第 1 章　程序设计基础

1.1　引例

从普通的办公自动化、上网冲浪到基因测序、嫦娥三号发射，都能看到计算机的影子；从智能手机中的单片机到“天河 2 号”超级计算机，各种计算机层出不穷。可以这么说，当今世界的各行各业都离不开计算机。那么人类是怎样“命令”计算机完成特定任务的呢？

通常情况下，人类是通过程序去“命令”计算机按照自己的意图工作的，这里大致存在两种情况：一是程序已经存在，如 Microsoft Word，专业人员使用该程序操作计算机完成相应的工作；二是程序不存在，需要组织一个团队去编写具有所需功能的程序。本书讨论的是第二种情况。

1.1.1　软硬件基础

“工欲善其事，必先利其器”。无论是编写程序还是运行程序，都需要一定的软硬件基础。

1．开发环境

开发环境就是编写程序所需的工具以及该工具运行的环境，主要包括：

（1）满足性能要求的计算机系统，包括计算机硬件和操作系统；

（2）开发工具，包括编译器、函数库、数据库等；

（3）数据库管理系统；

（4）必要的网络支持。

2．运行环境

运行环境是编写完成的应用程序运行的环境，主要包括：

（1）满足性能要求的计算机系统，包括计算机硬件和操作系统；

（2）程序运行必需的函数库、数据库系统等；

（3）必要的网络支持。

1.1.2　编写程序

具备必要的开发环境后，就要编写相应的程序来“命令”计算机完成特定的任务。下面就以模拟控制“玉兔号”月球车移动为例，来简要说明编写程序的大致过程。

【例 1.1】编写程序来控制“玉兔号”月球车的移动。

通过仔细分析这个问题，将月球表面抽象为一个平面直角坐标系（图 1.1.1），月球车处于某点（x，y）处，每一点的横、纵坐标均为整数。月球车可以实现 4 种运动方式，即向前、向后、向左、向右方每次走一个单位。

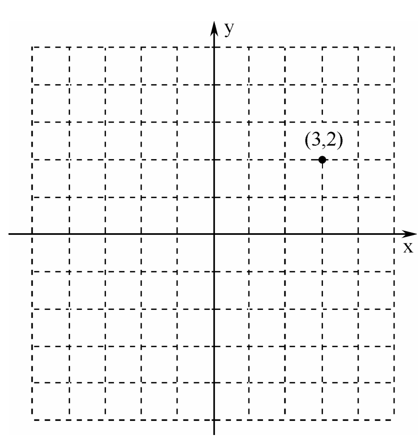

图 1.1.1　抽象的平面直角坐标系

设定 x、y、direction 三个变量，其中（x，y）来表示“玉兔号”月球车当前坐标，direction 来表示月球车接收到的移动命令。假设向前移动纵坐标值增大，向后

移动纵坐标值减小，向左移动横坐标值减小，向右移动横坐标值增大，那么 x、y、direction 三个变量之间的关系如表 1.1.1 所示。

表 1.1.1　控制“玉兔号”月球车移动命令说明

direction	direction 含义	目标点横（纵）坐标值变化
1	向前	y = y + 1
2	向后	y = y - 1
3	向左	x = x – 1
4	向右	x = x + 1

接下来，列出解决该问题的主要步骤，即所谓的算法。

（1）输入月球车初始坐标值（x，y）。

（2）输入移动命令给 direction。

（3）如果 direction<1 或 direction>4，转步骤（5）；否则，按表 1.1.1 所示规则，分情况求得移动后目标点的坐标值。

（4）输出求得的目标点的坐标值（x，y），转步骤（2）

（5）算法结束。

在设计好算法后，应该使用流程图作为工具来描述算法，如图 1.1.2 所示。

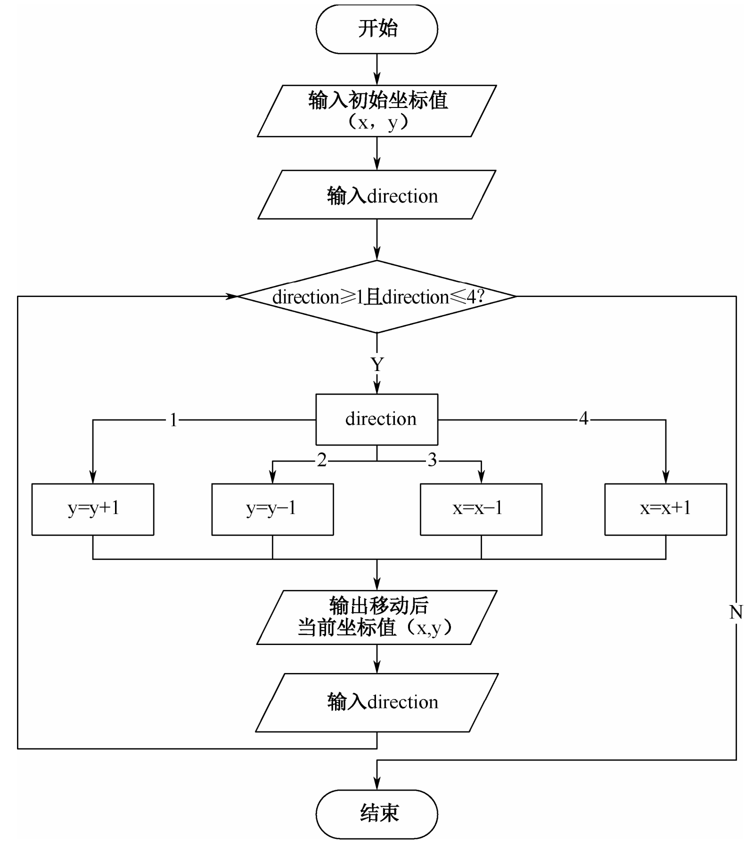

图 1.1.2　模拟控制“玉兔号”月球车移动流程图

为了实现以上的解决方案，使用 C 语言编写程序来实现该算法：

```
//该程序用来模拟控制"玉兔号"月球车的移动
#include<stdio.h>
int main()
{
    int x,y,direction;
    printf("请输入月球车的初始坐标（x, y）\n");
    scanf("%d,%d",&x,&y);
    printf("请输入移动方向（1-前  2-后   3-左  4-右）：");
    scanf("%d",&direction);
    while(direction>=1&&direction<=4)
    {
        switch(direction)
        {
        case  1:y=y+1;break;       //前
        case  2:y=y-1;break;       //后
        case  3:x=x-1;break;       //左
        case  4:x=x+1;break;       //右
        }
        printf("月球车当前坐标为（%d, %d）\n",x,y);
        printf("请输入移动方向（1-前  2-后   3-左  4-右）：");
        scanf("%d",&direction);
    }
    return 0;
}
```

在编写好程序后，应该仔细检查程序是否能够解决问题。

1.2 算法

由上面的介绍可知，设计算法是人们编写程序去"命令"计算机解决问题过程中非常重要的一环，算法是计算机科学最基本的概念，是计算机科学研究的核心之一。因此，了解算法及其表示和设计方法是程序设计的基础和精髓，也是读者学习程序设计过程中最重要的一环。

1.2.1 算法及其特性

1．什么是算法

算法（Algorithm）就是一组有穷的规则，它规定了解决某一特定问题的一系列运算。通俗地说，为解决问题而采用的方法和步骤就是算法。本书中讨论的算法主要是指计算机算法。

2．算法的特性

（1）确定性（Definiteness）。算法的每个步骤必须要有确切的含义，每个操作都应当是清晰的、无二义性的。例如，算法中不允许出现诸如"将 3 或 5 与 y 相加"等含混不清、具有歧义的描述。

（2）有穷性（Finiteness）。一个算法应包含有限的操作步骤且在有限的时间内能够执行完毕。例如，在计算下列近似圆周率的公式时：

$$\frac{\pi}{4}\approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots \qquad (1.2.1)$$

当某项的绝对值小于 10^{-6} 时算法执行完毕。

> ☞注意：如何正确理解算法的有穷性？
>
> 一个实用的算法，不仅要求步骤有限，同时要求运行这些步骤所花费的时间是人们可以接受的。例如，使用暴力破解密码的算法可能要耗费成百上千年。显而易见，这个算法是可以在有限的时间内完成，但是对于人类来说是无法接受的。

（3）有效性（Effectiveness）。算法中的每个步骤都应当能有效地执行，并得到确定的结果。例如，算法中包含一个 m 除以 n 的操作，若除数 n 为 0，则操作无法有效地执行。因此，算法中应该增加判断 n 是否为 0 的步骤。

（4）有零个或多个输入（Input）。在算法执行的过程中需要从外界取得必要的信息，并以此为基础解决某个特定问题。例如，在求两个整数 m 和 n 的最大公约数的算法中，需要输入 m 和 n

的值。另外，一个算法也可以没有输入，例如，在计算式（1.2.1）时，不需要输入任何信息，就能够计算出近似的 π 值。

（5）有一个或多个输出（Output）。设计算法的目的就是要解决问题，算法的计算结果就是输出。没有输出的算法是没有意义的。输出与输入有着特定的关系，通常，输入不同，会产生不同的输出结果。

☞注意：如何正确理解算法的输出形式？

算法的输出就是算法的计算结果，其输出形式多种多样：打印数值、字符、字符串，显示一幅图片，播放一首歌曲或音乐，播放一部电影……

3．算法的分类

根据待解决问题的形式模型和求解要求，算法分为数值和非数值两大类。

（1）数值运算算法。数值运算算法是以数学方式表示的问题求数值解的方法。例如，代数方程计算、线性方程组求解、矩阵计算、数值积分、微分方程求解等。通常，数值运算有现成的模型，这方面的现有算法比较成熟。

（2）非数值运算算法。非数值运算算法通常为求非数值解的方法。例如，排序、查找、表格处理、文字处理、人事管理、车辆调度等。非数值运算算法种类繁多，要求各自不同，难以规范化。本节主要讲述的是一些典型的非数值运算算法。

1.2.2 算法的表示方法

设计出一个算法后，为了存档，以便将来算法的维护或优化，或者为了与他人交流，让他人能够看懂、理解算法，需要使用一定的方法来描述、表示算法。算法的表示方法很多，常用的有自然语言、流程图、伪代码和程序设计语言等。

1．自然语言（Natural Language）

用人们日常生活中使用的语言，如中文、英文、法文等来描述算法。

【例 1.2】使用中文来描述计算 5！的算法，其中假设变量 t 为被乘数，变量 i 为乘数。

（1）初始化 t=1。

（2）初始化 i=2。

（3）计算 t×i，乘积仍放在 t 中。

（4）将 i 的值加 1 再放回到 i 中。

（5）如果 i 不大于 5，则返回步骤（3）执行；否则，进入步骤（6）。

（6）输出 t 中存放的 5!值。

使用自然语言描述算法的优点是通俗易懂，没有学过算法相关知识的人也能够看懂算法的执行过程。但是，自然语言本身所固有的不严密性使得这种描述方法存在以下缺陷：

① 文字冗长，容易产生歧义性，往往需要根据上下文才能判别其含义；

② 难以描述算法中的分支和循环等结构，不够方便、直观。

2．流程图（Flow Chart）

流程图是最常见的算法图形化表达，它使用美国国家标准化学会（American National Standards Institute，ANSI）规定的一些图框、线条来形象、直观地描述算法处理过程。常见的流程图符号如表 1.2.1 所示。

表 1.2.1　常见流程图符号

符号名称	图　形	功　能
起止框	（圆角矩形）	表示算法的开始或结束
处理框	（矩形）	表示一般的处理操作，如计算、赋值等

续表

符号名称	图形	功能
判断框		表示对一个给定的条件进行判断
流程线		用流程线连接各种符号，表示算法的执行顺序
输入/输出框		表示算法的输入/输出操作
连接点		成对出现，同一对连接点内标注相同的数字或文字，用于将不同位置的流程线连接起来，避免流程线的交叉或过长

【例 1.3】使用流程图来描述计算 5!的算法，其中假设变量 t 为被乘数，变量 i 为乘数。流程图如图 1.2.1 所示。

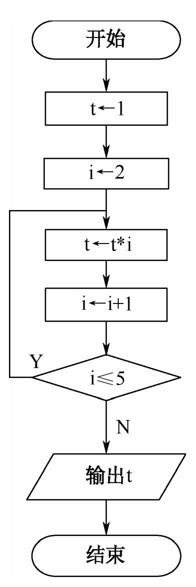

图 1.2.1　计算 5!的流程图

从本例可以看出，使用流程图描述算法简单、直观，能够比较清楚地显示出各个符号之间的逻辑关系，因此流程图是一种表示算法的好工具。流程图使用流程线指出各个符号的执行顺序，对流程线的使用没有严格限制，使用者可以毫无限制地使流程随意地转来转去。但是，当算法规模较大，操作比较复杂时，人们难以理解算法的逻辑。

为了提高算法的质量，便于阅读理解，应限制流程的随意转向。为了达到这个目的，人们规定了 3 种基本结构，由这些基本结构按一定规律组成一个算法结构。

（1）顺序结构。顺序结构是最简单、最常用的一种结构，如图 1.2.2 所示。图中操作 A 和操作 B 按照出现的先后顺序依次执行。

（2）选择结构。选择结构又称为分支结构。这种结构在处理问题时根据条件进行判断和选择。图 1.2.3（a）是一个“双分支”选择结构，如果条件 p 成立则执行处理框 A，否则执行处理框 B。图 1.2.3（b）是一个“单分支”选择结构，如果条件 p 成立则执行处理框 A。

（3）循环结构。循环结构又称为重复结构，在处理问题时根据给定条件重复执行某一部分的操作。循环结构有当型和直到型两种类型。

当型循环结构如图 1.2.4 所示。功能是：当条件 p 成立时，执行处理框 A，执行完处理框 A 后，再判断条件 p 是否成立，若条件 p 仍然成立，则再次执行处理框 A，如此反复，直至条件 p 不成立才结束循环。

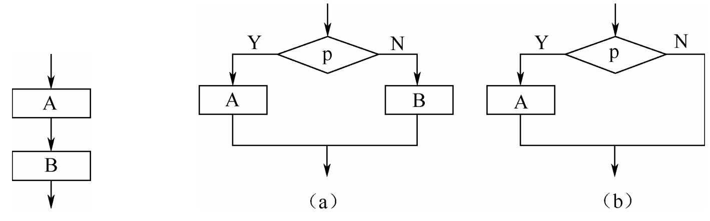

图 1.2.2　顺序结构　　图 1.2.3　选择结构

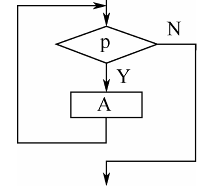

图 1.2.4　当型循环结构

直到型循环结构如图 1.2.5 所示。功能是：先执行处理框 A，再判断条件 p 是否成立，如果条件不成立，则再次执行处理框 A，如此反复，直至条件 p 成立才结束循环。

当型循环结构与直到型循环结构的区别如表 1.2.2 所示。

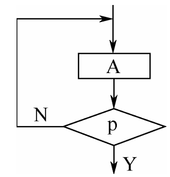

图 1.2.5　直到型循环结构

表 1.2.2　当型循环结构与直到型循环结构的比较

比 较 项 目	当型循环结构	直到型循环结构
何时判断条件是否成立	先判断，后执行	先执行，后判断
何时执行循环	条件成立	条件不成立
循环至少执行次数	0 次	1 次

以上 3 种基本结构具有以下的特点：

（1）只有一个入口，只有一个出口；

（2）结构中的每一部分都有机会被执行到；

（3）结构内不存在“死循环”。

计算机科学家已经证明，使用以上 3 种基本结构顺序组合而成的算法结构，可以解决任意复杂的问题。由基本结构所构成的算法就是所谓的“结构化”的算法。

自主学习：N-S 流程图

N-S 流程图是由美国学者 I.Nassi 和 B.Shneiderman 在 1973 年提出的一种新的流程图形式。读者可以上网查阅相关资料深入学习这种算法表示方法。

3．伪代码（Pseudocode）

虽然使用流程图来描述算法简单、直观，易于理解，但是画起来费事，修改起来麻烦。因此流程图比较适合于算法最终定稿后存档时使用，而在设计算法的过程中常用一种称为“伪代码”的工具。

伪代码是一种介于自然语言和程序设计语言之间描述算法的工具。程序设计语言中与算法关联度小的部分往往被伪代码省略，如变量的定义等。

【例 1.4】使用伪代码来描述计算 5!的算法，其中假设变量 t 为被乘数，变量 i 为乘数。

```
begin
    t←1
    i←2
    while(i≤5)
    {
        t←t*i
        i←i+1
    }
    print t
end
```

4．程序设计语言（Programming Language）

计算机无法识别自然语言、流程图、伪代码，因此算法最终要用程序设计语言实现，再被翻译成可执行程序后在计算机中执行。用程序设计语言描述算法必须严格遵守所选择语言的语法规则。

【例 1.5】使用 C 语言来描述计算 5!的算法。

```
#include<stdio.h>
int main()
{
    int i,t;
    t=1;
    i=2;
    while(i<=5)
    {
        t=t*i;
        i=i+1;
    }
    printf("5!=%d\n",t);
    return 0;
}
```

本节介绍了 4 种算法描述方法，读者可根据自己的喜好和习惯，选择其中一种。建议在设计算法过程中使用伪代码，交流算法思想或存档算法时使用流程图。

1.2.3 算法设计的基本方法

算法设计的任务是对各类问题设计良好的算法及研究设计算法的规律和方法。常用的数值算法设计方法包括迭代法、插值法、差分法等；非数值算法设计方法包括分治法、贪心法、回溯法等；还有些方法既适用于数值算法的设计也适用于非数值算法的设计。本书主要介绍后两种情况下算法设计的基本方法，数值算法的设计方法请读者参考《计算方法》、《数值分析》等课程的教材。

针对一个给定的实际问题，要找出确实行之有效的算法，就需要掌握算法设计的策略和基本方法。算法设计是一个难度较大的工作，初学者在短时间内很难掌握。但所幸的是，前人通过长期的实践和研究，已经总结出了一些算法设计的基本策略和方法，如穷举法、递推法、递归法、分治法、回溯法、贪心法、模拟法和动态规划法等。

1．穷举法（Exhaustive Algorithm）

穷举法（Exhaustive Algorithm）也称为枚举法、蛮力法，是一种简单、直接解决问题的方法。

使用穷举法解决问题的基本思路：依次穷举问题所有可能的解，按照问题给定的约束条件进行筛选，如果满足约束条件，则得到一组解，否则不是问题的解。将这个过程不断地进行下去，最终得到问题的所有解。

要使用穷举法解决实际问题，应当满足以下两个条件：

（1）能够预先确定解的范围并能以合适的方法列举；

（2）能够对问题的约束条件进行精确描述。

穷举法的优点是：比较直观，易于理解，算法的正确性比较容易证明；缺点是：需要列举许多种状态，效率比较低。

【例 1.6】使用流程图描述判断一个正整数 m（m≥2）是否为素数的算法。

素数也称为质数，其特点是，除了 1 和它自身之外没有其他的约数。例如，19 除了 1 和 19 之外没有其他约数，因此它就是一个素数。

通过观察和分析，该问题的求解符合穷举法解决问题的条件。

① 给定正整数 m 所有可能的约数（除了 1 和它自身）在[2,m−1]区间内，而且每两个可能的约数之间差 1。

② 约束条件非常容易描述：m mod n = 0，其中 n 表示正整数 m 的所有可能的约数，mod 的含义是求 m 除以 n 的余数，即求余运算，高级程序设计语言中基本都有该运算符。

判别素数算法的基本思路是，从区间[2,m−1]中取出一个数，看它是否满足约束条件 m mod n=0，如果满足，那么根据素数的定义可知，m 不是素数，算法结束；否则继续判别下一个可能的约数 n+1。重复上述过程，直到区间[2,m−1]中每个数都被判断过，并且都不是 m 的约数为止，这说明 m 是素数。流程图如图 1.2.6 所示。

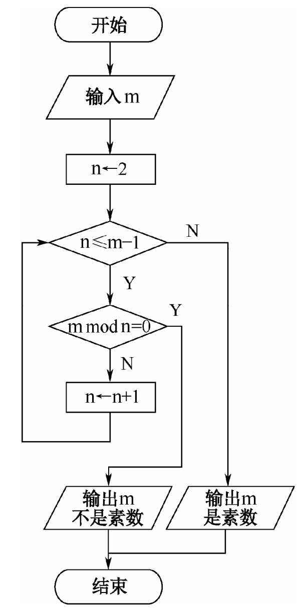

图 1.2.6　判别素数流程图

☞思考：如何优化判断素数算法，使其比较次数大幅减少？

2．递推法（Recurrence）

递推法是一种重要的算法设计思想。一般是从已知的初始条件出发，依据某种递推关系，逐次推出所要求的各中间结果及最后结果。其中初始条件可能由问题本身给定，也可能是通过对问

题的分析与化简后确定。在实际应用中，题目很少会直接给出递推关系式，而是需要通过分析各种状态，找出递推关系式，这也是应用递推法解决问题的难点所在。

在日常应用中，递推算法可分为顺推法和逆推法两种。

（1）顺推法。从已知条件出发，逐步推算出要解决问题的方法。例如，斐波那契数列就可以通过顺推法不断推算出新的数据。

（2）逆推法。逆推法也称为倒推法，是顺推法的逆过程。该方法从已知的结果出发，用迭代表达式逐步推算出问题开始的条件。

【例 1.7】使用顺推法解决斐波那契数列问题。

1202 年，意大利数学家斐波那契在他的著作《算盘书》中提出了一个有趣的问题：有一对兔子，从出生后第 3 个月起每个月都生 1 对小兔子。小兔子长到第 3 个月后每个月又生 1 对小兔子。假定在不发生死亡的情况下，由 1 对刚出生的兔子开始，1 年能繁殖出多少对兔子？

解题思路：

（1）首先，将兔子按照出生的月数分为 3 种：满 2 个月以上的兔子为老兔子；满 1 个月但是不满 2 个月的兔子为中兔子；不满 1 个月的兔子为小兔子。其中，只有老兔子具有繁殖能力。

（2）第 1 个月时，只有 1 对小兔子，总数为 1 对；

（3）第 2 个月时，1 对小兔子长成了 1 对中兔子，总数仍为 1 对；

（4）第 3 个月时，1 对中兔子长成了老兔子，并繁殖了 1 对小兔子，总数是 2 对；

（5）第 4 个月时，1 对小兔子长成了中兔子，原来的 1 对老兔子又繁殖了 1 对小兔子，总数是 3 对；

（6）以此类推，兔子的繁殖过程如表 1.2.3 所示。

表 1.2.3　兔子繁殖过程

月份	小兔子对数	中兔子对数	老兔子对数	兔子总数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13
8	8	5	8	21
9	13	8	13	34
10	21	13	21	55
11	34	21	34	89
12	55	34	55	144

解答：从表 1.2.3 可以看到每个月的兔子总数依次是 1,1,2,3,5,8,13,21,34,55,89,144,…，这就是著名的斐波那契数列。这个数列有着非常明显的特点，即从第 3 项开始，每一项都是其前面相邻两项之和。使用数学公式来表示：

$$\begin{cases} F_1 = 1 & (n = 1) \\ F_2 = 1 & (n = 2) \\ F_n = F_{n-1} + F_{n-2} & (n \geq 3) \end{cases} \tag{1.2.2}$$

从以上的分析可知，斐波那契数列可使用递推算法来计算求得，式（1.2.2）就是递推关系式。图 1.2.7 就是使用顺推法解决斐波那契数列问题的流程图（流程图中略去了输出斐波那契数列的部分）。

【例 1.8】使用逆推法解决猴子吃桃问题。

问题描述：猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半多一个。到第 10 天早上想再吃时，只剩一个桃子了。问第一天共摘了多少桃子？

解题思路：通过分析可知，该题是一个逆推问题：已知结果（第 10 天只剩 1 个桃子），规律是每天早上都吃了前一天剩下桃子的一半多一个，要求逐步推算出开始的条件（第 1 天总共摘了多少桃子）。

解答：假设第 i 天的桃子数为 p_i（$1 \leq i \leq 9$），由题意可知：

$$p_{i+1} = (p_i / 2 - 1) \tag{1.2.3}$$

从而推导出式（1.2.4）

$$p_i = \begin{cases} 1, & (i = 10) \\ 2(p_{i+1} + 1) & (1 \leq i \leq 9) \end{cases} \tag{1.2.4}$$

在此基础上，以第 10 天的桃子数 1 为基数，用上面的递推公式，可以推出第 1 天的桃子数。推导过程如表 1.2.4 所示，具体算法的流程图如图 1.2.8 所示。

表 1.2.4　猴子吃桃问题推导过程

天	计算过程	桃子数
10	p_{10}=1	1
9	p_9=2×（p_{10}+1）=2×（1+1）=4	4
8	p_8=2×（p_9+1）=2×（4+1）=10	10
7	p_7=2×（p_8+1）=2×（10+1）=22	22
6	p_6=2×（p_7+1）=2×（22+1）=46	46
5	p_5=2×（p_6+1）=2×（46+1）=94	94
4	p_4=2×（p_5+1）=2×（94+1）=190	190
3	p_3=2×（p_4+1）=2×（190+1）=382	382
2	p_2=2×（p_3+1）=2×（382+1）=766	766
1	p_1=2×（p_2+1）=2×（766+1）=1534	1534

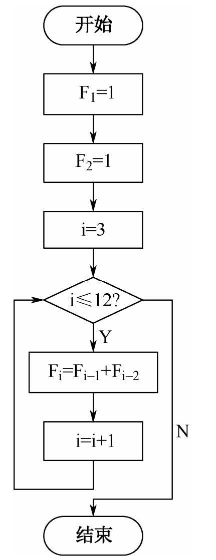

图 1.2.7　求解斐波那契数列的流程图

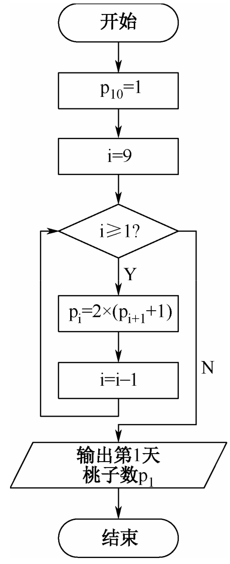

图 1.2.8　猴子吃桃流程图

3．递归法（Recursive）

递归法是算法设计基本方法中比较难的部分。为了搞清楚递归算法的基本思想，先研究一下下面的例子。

【例 1.9】分别使用递推法和递归法求解 1+2+3+…+100。

（1）递推法。使用递推法计算 1+2+3+…+100 的基本思路如下：

① 设置一个放置和的变量 sum，设置该变量的初值为 0；

② 设置一个放置加数的变量 i，设置其初值为 1；

③ 如果加数 i≤ 100，那么将加数 i 加到放置和的变量 sum 中，即 sum=sum+i；否则转⑤；

④ 计算新的加数 i，即 i=i+1，转③；

⑤ 输出 1+2+3+…+100 的和 sum；

⑥ 算法结束。

最终得到如图 1.2.9 所示的流程图。

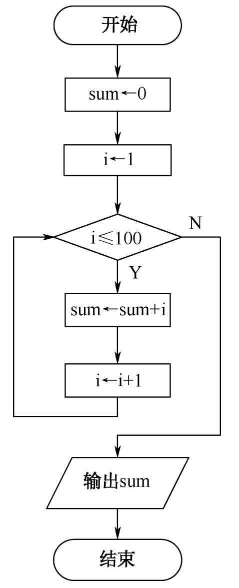

图 1.2.9　计算 1+2+3+…+100 的流程图

（2）递归法。对于这道题，还可以从另外一个角度考虑问题。首先，将求解式子 1+2+3+…+n 的值的过程定义为一个数学函数 f（n），这里 n 是正整数。通过分析，得到以下公式：

$$
\begin{cases} f(1)=1 & n=1 \quad (1) \\ f(n)=f(n-1)+n & n>1 \quad (2) \end{cases} \qquad (1.2.5)
$$

那么求解式子 1+2+3+…+100 的值的过程可以表示为 f（100）。由式（1.2.5）可得到如图 1.2.10 所示的基本思路。

通过以上的分析可以看出，要解决一个较大规模的问题（f(100)），可以归结为求解一个较小规模的问题（f(99)），而解决较小规模问题的方法和解决原问题的方法相同，但是规模不断缩小，当问题的规模缩小到一定程度时（n=1），当前问题得到确定的解（f(1)=1）。将 f(1)=1 代入求解 f(2) 的公式，则求得 f(2)的解，同理，f(3)、…、f(99)、f(100)也就得到解。

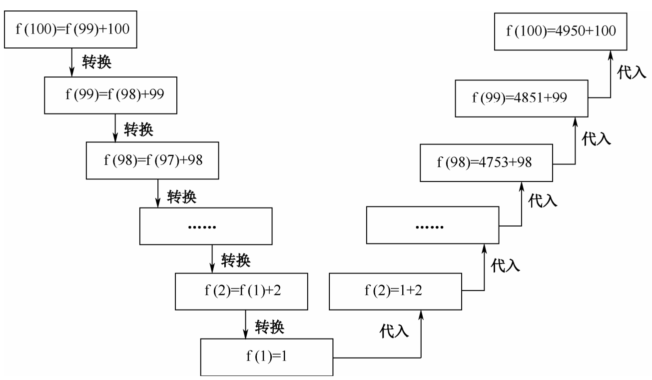

图 1.2.10　递归法求 1+2+3+…+100 的值

以上求解 f(100)的过程使用的就是递归方法。所谓递归算法，就是一种直接或间接地调用自身的算法。使用递归法解决问题时必须符合下面三个条件。

① 可以把一个问题转化为一个新问题，而这个新问题的解决方法与原问题的解决方法相同，只是所处理的对象不同。通常这种转化有一定的规律，典型的问题可以将这种规律归纳为一个公式（如上例中式（1.2.5）中的（2）），称为 “ 递归关系 ” 或 “ 递归形式 ”。

② 可以通过转化过程使问题最终得到解决。

③ 必定要有一个明确的递归结束条件，否则递归将会无休止地进行下去。如上例中式（1.2.5）中的（1）。

对于问题的描述或定义本身就是递归形式的问题，用递归方法可以编写出非常简单、直观的程序，但是这种程序在执行时需要开辟空间和不断地进行调用和返回操作，因此时空效率较低。

前面只是简单地讲述了递归法的基本思想，至于如何彻底理解该种设计算法的思想、步骤，进而编制出相应的程序，请读者参考函数一章。

☞举一反三：请大家仿照例 1.9 的方法使用递归法求解 5!。

4．分治法（Divide-and-Conquer）

使用计算机解决实际问题时，影响时间效率的主要因素是问题的规模。问题的规模越小，越容易直接求解，所需的计算时间也越少。若想直接解决一个规模较大的问题，有时是相当困难的。这时，可以试着将该问题分割成一些规模较小的相同问题，以便各个击破，分而治之。

【例 1.10】有 16 只相同大小和颜色的球，其中有一只是假球，假球比真球略轻。现在要求利用一台无砝码的天平，如何用最少的次数称出这只假球。

解题思路：大家常用的方法是先将这些球分成两只一组，共 8 组。每一次只称一组，最好情况是只称一次，最坏情况要称 8 次才可以找出那只假球。这种直接寻找的方法存在着相当大的偶然性。

试着改变一下策略：将全部球分成两组，每次比较两组球。会发现通过一次比较后，基于假球比真球略轻这一特点，完全可以舍弃全部是真球的一组球，选取与原有问题一致的另一半进行下一次比较，这样问题的规模就明显缩小，而且每一次比较的规模也成倍减少。

解答：具体比较过程如图 1.2.11 所示。

根据图 1.2.11 所示的求解过程，可以得到如下的结论：

① 问题的规模越大，也就是球的只数越多，使用以上的方法效果越显著；

② 解决此类问题的关键在于将规模较大的问题分割成了若干规模较小的问题；

③ 规模较小的问题与原问题的解决方法完全类似。

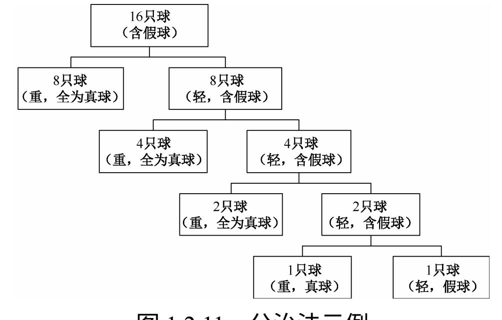

图 1.2.11　分治法示例

通常这种将一个难以直接解决的、规模较大的问题分割为一些解决方法与原问题相同，规模较小的问题，以便各个击破，分而治之的算法设计策略称为分治法。

一般来说，分治法比较适合求解具有如下 4 个特征的问题。

① 问题可以分解为若干个规模较小的相同问题。

② 当问题的规模缩小到一定的程度就能够很容易地解决问题。

③ 合并子问题的解可以得到求解问题的解。

④ 由求解问题所分解出的各个子问题是相互独立的。

一般可按如下步骤使用分治法设计算法：

① 分解：将要求解的问题划分为若干规模较小的同类问题。

② 求解：当子问题划分得足够小时，用较简单的方法求得子问题的解。

③ 合并：按照求解问题的要求，将子问题的解逐层合并，即可构成最终的解。

除了前面介绍的 4 种常用的算法设计的基本方法外，算法设计的策略还包括回溯法、贪心法、

模拟法、动态规划等。

自主学习：请有兴趣和余力的读者参考算法设计的书籍或上网查阅相关资料，自主学习回溯法、贪心法、模拟法、动态规划等算法设计策略。

1.3　程序与程序设计

1.3.1　程序与程序设计语言

解决实际问题的算法设计完成后，接下来就该使用某种高级程序设计语言来实现这一算法。算法的实现就是编写解决问题的程序，并运行程序得到结果。

人与人之间互相交流，应能够听懂彼此的语言；人与计算机交流，计算机应能够“听懂”人的语言。非常不幸的是，计算机“听不懂”人类的自然语言，因此，人类发明了各种称为“程序设计语言”的工具，以便与计算机交流。按照程序设计语言发展的过程，程序设计语言大致分为机器语言、汇编语言和高级语言。

1．机器语言（Machine Language）

每种计算机被设计、制造出来后，都能够执行一定数量的基本操作，如加、减、移位等，即机器指令（Instruction）。一种计算机所有指令的集合称为该计算机的“机器语言”。为了解决某一实际问题，从指令集合中选择适当的指令组成的指令序列，称为“机器语言程序”。

由于机器语言是由“与生俱来”的机器指令组成的，因此它是唯一能够被计算机直接识别和执行的程序设计语言。

【例 1.11】用 Intel 8086/8088CPU 的指令系统编写 3+8 的机器语言程序段如下：

```
10110000  00000011          表示将数3送到累加器AL中
00000100  00001000          表示把AL中的数3与8相加的结果保留在AL中
```

由上例可以看出，机器语言编写的程序晦涩难懂，就像“天书”一样，可阅读性、可维护性差，编写起来效率低下；由于不同种类的计算机指令系统不同，因此其机器语言也各自不同，移植性差，是一种面向机器的语言。虽然机器语言缺点明显，但是其优点也十分明显：机器语言编写的程序不需要翻译，计算机就能够直接识别、执行，所占内存少，执行效率高。

2．汇编语言（Assembly Language）

鉴于机器语言的种种缺点，计算机科学家用人们容易记忆的符号（如英文单词或其缩写），即所谓的“助记符”来表示机器语言中的指令，如用 ADD 表示加、SUB 表示减、MOV 表示数据传递等。这种用助记符表示指令的程序设计语言就是汇编语言，用汇编语言编写的程序称为汇编语言源程序。

【例 1.12】用 Intel 8086/8088 的汇编语言编写 3+8 的汇编语言源程序段如下：

```
MOV   AL , 3          表示将数3送到累加器AL中
ADD   AL , 8          表示把AL中的数3与8相加的结果保留在AL中
```

由上例可以看出，与机器语言相比，汇编语言在可阅读性、可维护性方面有一定的改善，同时保持了占用内存少，执行效率高的特点，比较适合实时性要求较高的场合，如系统软件的编写。但是汇编语言依然是一种面向机器的语言，其移植性、可维护性还是很差，而且汇编语言源程序必须翻译成机器语言后才能够被计算机执行。

3．高级语言（High-level Language）

为了进一步提高程序设计生产率和程序的可阅读性，使程序设计语言更接近于自然语言或数学公式，并力图使其脱离具体机器的指令系统，提高可移植性，在 20 世纪 50 年代出现了高级语言。高级语言是一类语言的统称，而不是特指某种具体的语言。C、C++、Java、Visual Basic 是

目前比较流行的高级语言。

【例 1.13】用 C 语言编写 3+8 的 C 语言源程序段如下：

```
int  a ;                    /* 定义整型变量a */
a = 3 + 8 ;                 /* 将3+8的和赋值给变量a */
```

使用高级语言进行程序设计，程序员不用直接与计算机的硬件打交道，不必掌握机器的指令系统，学习门槛较低，可使程序员将主要精力集中在算法的设计和程序设计上，可大大提高编写程序的效率。由于高级语言更接近于自然语言或数学公式，不依赖于具体的计算机硬件，因此与机器语言和汇编语言相比，在可阅读性、可维护性、可移植性方面均有了极大的提高。用高级语言编写的程序必须翻译成机器语言后才能够被计算机执行。为了保持程序的通用性，在翻译过程中可能会衍生出一些完成常规性功能的代码，如引用数组元素时要判别下标是否越界，因此，用高级语言编写的程序其执行效率比用机器语言或汇编语言编写的、用于完成相同功能的程序要低。大部分高级语言用于编写对实时性要求不高的场合，如应用软件的编写。

1.3.2 程序设计语言处理过程

计算机只能直接识别和执行用机器语言编制的程序，为使计算机能识别用汇编语言和高级语言编制的程序，要有一套预先编制好的起翻译作用的翻译程序，把它们翻译成机器语言程序，这个翻译程序被称为语言处理程序。被翻译的原始程序（用汇编语言或高级语言编制而成）称为源程序，翻译后生成的程序称为目标程序。

1．汇编程序

语言处理程序翻译汇编语言源程序及可执行程序执行的过程如图 1.3.1 所示。

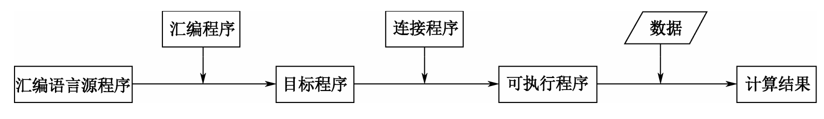

图 1.3.1　汇编语言处理过程

在图 1.3.1 中，汇编语言处理过程包括汇编、连接和执行 3 个阶段。

（1）汇编（Assembly）。将汇编语言源程序翻译成目标程序的过程称为汇编，而完成汇编任务的语言处理程序称为汇编程序或汇编器（Assembler）。

（2）连接（Link）。虽然图中的目标程序已经是机器语言程序了，但是它还不能被直接执行，还需要把目标程序与库文件（提供一些基本功能，如基本输入/输出、字符串处理等）或其他目标程序（由自己或他人编写，完成实际问题的部分功能）组装在一起，才能形成计算机可以执行的程序，即可执行程序，这一过程称为连接，完成连接任务的工具称为连接程序或连接器（Linker）。

（3）执行（Execute，Run）。操作系统将连接后生成的可执行程序装入内存后开始执行，在这期间一般需要输入其要处理的数据，执行完成后得到计算的结果。

可执行程序可以脱离汇编程序、连接程序和源程序独立存在并反复执行，只有源程序修改后，才需要重新汇编和连接。

2．高级语言翻译程序

将高级语言源程序翻译成目标程序的工具称为高级语言翻译程序，其翻译的方式有两种，即编译（Compilation）和解释（Interpretation）。完成编译功能的程序称为编译程序或编译器（Compiler），完成解释任务的程序称为解释程序或解释器（Interpreter）。

（1）编译程序。编译方式的高级语言处理过程与汇编语言处理过程基本相同，如图 1.3.2 所示，其具体过程不再赘述。

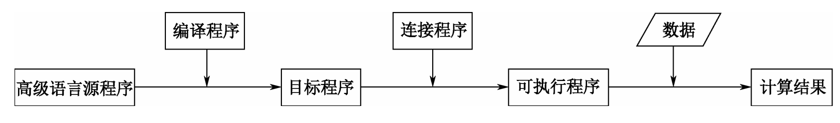

图 1.3.2　高级语言编译处理过程

高级语言的编译过程类似于“笔译”，翻译过程会产生目标程序，连接后会生成可执行程序，可执行程序可以脱离编译程序、连接程序和源程序独立存在并反复执行，只有源程序修改后，才需要重新编译和连接。

（2）解释程序。如图 1.3.3 所示，使用解释方式翻译高级语言源程序时，解释程序对源程序进行逐句解释、分析，计算机逐句执行，并不产生目标程序和可执行程序，整个过程类似于“同声传译”。

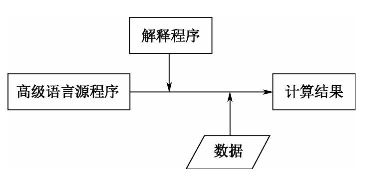

图 1.3.3　高级语言解释处理过程

解释方式处理高级语言源程序时，不生成目标程序和可执行程序，程序的执行不能脱离解释程序和源程序。

（3）解释方式与编译方式的区别。在处理高级语言时，有些高级语言使用解释方式，如 Basic、Python 语言等；而另外一些语言使用编译方式，如 C、C++、Pascal、FORTRAN 语言等。不同的语言，其编译或解释程序不同，彼此不能替代。两种翻译方式的比较如表 1.3.1 所示。

表 1.3.1　解释方式与编译方式的比较

比 较 项 目	解 释 方 式	编 译 方 式
类比	同声传译	笔译
是否生成目标程序	否	是
是否生成可执行程序	否	是
执行过程是否可脱离翻译程序	否	是
执行过程是否可脱离源程序	否	是
执行效率	较低	较高
便于跨平台	是	否
其他	适合于初学者	—

（4）其他处理方式。除了传统的编译和解释两种高级程序设计语言处理方式外，为了实现跨平台、跨语言等特点，还出现了既不是纯粹的编译型，也不是纯粹解释型的翻译方式，其中 Java 和.NET 技术是比较典型的代表。

① Java 语言的处理机制。谈到 Java，大家可能马上想起有关它的一句名言：Write once，run anywhere（一次编写，到处运行）。这句话非常贴切地描述了 Java 语言最重要的一个的特点，即跨平台的特性。那么，Java 是如何实现跨平台的呢？

Java 虚拟机（Java Virtual Machine，JVM）是 Java 语言跨平台的关键所在。为了说明 JVM 的工作原理，先来看一下如图 1.3.4 所示的 Java 程序运行的一般过程。

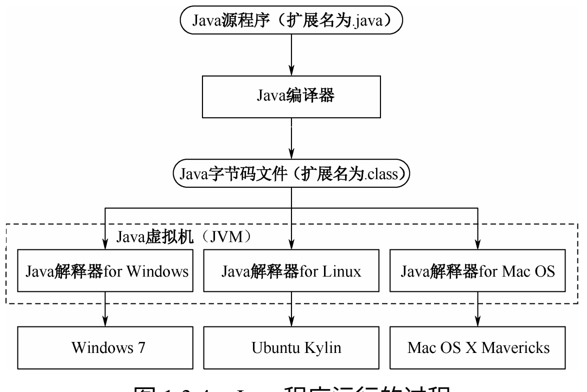

图 1.3.4　Java 程序运行的过程

程序员使用 Java 语言编写 Java 源程序文件（扩展名为.java），然后由 Java 编译器将 Java 源程序编译为与平台无关的、可在 Java 虚拟机上解释执行的 Java 字节码文件（扩展名为.class）。一台 Java 虚拟机就是一个 Java 解释器，它负责解释执行字节码文件。不同平台上的 JVM 向编译器