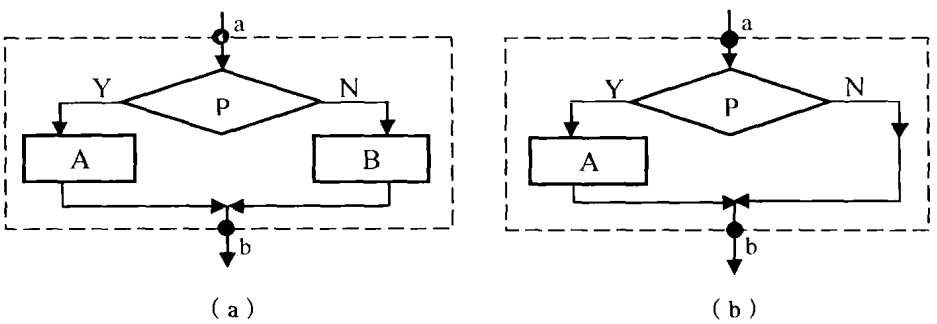


图 1.6 选择结构

(3) 循环结构（又称重复结构），即重复执行一组操作，它可进一步分为当型循环结构和直到型循环结构两类。

① 当型循环结构，见图 1.7 (a)。从 a 点（入口）进入，首先对给出的条件 P 进行判断：如果 P 成立，则执行 A 操作；执行完 A 操作后重新对条件 P 进行判断，如果 P 仍然成立，再次执行 A 操作；如此反复“判断→执行”，直到条件 P 不成立为止，最后从 b 点（出口）脱离该结构。

② 直到型循环结构，见图 1.7 (b)。从 a 点（入口）进入，首先执行 A 操作，然后对给

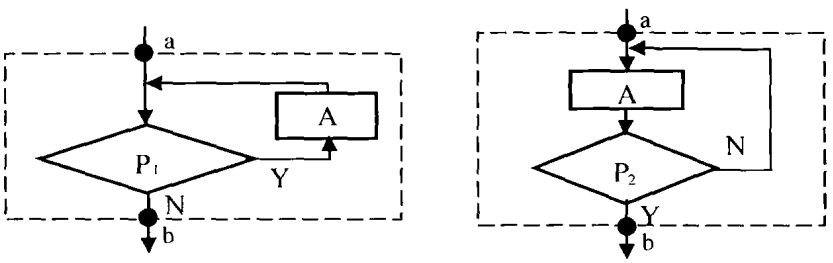


图 1.7 循环结构

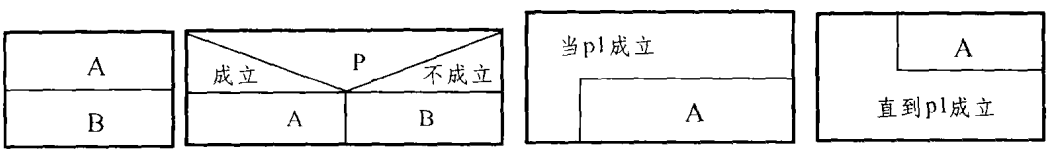
出的条件 P 进行判断，如果 P 不成立，则执行 A 操作；执行完 A 操作后重新对条件 P 进行判断，如果 P 仍然不成立，再次执行 A 操作；如此反复“判断→执行”，直到条件 P 成立为止，最后从 b 点（出口）脱离该结构。

当型循环和直到型循环是可以互相转换的，凡用当型循环处理的问题，用直到型循环可以解决，反之亦然。

这三种基本结构的共同点是：只有一个入口，只有一个出口，结构内的每一部分都有机会被执行，结构内不存在“死循环”。由这三种基本结构所构成的算法称为结构化算法，并可以组合应用来解决任何复杂问题。

2. N-S 流程图表示

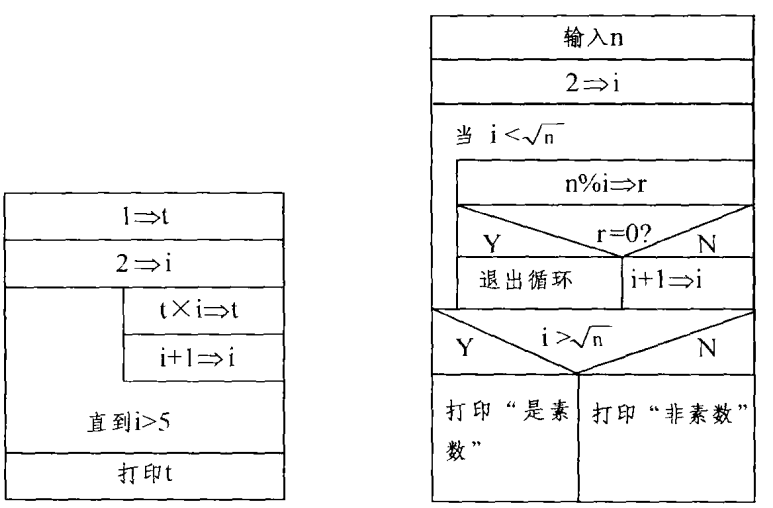
1971 年，美国学者 I. Nassi 和 B. Shneiderman 提出了一种新的流程图形式，称为 N-S 流程图，它是在三种基本结构流程图的基础上完全去掉了箭头和流程线，见图 1.8。



（a）顺序结构　（b）选择结构　（c）当型循环　（d）直到型循环

图 1.8　N-S 流程图

【例 1.5】 用 N-S 流程图表示【例 1.3】和【例 1.4】的算法（见图 1.9）。



（a）【例 1.3】的算法流程图　（b）【例 1.4】的算法流程图

图 1.9　【例 1.5】的算法流程图

【例 1.6】 用 N-S 流程图表示“判定 2000—2500 年中的每一年是否闰年，将结果输出”的算法（见图 1.10）。

说明：

能被 4 整除但不能被 100 整除的年份，或者能被 100 整除同时又能被 400 整除的年份是闰年。

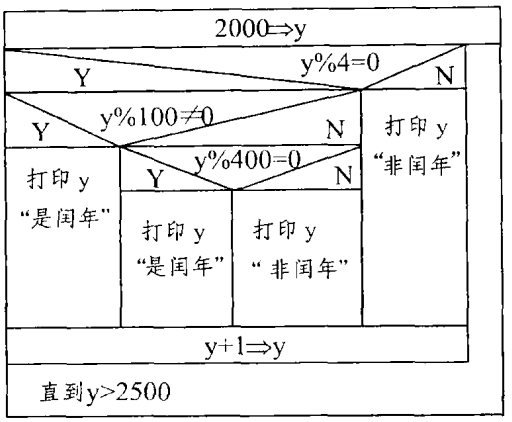


图 1.10 【例 1.6】算法流程图

1.2.2.4 伪代码表示

伪代码是用介于自然语言与计算机语言之间的文字和符号来描述算法，自上而下的每一行（或几行）表示一个基本操作，书写和修改方便，格式紧凑，比较好懂，且便于向计算机语言程序过渡。

【例 1.7】 使用伪代码表示【例 1.5】的算法。

```
input n
i←2
while i ≤ √n do
    r←n/i 的余数
    if r=0
        then break
        else i←i+1
if i>√n
    then print "是素数"
    else print "非素数"
```

1.2.3 算法设计策略

下面简要介绍几种在算法设计中经常运用的科学思维方法。

1. 枚举法

枚举法也称为穷举法，其设计思想是：在有限的范围中，列举和检验所有可能的结果，从中找出那些符合要求的候选解作为问题的解。

【例 1.8】 设有算式：A B C D－C D C=A B C，其中的 A，B，C，D 均为一位非负整数，要求找出 A，B，C，D 各值。

思路分析：

设正整数 A、B、C、D，A 和 C 的取值范围应是[1, 9]，B 和 D 的取值范围应是[0, 9]，

分别对相应范围中的每一个数值进行检测，输出满足条件（1000×a+100×b+10×c+d）－（100×c+10×d+c）=（100×a+10×b+c）的数值。

算法描述：

```
for  a←1 step 1 until 9 do
  for  b←0 step 1 until 9 do
    for  c←1 step 1 until 9 do
      for  d←0 step 1 until 9 do
        x←1000×a+100×b+10×c+d
        y←100×c+10×d+c
        z←100×a+10×b+c
        if x-y=z
            then 输出 a，b，c，d
                 跳转到 end 标号处
```

2. 递推法

递推法的设计思想是：利用问题本身所具有的一种递推关系求问题的解，即从已求得的规模为 1，2，…，n－1 的一系列解，构造出问题规模为 n 的解。

【例 1.9】 计算 n!。

思路分析：

1!=1，由 1!×2 得 2!→由 2!×3 得 3! →…由（n－1）!×n 得 n!。

算法描述：

```
f←1
i←1
while i≤n do
    f←f×i
    i←i+1
print f
```

3. 递归法

递归法的设计思想是：为求解规模较大的问题，设法将它分解成规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模较大问题的解。特别地，当规模 N=1 时，能直接得解。

【例 1.10】 计算斐波那契（Fibonacci）数列的第 n 项函数 fib(n)。

思路分析：

斐波那契数列为：0，1，1，2，3，…

即 f(0)=0

f(1)=1

f(n)=f(n-1)+f(n-2) （当 n>1 时）。

算法描述：

```
call Fibonacci （n）
if n=0
  then return 0
if n=1
  then return 1
if n>1
  then return Fibonacci（n-1）+ Fibonacci（n-2）
```

递归法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题的求解推到较简单问题的求解。例如，计算 fib(n) →计算 fib(n-1) 和 fib(n-2)→计算 fib(n-3) 和 fib(n-4)→……，以此类推，直至计算 fib（1）和 fib（0），分别能立即得到结果 1 和 0。

在回归阶段，当获得最简单情况的解后，逐级返回，依次得到稍复杂问题的解。例如，得到 fib(1) 和 fib(0) 的结果 1 和 0→得到 fib(2) 的结果→得到 fib(3) 的结果→……依次返回，直至得到 fib(n) 的结果。

4. 分治法

分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的子问题，以便各个击破，分而治之。由分治法产生的子问题往往是原问题的较小模式，在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解，这自然导致递归过程的产生。分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

1.2.4 算法复杂性分析

【例 1.11】 在一个按升序排列的 n 个元素 a_1，a_2，…，a_n（$a_i \leqslant a_{i+1}$）中，查找是否有与 b 相同的元素。

算法一： 从第一个元素 a_1 开始，用 b 与队列中的元素逐一比较。此时，最好的情况是 a_1 就是要查找的元素，只需比较一次；最坏情况则需要比较 n 次，即一直比较到 a_n 才能得到结果；假定每个元素与 b 相同是等概率的，则平均需要比较 n/2 次。

算法二： 采用折半查找（二分查找）的方法，即先用 b 与队列中位居‘中间点’的元素 $a_{(n/2)}$ 比较，若 $b=a_{(n/2)}$，则查找成功；若 $b \neq a_{(n/2)}$，同时 $b < a_{(n/2)}$，则在 a_1，a_2，…，$a_{(n/2)}$ 中采用上述方法继续查找；否则在 $a_{(n/2+1)}$，$a_{(n/2+2)}$，…，a_n 中采用上述方法继续查找，最多需要比较 log2n 次。

该例告诉我们：通常求解一个问题可能会有多种算法可供选择，选择的主要标准是算法所需要的存储空间少和执行更快等。我们要对这些可行的算法进行分析，才能知道哪一个算法效率更高。

算法的复杂性分析是对算法效率的度量，是算法运行所需要的计算机资源的量，这个量是只依赖于算法要解的问题的规模、算法的输入和算法本身的函数。需要的时间资源的量称作时间复杂性，需要的存储空间资源的量称作空间复杂性。对于给定的一个算法，怎样计量