第 3 章　程序设计方法及程序算法

想要编写一个高质量、高效率的程序，必须要设计一个优秀的程序算法。在现代语言程序设计中，算法是一个程序的核心和灵魂之所在，作为一名合格的程序员，对程序设计方法和程序算法要有较好的了解和掌握。

学习目标：

- 了解程序设计方法的基本概念
- 熟悉算法的概念和常用基本算法的编写
- 掌握传统流程图编写算法的方法
- 掌握 N-S 流程图编写算法的方法

3.1　程序设计方法简介

在现代社会中，随着科学技术的迅猛发展，越来越多的工程方法都要由程序来完成，比如数值计算、图像处理、人工智能、语音识别等。那么究竟什么是程序呢？简单来说，程序就是程序员利用计算机语言编写的、能够完成特定功能和任务的计算机代码，程序编写的目的是为了提高工程应用的自动化水平，简化人的操作，提高工作效率，促进社会发展。著名计算机科学家沃思（Niklaus Wirth）提出了如下公式，可以作为程序设计的最好解释。

程序=数据结构+算法

此公式可以说是计算机程序设计的根本灵魂之所在，它深刻地诠释出了计算机程序设计的本质。

随着现代软件行业的不断发展，作为一名程序员，除了对数据结构和算法有深刻的理解之外，还要选择适当的开发工具进行程序的开发和编写，这样能够提高工作效率。我们在第 2 章中讲到的 Intel Visual FORTRAN 和 Visual Studio 2010 集成的工具就是一个高效的 FORTRAN 语言开发环境，利用这些可以极大地提高开发效率，达到事半功倍的效果。

一名优秀的程序员不仅要有扎实的基础知识和熟练的开发技巧，还要有专业化的程序设计方法。因此，现代程序设计方法定义还可以扩展为：

程序 = 算法+数据结构+开发工具+程序设计方法

现代软件工程行业对于软件设计开发主要基于两种方法。

1．结构化程序设计方法

结构化程序设计由迪克斯特拉（E.W.Dijkstra）在 1969 年提出，它是以模块化设计为中心，

将待开发的软件系统划分为若干个相互独立的模块，这样使每一个模块的工作变得单纯而明确，为设计一些较大的软件打下了良好的基础。

结构化程序设计的原则是自顶向下、逐步细化、模块化设计及结构化编码，这样做的优点就是，由于所有的模块相互独立，不会受到其他模块的干扰，从而提高了程序设计的独立性，并且可以将一系列复杂的软件设计改为相互独立的、简单的软件设计方式。

2．面向对象程序设计方法

面向对象（Object-Oriented，OO）是 20 世纪 90 年代开始形成的，是现代软件开发方法的主流方法，它是一种适用于设计、开发各类软件的范型。它将软件看成是一个由对象组成的社会，这些对象具有足够的智能，能理解从其他对象接受的信息，并以适当的行为做出响应；允许低层对象从高层对象继承属性和行为。通过这样的设计思想和方法，将所模拟的现实世界中的事物直接映射到软件系统的解空间。

与传统的结构化程序设计相比，面向对象程序设计吸取了结构化程序设计的一切优点（自顶向下、逐步求精的设计原则）。而两者之间的最大差别如下：

面向对象程序采用数据抽象和信息隐藏技术，使组成类的数据和操作不可分割，避免了结构式程序由于数据和过程分离引起的弊病。这正是面向对象设计方法和结构化程序设计方法的根本差异，也是面向对象设计方法的精华所在。

面向对象程序是由类定义、对象（类实例）和对象之间的动态联系组成的。而结构式程序是由结构化的数据、过程的定义以及调用过程处理相应的数据组成的。

本书主要讲解 FORTRAN 程序设计的基本方法，对于程序设计思想的方法只做一个简要的介绍，读者想要深入学习可以参考有关书籍。

3.2　算法的概念及特性

本节中我们对算法的基本概念和算法的特性进行详细的介绍，从而加深读者对算法的理解，为后面学习算法的表示方法打下基础。

3.2.1　算法的概念

正如我们做某一件事情需要按照一定的步骤进行一样，那么用计算机程序解决一个问题所采取的方法和步骤就是算法。算法是程序设计必不可少的部分，也是程序设计的核心内容，算法设计的好坏直接影响到程序的执行效率，并且不好的算法可能会导致问题求解失败。因此，本书将对程序算法做详细的介绍。

在数学和计算机科学之中，算法（Algorithm）为一个计算的具体步骤，常用于计算、数据处理和自动推理。算法应包含清晰定义的指令用于计算和解决问题。一个完整的算法能够对具有一定规范的输入，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。

注意：“计算方法”（computational method）和“算法”（algorithm）是两个不同的概念。前者指的是求数值解的近似方法，后者是指解决问题的一步一步的过程。在解一个数值计算问题时，除了要选择合适的计算方法外，还要根据这个计算方法写出如何让计算机一步一步执行以求解的算法。

对于计算机外行来说，他们可以使用别人已设计好的现成算法，只需根据算法的要求给以必要的输入，就能得到输出的结果。对他们来说，算法如同一个“黑箱子”一样，他们可以不了解“黑箱子”中的结构，只是从外部特性上了解算法的作用，即可方便地使用算法。但对于程序设计人员来说，必须会设计算法，并且根据算法编写程序。

对同一个问题，可以有不同的解题方法和步骤。例如，求 1+2+3+…+100，可以先进行 1+2，再加 3，再加 4，一直加到 100，也可采取 100+(1+99)+(2+98)+…+(49+51)+50＝100+50+49×100＝5050，当然还可以有其他方法进行运算。不同的算法有质量优劣之分。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。一般来说，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。

算法可以分为计算型算法和非计算型算法。

计算型算法的主要目的是解决某一数值问题，对问题求解出数值解，如求方程的根、计算函数的微分、计算圆形的面积、开平方等。非计算型算法的范围比较广泛，如各种人事管理系统、信息检索系统、语音识别系统等，但是其成熟度不如计算型算法高，这是因为计算型算法可以用现成的数值分析的方法进行程序的编写，写成函数库供不同的用户来调用，比如 FORTRAN 语言中的 IMSL 库就是很方便的数值计算库，这为用户的操作提供了极大的方便。

而非计算型算法的用户需要千差万别，想要完成某一具体任务需要重新设计方法并编写程序，所使用的算法各不相同，比如图书馆信息管理系统和行车高度系统需要相差极大，所以这也导致算法的开发各异，其成熟度不高。本书不可能将所有算法都罗列出来，只能通过一些典型算法的介绍，使读者对算法能够有一个较深入的理解和掌握，为将来设计算法、编写程序打下一个坚实的基础。

3.2.2 简单算法举例

在本节中，我们对一些简单算法进行举例说明，从而使读者对算法有一个深入的理解，请读者仔细阅读下面讲解的算法。

例 3.1：求 1×3×5×7×9×11。

用原始的方法进行求解的步骤如下：

1）求 1×3 得到结果 3；

2）将得到的结果 3 乘以 5，得到结果 15；

3）将结果 15 乘以 7，得到 105；

4）将结果 105 乘以 9，得到 945；

5）将结果 945 乘以 11，得到 10395，这就是最终的结果。

从上述步骤我们可以看出，这种计算方法可以得到正确的结果，但是却比较麻烦。如果要求的是 1×3×5×7×9×…×100，那么将导致计算非常繁琐。因此，我们需要对这种方法进行改进，得到一个通用的方法来进行计算。

可以如下考虑：因为乘法计算从第二步开始每一步都可以看成将上一步得到的新结果值与下一个数进行相乘，那么我们可设置两个变量，第一个变量存放上一步得到的结果（在乘法运算中被称为被乘数），第二个变量存放待乘的数（乘法运算中的乘数），经过不断的循环最终将结果计算出来。现设 x 为被乘数，y 为乘数，将算法改写如下（其中，S 代表 Step，表示步骤的意思）。

S1：使 x=1；

S2：使 y=3；

S3：使 x*y；将得到的结果存放到 x 中，可表示为：x*y→x；

S4：使 y 的值加 2，即：y+2→y；

S5：如果 y 不大于 11，则返回重新执行步骤 S3 及其后面的步骤 S4、S5；如果 y 大于 11 则算法结束，最后得到的 x 值就是 1×3×5×7×9×11 的结果。

从上面的算法可以看出，改进之后的算法比较简洁，请读者仔细分析。

如果将题目的要求改为 1+3+5+7+9+11，那么只需要进行少量的改动就可以实现问题的求解：

S1：使 x=1；

S2：使 y=3；

S3：x+y→x；

S4：y+2→y；

S5：如果 y≤11，则继续，否则结束。

从上面可以看出，改进后的算法通用性和灵活性都要优于原来的算法。步骤 S3 到 S5 组成一个循环，这在计算机中是可以轻而易举地实现的，所有高级语言都有专门用于循环的语句，可以方便地实现此算法。

例 3.2：对一个大于或等于 3 的正整数，判断它是不是一个素数。

所谓素数，是指除 1 和该数本身之外，不能被其他任何整数整除的数。例如，13 是素数，因为它不能被 2，3，4，…，12 整除。

判断一个数 n（n≥3）是否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。算法可以表示如下：

S1：输入 N 的值；

S2：i=2；

S3：N 被 i 除；

S5：如果余数为 0，表示 N 能被 i 整除，则打印“N 不是素数”；算法结束；否则继续；

S6：i=i+1；

S7：如果 i≤N−1，返回到 S3；否则打印“N 是素数”，然后结束。

实际上，n 不必被 2 到（n−1）的整数除，只需被 2 ~ n/2 间整数除即可，甚至只需被 2 ~ $\sqrt{n}$ 之间的整数除即可。例如，判断 13 是否素数，只需将 13 被 2、3 除即可，如都除不尽，n 必为素数。因此，步骤 S6 可改为：

S6：如果 I≤$\sqrt{N}$，返回 S3；否则算法结束。

例 3.3：输入两个数，求出其中最大值。

S1：输入第一个数，将其存入变量 a；

S2：输入第二个数，将其存入变量 b；

S3：如果 a>b，输出“第一个数最大”；如果 a<b，输出“第二个数最大”；如果 a=b，输出“两个数一样大!”。

例 3.4：求两个自然数的最大公约数，设两个变量为 M 和 N。

S1：如果 M<N，则交换 M 和 N；

S2：M 被 N 除，得到余数 R；

S3：如果 R＝0，则 N 即为“最大公约数”，否则转到下一步；

S4：将 N 赋值给 M，将 R 赋值给 N，转到 S1。

在这个算法中，要不断地进行循环和判断，当余数为 0 时，就求解出最大公约数。

例 3.5：根据学生成绩的多少可分为不及格、及格、中等、良好和优秀 5 个等级。通常规定如下：

1）不及格：成绩小于 60 分；

2）及格：成绩大于等于 60 分小于 70 分；

3）中等：成绩大于等于 70 分小于 80 分；

4）良好：成绩大于等于 80 分小于 90 分；

5）优秀：成绩大于等于 90 分小于等于 100 分。

从键盘上接收一个成绩，输出成绩的级别。

求解的步骤如下：

S1：从键盘接收一个正实数 r；

S2：若 r<60，则输出“不及格”并转 S7；

S3：若 60≤r<70，则输出“及格”并转 S7；

S4：若 70≤r<80，则输出“中等”并转 S7；

S5：若 80≤r<90，则输出“良好”并转 S7；

S6：若 90≤r≤100，则输出“优秀”并转 S7；

S7：算法结束。

说明：数据的合法性没考虑，实际应用中需对输入的数据进行合法性检查。

例 3.6：写一个算法，输入北京市 2012 年 9 月中每天的气温，求出这个月的平均气温并输出。

依题意，先给出求解的步骤如下。

S1：定义一个 30 个元素的实型数组 T，以及总的温度 SumTemperature、平均温度 AvgTemperature；

S2：借助循环，从键盘接收每天的气温；
S3：借助循环，统计这个月的温度总和；
S4：用温度总和 SumTemperature 除以 30 天，得该月的平均气温 AvgTemperature；
S5：输出平均气温 AvgTemperature；
S6：算法结束。
对 S2：进一步细化为：
S21：循环变量初始化 day=1；
S22：接收一个温度，并存入相应温度数组元素 T[day]中；
S23：day=day+1；
S24：如果 day≤30，则转 S22；
S25：输入结束。
对 S3 进一步细化为：
S31：循环变量初始化 day=1；
S32：从温度数组中读取 T[day]元素，SumTemperature=Sun Temperature+T[day]；
S33：day=day+1；
S34：如果 day 不大于 30，则转 S32；
S35：统计结束。
说明：数据的合法性没考虑，实际应用中需对输入的数据进行合法性检查。

例 3.7：判断 1900 年 ~ 2000 年中的每一年是否是闰年，将结果输出。

闰年的求解原则是：
1）能被 4 整除，但是不能被 100 整除的年份是闰年，如 1964 年、1980 年、1988 年等；
2）能够被 100 整除，又能被 400 整除的年份也是闰年，如 1200 年、2000 年等。
该算法可表示如下：
首先用变量 year 表示所要检测的年份，步骤如下：
S1：1900->year；
S2：若 year 不能被 4 整除，就直接输出“不是闰年”，然后转到步骤 S6；
S3：若 year 能被 4 整除，不能被 100 整除，则输出“是闰年”，然后转到步骤 S6；
S4：若 year 能被 100 整除，又能被 400 整除，则输出“是闰年”，然后转到步骤 S6；
S5：输出“不是闰年”；
S6：year+1->year；
S7：当 year≤2000 时，转 S2 继续执行，否则算法结束。

在此步骤中，我们进行了多次判断。首先判断 year 能否被 4 整除，如果不能的话，那么 year 肯定不是闰年。如果 year 能够被 4 整除，还要进行进一步判断，还要判断它能否被 100 整除，如果不能被 100 整除，那么肯定是闰年（比如 1980 年）。如能被 100 整除还不能完全判定是否是闰年，还要进一步被 400 整除，如果不能被 400 整除，则肯定不是闰年，如果能被 400 整除，则肯定是闰年。

在此算法中，每进行一步，判断范围就缩小了一些，直到执行到 S5 时，只可能出现“不是闰年”的情况，这时直接输出“不是闰年”即可。

我们在实际编程设计算法的过程中，要首先对题目的要求进行仔细的分析，判断题目都需要进行哪些判断、循环，如何一步一步地进行范围判定及数值计算。很多问题的解决都需要循序渐进，不断将判定条件缩小，直到最终求解出答案。

3.2.3 算法的特性

一般地，一个有效的算法应该有如下一些基本特征。

（1）有穷性

一个算法必须总是在执行有限步骤之后结束，即一个算法必须包含有限个操作步骤，而不是无限的。例如在例 3.7 中，如果将 S7 改成当 year>0，那么循环将永远执行下去，永不终止，这个算法在实际中是无效的，也是没有任何实际意义的，因为程序无法结束，答案就永远不会计算出来。

另外，在实际设计算法中还要注意一点，那就是设计的算法不仅是有穷的，而且计算时间也要考虑一下。一般的有特殊要求的行业除外（进行有限元分析、液体力学的计算等），大部分的算法设计都要考虑计算时间的长短。如果一个算法是有穷的，但是需要计算 100 年，那么这个算法在实际中也是无意义的，这超出了一个合理的范围，因此不能算作一个有效的算法。

（2）确定性

算法中的每一个步骤应当是确定的，而不应当是含糊的、模棱两可的，即程序员在阅读算法时不产生二义性。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。

在编写程序中，程序员必须利用程序代码，清楚明确地告知计算机要做什么事情，就像在实际生活中，如果上级领导给下级人员下一个“购买书”的指令一样，那么下级人员并不能确定到底要买什么书、买多少本、从哪买之类的问题，这样必然会产生很多问题。计算机中的算法也同样如此，如果程序员不能在设计算法的时候确定一个明确的步骤，那么计算机是不能执行的。比如在例 3.5 中，如果我们不把所有的情况（如分数在不同的区间段的输出情况）都罗列清楚的话，当从键盘输入一个数据的时候，如果在没有给出确定的判断条件下，那么程序就会出错，算法无法向下进行，如图 3-1 所示。

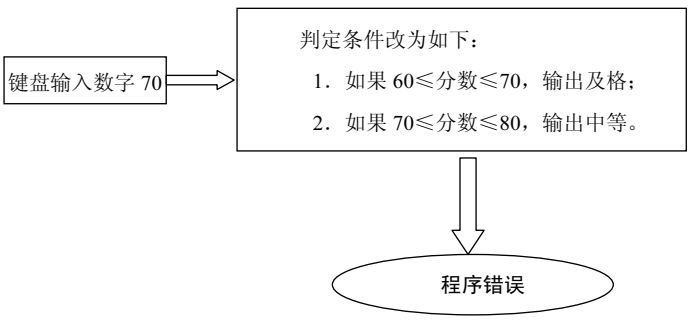

图 3-1　错误的程序算法

（3）有零个或多个输入

所谓输入是指在执行算法时，从键盘或其他设备中获取的信息。一个有效的算法可以有多个输入，也可以没有输入。例如在例 3.6 中，我们就需要首先接收 30 个温度信息，这就是有多个输入的情况，如果不输入温度信息的话，后续的计算就无法进行。而在例 3.5 中，就只需要一个输入信息（学生的成绩）即可，只有按照程序的要求进行输入才可以得到正确的求解答案。

（4）有一个或多个输出

编程的目的是为了求解问题，“解”就是输出。一个算法可以有一个或多个输出，无输出的算法是没有意义的。例如在上面举的每一个例子中，我们都在有确定结果的情况下输出了一个结论，这样的程序才能真正解决问题。

注意：并不是用计算机打印的输出才算是真正的输出，一个算法得到的结果就是算法的输出。

（5）可行性

算法做的每一步都能有效地被执行，并且能得到确定的结果。比如在计算除法运算时，如果 y=0，那么 x/y 是不能被有效执行的。

在实际应用中，我们编写算法的目的就是为了解决某个特定的问题，那么可以将设计好的算法进行封装，使之有必要的输入和输出即可。就如同一个黑匣子一样，使用人员不必要知道其内部的具体结构，只需要按照要求输入一定的数据，算法就会输出结果，如图 3-2 所示。

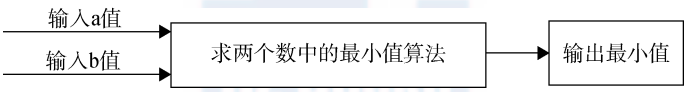

图 3-2　求最小值算法

3.3　算法的表示方法

有效、简洁地描述一个计算机求解过程，称为算法的表示。为了表示一个算法，可以用不同的方法。常用的有：自然语言、传统流程图、结构化流程图、伪代码、PAD 图等，这里我们对这几种表示方法进行详细的介绍。

3.3.1　用自然语言表示算法

自然语言描述就是用人们日常使用的语言，如汉语、英语或其他语言来描述算法。如前面 3.2.2 节中的所有例子就都是用自然语言来描述算法的求解过程的。用自然语言来描述和表示算法的优点是通俗易懂，缺点是文字过于冗长，容易出现歧义性。比如这句话“我们这个教室里有 10 个计算机学院的学生”，这句话既可以指教室里只有 10 个学生，也可以指教室里可能有很多个学生，其中计算机学院的学生有 10 个，此处使用自然语言就出现了语义的歧义。此外，用自然语言来表示算法还不能较好地表示循环和分支，使得算法描述起来很不方便，因此，除

了简单的问题外，一般不用自然语言描述算法。

3.3.2 用流程图表示算法

用图表示的算法就是流程图。流程图是用一些图框来表示各种类型的操作，在框内写出各个步骤，然后用带箭头的线把它们连接起来，以表示执行的先后顺序。用图形表示算法，直观形象，易于理解。

美国国家标准化协会 ANSI（American National Standard Institute）曾规定了一些常用的流程图符号，为世界各国程序工作者普遍采用，如图 3-3 所示。下面对图 3-3 中的各种图形的使用进行简要的说明。

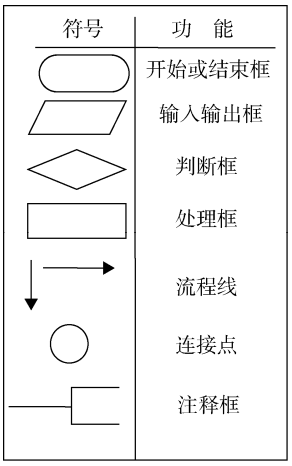

图 3-3 常用流程图符号表示

1）处理框（矩形框），表示一般的处理功能。

2）判断框（菱形框），表示对一个给定的条件进行判断，根据给定的条件是否成立决定如何执行其后的操作。它有一个入口，两个出口。

3）输入输出框（平行四边形框），表示输入或输出操作。

4）起止框（圆弧形框），表示流程开始或结束。

5）连接点（圆圈），用于将画在不同地方的流程线连接起来。用连接点可以避免流程线的交叉或过长，使流程图清晰。

6）流程线（指向线），表示流程的路径和方向。

7）注释框，是为了对流程图中某些框的操作做必要的补充说明，以帮助阅读流程图的人更好地理解流程图的作用。它不是流程图中必要的部分，不反映流程和操作。

程序框图表示程序内各步骤的内容以及它们的关系和执行的顺序，它说明了程序的逻辑结构。框图应该足够详细，以便可以按照它顺利地写出程序，而不必在编写时临时构思，甚至出现逻辑错误。流程图不仅可以指导编写程序，而且可以在调试程序中用来检查程序的正确性。如果框图是正确的而结果不对，则按照框图逐步检查程序是很容易发现错误的。流程图还能作为程序说明书的一部分提供给别人，以便帮助别人理解程序员编写程序的思路和结构。

下面我们将 3.2.2 节讲的例子用流程图重新表示一下，以帮助读者清楚地理解流程图的用法。

例 3.8：将例 3.1 中求 1×3×5×7×9×11 的算法用流程图表示，如图 3-4 所示。

例 3.9：将例 3.2 中的算法用流程图表示，如图 3-5 所示。

例 3.10：将例 3.3 中求两个数最大值的算法用流程图表示，如图 3-6 所示。

例 3.11：将例 3.4 中求两个数的最大公约数的算法用流程图表示，如图 3-7 所示。

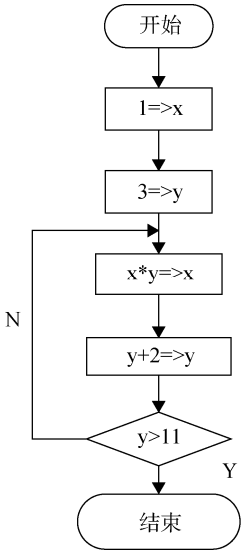

图 3-4　例 3.8 流程图

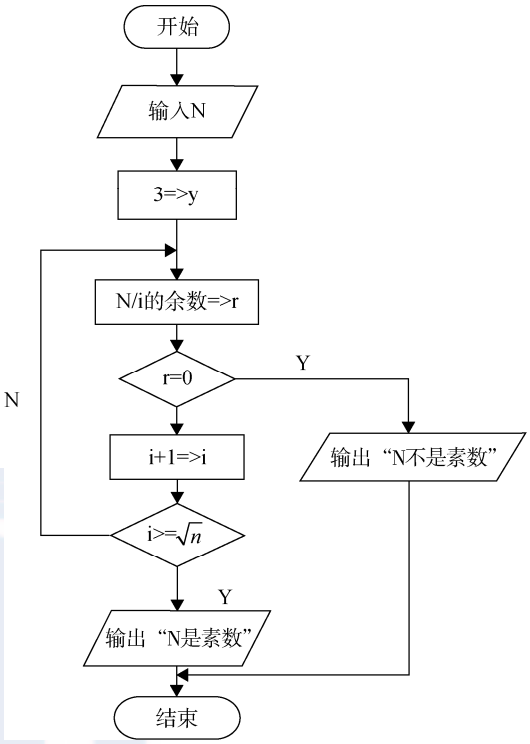

图 3-5　例 3.9 流程图

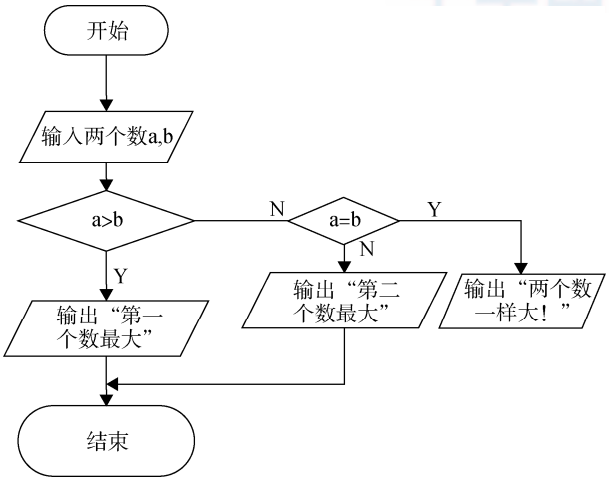

图 3-6　例 3.10 流程图

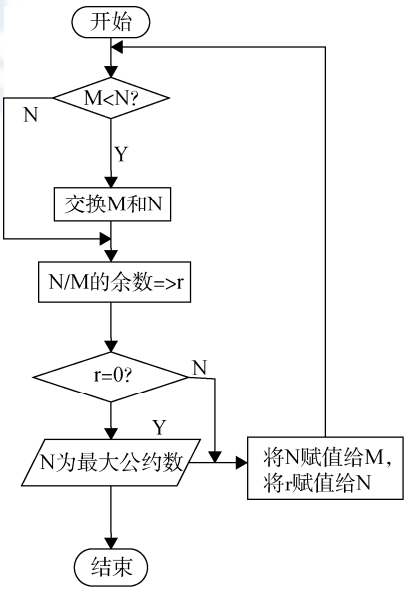

图 3-7　例 3.11 流程图

例 3.12：将例 3.5 中的算法用流程图表示，如图 3-8 所示。

例 3.13：将例 3.6 中求 2012 年 9 月平均气温的算法用流程图表示，如图 3-9 所示。

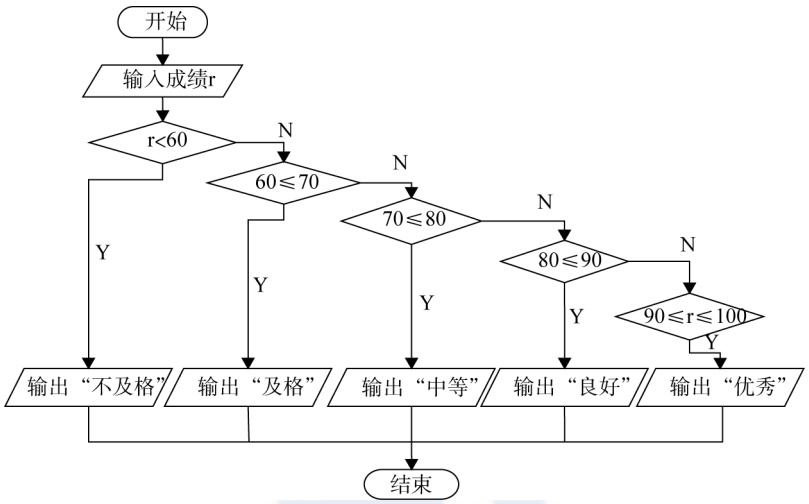

图 3-8　例 3.12 流程图

例 3.14：将例 3.7 中判定闰年的算法用流程图表示，如图 3-10 所示。

通过以上几个例子我们可以看出，用流程图表示算法是一种很方便的工具。下面我们对流程图的用法进行一下简要的总结。

一个流程图必须包括如下几个部分：

1）表示不同操作的框，如处理框、判断框等。

2）带箭头的流程线，这一点尤其重要，因为箭头线表示程序的执行过程，如果没有画出的话，那么很难判断程序如何执行。

3）框内外的文字、符号说明，这样将使程序更直观易懂。

4）开始和结束框，如果没有这两个框的话，无法确定程序从哪里开始及结束。

用流程图表示程序算法直观形象、易于理解，能够清楚地表达各个步骤之间的逻辑关系，因此，在很多书籍和论文中得到了广泛的应用。但是利用流程图也有其相应的缺点及不足之处，即占用篇幅较多，画流程图费时费力等。那么为了解决这个问题，改进之后的 N-S 流程图逐渐取代了传统的流程图，N-S 流程图也得到了更广泛的应用。但是，作为一个合格的计算机程序员，请读者一定要掌握这种传统流程图的表示方法，做到会看会用，这样可以看懂别人画的程序，同时也能具备画传统流程图的能力。

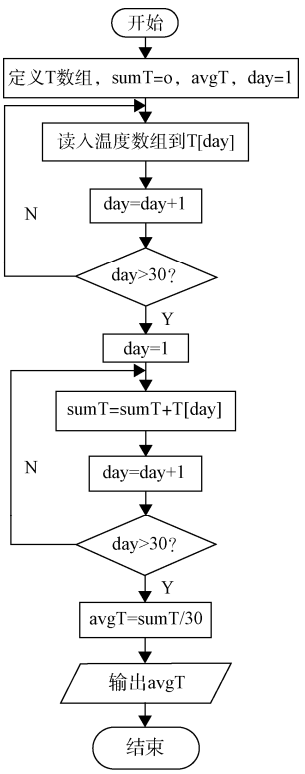

图 3-9　例 3.13 流程图

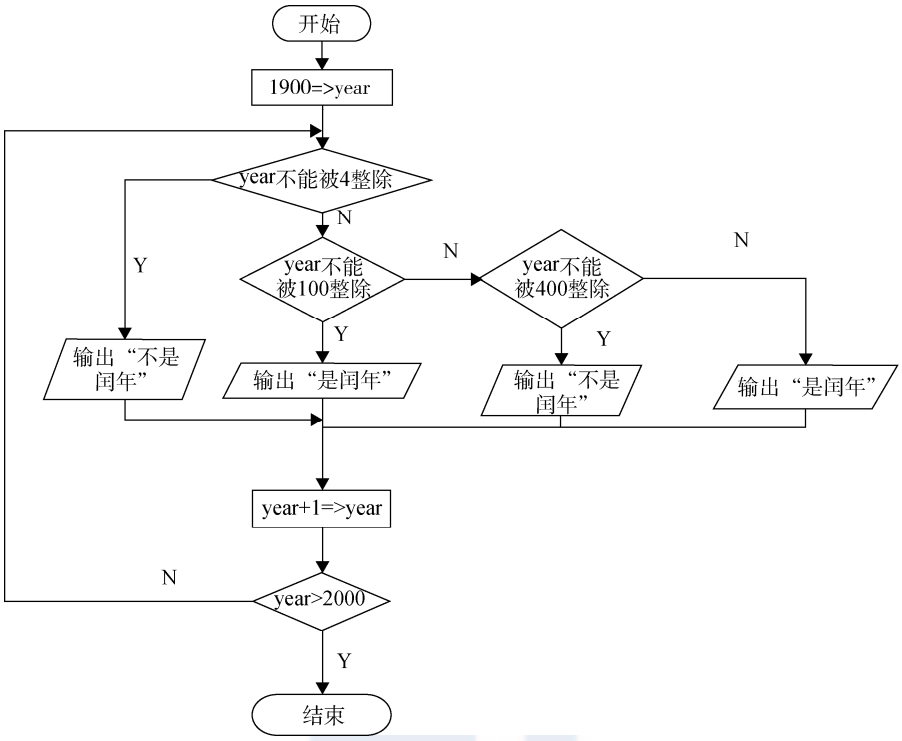

图 3-10　例 3.14 流程图

3.3.3　三种基本结构

传统的流程图用流程线指出各框的执行顺序，对流程线的使用没有严格限制。因此，使用者可以不受限制地使流程随意转来转去，使流程图变得毫无规律，但当程序较为复杂的时候，阅读者要花很大精力去追踪流程，这就使人难以理解算法的逻辑。如果我们写出的算法能限制流程的无规律任意转向，那样程序阅读起来就很方便，不会有任何困难，只需从头到尾顺序地看下去即可。

为了提高算法的质量，使算法的设计和阅读方便，必须限制箭头的滥用，即不允许无规律地使流程乱转向，只能按顺序地进行下去。但是，在实际开发中，计算机算法难免会包含一些分支和循环，而不可能全部由一个个框顺序组成。为了解决这个问题，人们设想，如果规定出几种基本结构，然后由这些基本结构按一定规律组成一个算法结构，就如同用一些基本预制构件来搭房屋一样，整个算法的结构是由上而下地将各个基本结构顺序排列起来的。1966 年，Bohra 和 Jacoplni 提出了顺序结构、选择结构和循环结构三种基本结构，这三种基本结构就构成了一个良好算法的基本结构单元。

（1）顺序结构

顺序结构就是从头到尾一次执行每一个语句，严格按照语句的书写顺序从上到下、从左到右执行，如图 3-11 所示。虚线框内是一个顺序结构，A 和 B 两个框内是按照顺序执行的，即执行完 A 框中的内容之后就执行 B 框中的内容。

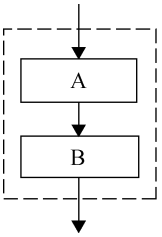

图 3-11　顺序结构

（2）选择结构

选择结构又称为选取结构或者分支结构，是根据不同的条件执行不同的语句或者语句体。选择结构可以分为：单分支、二分支和多分支结构。例如：如果明天放假，我就出去旅游，否则我就去学校上课。显然，多分支结构执行时，一次只会执行一个；但不同时刻执行，会执行到不同的分支，比如前面的例题中查询学生成绩属于哪个区间的就是一个多分支结构，因为输入不同的成绩会查找相应的区间段执行相应的代码。如图 3-12 所示就是双分支和单分支的情况。

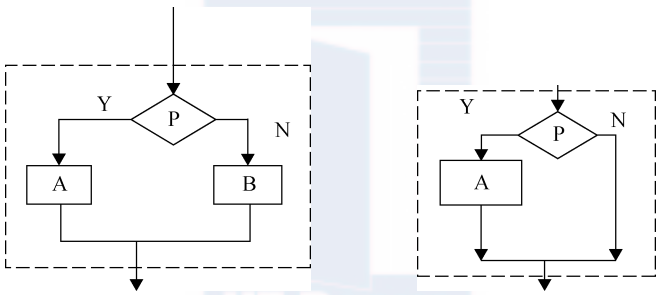

图 3-12　选择结构

在图 3-12 中，先对判断框 P 中的条件进行判断，后根据判断值的真假，选择相应的处理框执行。对于第一个双分支结构来说，如果 P 条件成立，那么就会执行 A 框内的内容；如果 P 不成立，就会执行 B 框内的内容。对于第二个单分支结构来说，如果 P 条件成立，就会执行 A 框的内容；如果 P 不成立，那么什么都不执行，继续向下执行相应的语句。

注意：选择结构无论条件是否成立，都只能执行其中一个分支的内容，不可能既执行 A 分支又执行 B 分支，这样的程序是错误的。选择结构可以有一个框中的内容是空的，即不执行任何操作，接着进行选择结构后面的操作。

（3）循环结构

循环结构就是重复地执行语句或者语句体，达到重复执行一类操作的目的。例如：每天都有 24 个小时，时钟每天 0 点的时候都会重新走一个循环，程序也有循环结构。循环可以分为当（while）型循环和直到（until）型循环两种。

- 当（while）型循环：如图 3-13a 所示。当给定的条件 p 成立时，执行 A 框操作，然后再判断 p 条件是否成立；如果仍然成立，再执行 A 框，如此反复直到 p 条件不成

立为止。此时不再执行 A 框而脱离循环结构。

- 直到（until）型循环：如图 3-13b 所示。先执行 A 框，然后判断给定的 p 条件是否成立；如果 p 条件不成立，则再执行 A，然后再对 p 条件作判断。如此反复直到给定的 p 条件成立为止。此时脱离本循环结构。

图 3-13　循环结构

注意这两种循环结构的异同：

1）两种循环结构都能处理需要重复执行的操作。

2）当型循环是“先判断（条件是否成立），后执行（A 框）”。而直到型循环则是“先执行（A 框），后判断（条件）”。

3）当型循环是当给定条件成立满足时执行 A 框，而直到型循环则是在给定条件不成立时执行 A 框。

下面举例说明这两种循环的异同之处，如图 3-14 所示。图中 a 和 b 都表示输出 1、2、3 这 3 个数字。其中 a 用当（while）型循环表示，b 用直到型循环表示。从中我们可以看出，用当型循环需要先判断 x 的值是不是大于 3，如果大于 3 那么不再执行“输出 x 值”这一步；而直到型循环则先执行“输出 x 值”的动作，然后再判断 x 的值是不是大于 3。

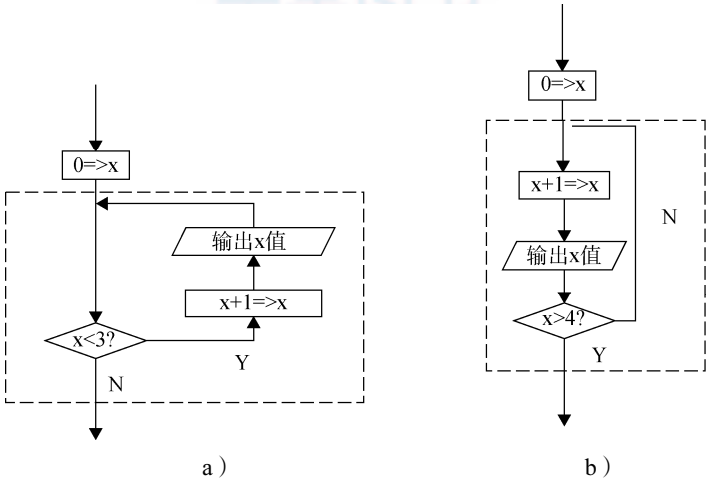

图 3-14　两种循环结构的异同

从以上 3 种基本流程图来看，我们可以归纳出基本结构有以下共同特点：

1）只有一个入口。无论哪种结构都只能有一个入口，不可能有多个入口。

2）只有一个出口。一个结构只能有一个出口。

3）结构内的每一部分都有机会被执行。即，对于每一个框来说，都应该有一条从入口到出口的路径通过它，没有路径的结构是不能有效地被执行的。

4）结构内不能存在无终止的循环，这样的话，程序将会一直执行下去，永不结束。

已经可以证明的是，利用上述 3 种基本结构顺序构成的算法，是可以解决任何复杂的问题的。由这些基本结构可以组成结构化的程序代码进行程序的执行。

3.3.4 用 N-S 流程图表示算法

1973 年，美国学者 Isaac Nassi 及其学生 Ben Shneiderman 提出了一种新型的流程图：N-S（N 和 S 就是这两名学者英文姓氏的首字母组合）流程图。与 3 种基本结构流程图的最大区别是不允许 N-S 流程图使用流程线，而将全部的算法写在一个框内，这样做带来的好处是使流程更加规范、清晰，避免了频繁使用流程线导致流程凌乱的弊端。这种流程图适合于结构化程序设计，因此也得到了广泛的应用。

N-S 结构化流程图用以下的流程图符号表示。

1）顺序结构：由 A 和 B 两个框组成一个顺序结构，如图 3-15 所示。

2）选择结构：其表示方法如图 3-16 所示。在图中，当 p 条件成立时执行 A 操作，p 不成立则执行 B 操作。

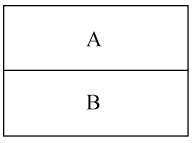

图 3-15　顺序结构 N-S 图

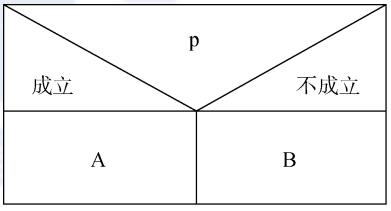

图 3-16　选择结构 N-S 图

3）循环结构：当型循环结构的表示方式如图 3-17 所示。图符表示先判断后执行，当 p1 条件成立时反复执行 A 操作，直到 p1 条件不成立为止。直到型循环结构如图 3-18 所示。图符表示先执行后判断，当 p1 条件不成立时反复执行 A 操作，直到 p1 条件成立为止。

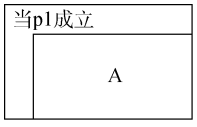

图 3-17　当型循环结构图

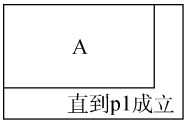

图 3-18　直到型循环结构图

利用以上 3 种 N-S 流程图的基本框可以组成任何复杂的 N-S 流程图，来表示不同的程序算法。下面我们对 3.2.2 节讲的几个例子用 N-S 流程图重新表示一下，以加深读者对 N-S 流程图的理解。

例 3.15：用 N-S 流程图表示例 3.1 中的算法，如图 3-19 所示。

例 3.16：用 N-S 流程图表示例 3.2 中求素数的算法，如图 3-20 所示。

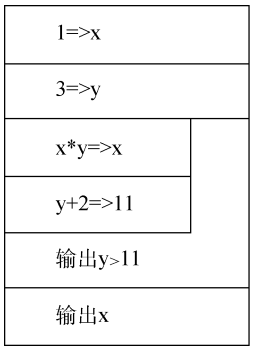

图 3-19　例 3.15N-S 流程图

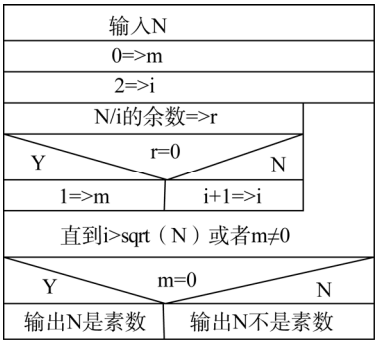

图 3-20　例 3.16N-S 流程图

例 3.17：用 N-S 流程图表示例 3.5 中求不同成绩的算法，如图 3-21 所示。

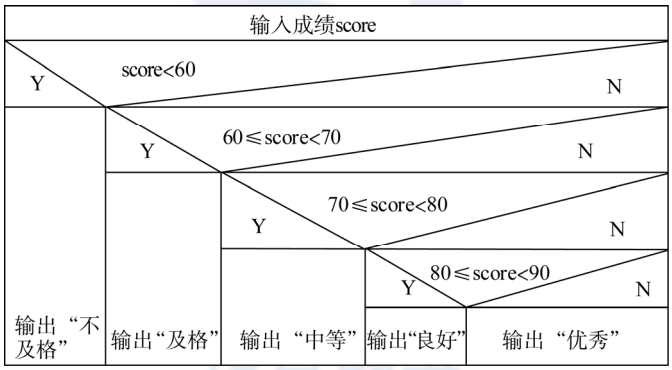

图 3-21　例 3.17N-S 流程图

例 3.18：用 N-S 流程图表示例 3.7 中求闰年的算法，如图 3-22 所示。

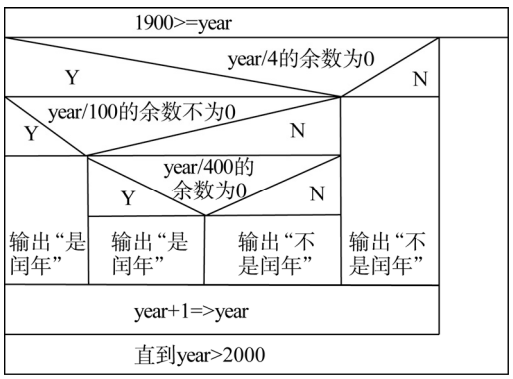

图 3-22　例 3.18N-S 流程图

通过以上几个例子的讲解，可以看出 N-S 流程图表示算法直观、形象，易于理解。而且使用 N-S 流程图简洁易画，没有了流程线，程序执行的顺序就是图中的上下顺序，写算法和看算法的时候只需从上至下即可。

3.3.5 用伪代码表示算法

用传统的流程图和 N-S 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，而修改流程图的过程是很麻烦的。因此，流程图适于表示一个算法，但是当算法比较复杂、需要反复修改时，在设计算法过程中使用流程图就不是很理想了。为了方便设计算法，常用伪代码来进行算法的设计。

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法。它如同一篇文章一样，自上而下地写下来。每一行（或几行）表示一个基本操作。它不用图形符号，因此书写方便、格式紧凑、易懂，也便于向计算机语言算法（即程序）过渡。伪代码是指对程序设计语言的精简的、非正式的描述，目的是便于人们的阅读和理解，而不是用于计算机的执行。

举一个简单的例子来简要对伪代码表示的算法进行说明。比如设计一个算法，当一个数 x>5 的时候就将它打印出来，用伪代码可以如下表示。

（1）用中文表示

```
如果 x 大于 5
将 x 的值打印出来
否则
   打印出一句话"x 的值不大于 5"
```

（2）用英文表示

由于各种编程语言都是由外国人设计的，所以用英文进行伪代码表示将更便捷。

```
if x is greater than 5 then
       print  x  !在 FORTRAN 中，print 表示输出
else
       print "the value of x is less than 5"
```

从上面可以看出，用伪代码表示的算法利于书写和阅读，即使是没有太高计算机基础的人也可以轻松地读懂一段代码。用伪代码写算法并没有固定的、严格的语法规则，只要把意思表达清楚且书写的格式清晰易读即可。

下面将例 3.1 到例 3.7 中的几个例子的算法用伪代码表示如下。

例 3.19：用伪代码表示 1×3×5×7×9×11 的算法。

```
开始
置 x 的初值是 1
置 y 的初值是 3
当 x<=11 时，执行以下操作
    使 x=x*y
    使 y=y+2
输出 x 的值
算法结束
```

用英文表示的伪代码如下：

```
begin !算法开始
      1=>x
      3=>y
while y<=11  !使用 while 表示循环开始
      x*y=>x
      y+2=>y
      end while  ! 结束 while 循环
write x
end !算法结束
```

在本算法中，使用的循环是当型循环。为了简洁方便并且培养读者的编程能力，下面的例题中的伪代码表示方法全部改用英文表示，因为英文更接近计算机语言。

例 3.20：两个数中的最大值（设 x 不等于 y）。

```
begin  ! 程序开始
     read x,y !输入两个数给 x 和 y
     if x>y
           print  x is greater than y
     else
           print  x is less than y
     end if
end  !程序结束
```

例 3.21：输入一名学生的成绩，输出相应的成绩区间。

```
begin !程序开始
     read score
     if score<60
            write "不及格" !write 也表示输出
     else if score>=60 and score<70
            write "及格"
     else if score>=70 and score<80
            write "中等"
     else if score>=80 and score<90
            write "良好"
     else if score>=90 and score<=100
            write "优秀"
     end if
end !程序结束
```

例 3.22：检验 1900 年 ~ 2000 年中哪一年是闰年。

```
begin
     1900=>year
```

```
        while year<2000
         if year 能被 4 整除
               if year 不能被 100 整除
               write "是闰年"
               else
                     if year 能被 400 整除
                     write "是闰年"
                     else
                     write "不是闰年"
                     end if
        end if
        else
             write "不是闰年"
        end if
        year+1=>year
        end while
    end
```

从上面的例子可以看出，利用伪代码书写灵活，格式自由，可以随着设计者的思路轻松地写下去，而且较容易读懂。另外，利用伪代码书写易于修改，无论是添加一行、删除一行还是修改其中的某一行都比流程图要省时省力，最重要的一点是，利用伪代码写的算法比较接近计算机语言，能够较容易地转化为计算机语言，这有利于提高程序编写人员的编程能力。

3.3.6 用计算机语言表示算法

在实际应用中，计算机是无法识别流程图和伪代码的，只有用计算机语言编写的程序才能被计算机执行。因此在用流程图或伪代码描述出一个算法后，还要将它转换成计算机语言程序。

用计算机语言表示算法必须严格遵守所用语言的语法规则，这是与伪代码不同的。其用途主要包括设计算法和实现算法两个部分，设计算法的目的是为了最终实现算法。

下面我们举 3 个利用 FORTRAN 语言实现算法的例子。

例 3.23：求 1×3×5×7×9×11。

Ex0301.f90

```
1       PROGRAM ex0301
2       IMPLICIT NONE
3           INTEGER::x=1
4           INTEGER::y=3
5           DO WHILE (y<=11)
6               x=x*y
7               y=y+2
8           END DO
9           WRITE(*,*) x
10  END
```

上述程序运行后输出 10395。

例 3.24：用 FORTRAN 语言表示例 3.5 中输入一个成绩表示不同区间的算法。

Ex0302.f90

```
PROGRAM ex0302
    IMPLICIT NONE
        INTEGER::score
        WRITE(*,*) "请输入学生成绩"
        READ(*,*) score
        IF (score<60) THEN
              WRITE(*,*) "不及格"
        ELSE IF ((score>=60) .and. (score<70)) THEN
              WRITE(*,*) "及格"
    ELSE IF ((score>=70) .and. (score<80)) THEN
          WRITE(*,*) "中等"
    ELSE IF ((score>=80) .and. (score<90)) THEN
          WRITE(*,*) "良好"
    ELSE IF ((score>=90) .and. (score<=100)) THEN
          WRITE(*,*) "优秀"
    ENDIF
 END
```

程序运行的输出如下：

```
请输入学生成绩
30 ↙（说明：↙表示从键盘输入数据之后按下回车键，后面也是如此含义，不再说明）
不及格
```

由于此处主要介绍程序的算法，因此，读者如果不明白上面两个程序也无所谓，待后面学习 FORTRAN 语言基础之后再回头看这两个程序即可，在此不做详细介绍。

3.4　本章小结

算法是程序的关键和灵魂。算法是使用计算机求解问题的思路和步骤，一个好的算法不仅能节省计算时间和存储空间，而且能够提高计算结果精度。本章主要介绍了程序设计方法及程序算法，读者要明确掌握程序算法的概念、表示方法。通过具体案例，对算法进行了详细的讲解，请读者仔细研讨本章内容，打下一个坚实的基础，从而能够较好地学习后面章节的内容。