# 第 3 章　算法与程序设计

## 3.1　上帝创造宇宙时的自言自语：程序

自言自语，就是一个人低声自己嘀咕给自己听。无所不能的上帝也会自言自语？我想是的。他太孤独了，为了排解孤独才创造了世界，也是挑战自我吧。

当然，我不是说上帝由于长期精神压抑或抑郁产生幻觉、幻听，他在与实际不存在的人进行言语沟通，不是的。我说的是每个人都有多重性格，当人们遇到棘手的问题、内心出现矛盾的时候，各种不同性格之间就会发生冲突，实际上，冲突的过程也正是人们思考的过程。有些人的思考过程是在内心进行的，当一个人专注于某一事情、完全沉浸在对这件事的思考之中的时候，会不经意地把内心活动的某些“片段”说出来，即所谓自言自语。心理学研究认为，自言自语是消除紧张的有效方法，自言自语有利于身心健康，只要不是与幻觉有关的自言自语都是正常的。它可以有效地发泄心中的不满、郁闷、愤怒及悲伤等不良情绪，有助于消除紧张，恢复心理平衡。当人们思虑重重时，若有机会听听自己的谈话，并对自己提一些问题，那么从一个角度看问题或钻牛角尖的可能性就会减小。

所以说，上帝为了排解孤独而创造了世界，这是一个很好的主意，而这样一个巨大复杂的工程给上帝带来的难题也是可想而知的，所以，上帝需要自言自语。

上帝自言自语地说了些什么？我想，应该和他建造的工程有关吧。

### 3.1.1　算法是什么

在计算机软件开发领域里，“算法”是人们经常挂在嘴边的一个词汇。不过，现在严格地考察“算法”一词的提出，恐怕十分困难。那么，算法是什么呢？今天，软件开发行业把求解问题的方法和步骤称之为算法。这里的“方法”可以理解为求解问题时数据计算和处理的规则，“步骤”强调的是运用这些规则的次序。

切不可漫不经心地对待“方法——数据计算和处理的规则”、“步骤——运用这些规则的次序”这两句话，在你能够深刻地体会、认识算法定义的时候，你才会认识到完整地表述算法不像是理解算法本身那么容易！在很多情况下，算法可以由更加基本的算法构成；不同的人描述同一个算法其表述的结果可能完全不同，就像给同一个人画肖像画一样，不同的人绘画的结果差别会很大。图 3-1 表示了算法的定义，它表示了求解问题时如何从输入的数据信息开始，经过计算过程得到输出结果，即为问题的解。然而，理解这个定义的内涵和外延还需要一定的经验和阅历。

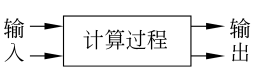

图 3-1　算法图示

例如，当我们说到“课程平均分数”这个概念时，有人会说“就是把各科的分数求和，然后除以科目数”，这是从评价个人学习状况角度产生的理解；另有一些人会认为“是每个学生的分数之和除以学生数”，这是从评价全体学生这门课程学习状况角度产生的理解。显然，这是完全不同的理解，可能还会有另外的不同理解。不过，不管怎样，理解“课程平均分数”这

个概念并不困难。这里讨论的是关于一个“概念定义”的问题。

从一定角度说，人们认识客观世界的结果往往可以抽象地表述为形成概念和表述概念的过程，例如，我们可以用长度表述物体的长短，用光洁度表述物体的表面粗糙程度。人们对客观世界的描述就是通过定义概念开始的，不论你是自觉地定义一种概念，还是不自觉地感受一种概念，对概念的把握是我们认识客观世界和表述这种认识的必要形式。有这样一种娱乐游戏——用一块不透光的布遮住被测试人的双眼，把物品放到一个只能从两个窟窿伸入双手的箱子里，被测试人仅凭双手触摸物品的触觉来判断物品的种类，也就是说，通过感知物品的材质（质感）、温度、大小、形态（构造）等触觉来判断物品的种类。当然许多被测试人都顺利地通过了测试，我们也知道大多数人都能通过测试，但是，如果让你准确地描述你的触觉，将你的触觉以数字（数据）的形式“刻度”出来，你会发现这是一件十分困难的事情。例如，你怎样告诉我们计算“苹果”的大小、形状、表面光洁度、重量（还可以增加其他因素）等参数来识别握在手里的哪一个是苹果、哪一个是“苹果梨”？从测量触觉和计算触觉的角度定义一个区别“苹果”与“苹果梨”的方法和步骤不是一个简单的任务。

求解问题的方法和步骤总是和判断数字（数据）条件、计算相关数据联系在一起的，因此称之为“算法”是很自然的。

以算法表述的方式来描述你对客观世界的认识是一个极具启发性的“游戏”，它的关键问题在于如何定义才不会产生歧义的理解，也就是说，被测试人（例如计算机）依据文字表述的算法识别物品的结果应该和我们识别的结果是一致的！

从算法定义角度说，前面讨论的“课程平均分数”是斟酌一个有关数据处理问题的算法描述问题（这里，我们从应用计算技术的角度把获得“课程的平均分数”这个计算任务的性质称之为数据处理），我们所要做的是严谨地给出这个概念的数学计算方法的文字描述，因为凡是要严谨定义的事情最好的方法就是用数学定义的描述方法。

首先，我们会想到应该把前面那两类大相径庭的理解做一点儿限制，以避免产生如前所述的歧义。例如“个人的课程平均分数是把个人各科的分数求和，然后除以科目数”，或是“某课程的平均分数是该课程中每个学生的分数和除以学生数”，这样就比较严格地定义了两个不同的概念：一个是某学生的课程平均分数，一个是某课程的学生平均分数，他们描述的对象是完全不同的，用途也完全不同。

在此基础上，给出它们更加严格的、颇有些数学定义味道的描述方案并不困难：

个人的课程平均分数定义为：设 S$i$ 为某学生第 $i$ 门课程的成绩，$n$ 为该学生的所有课程数量，$1\leqslant i\leqslant n$；该学生（个人）的课程平均分数为 $P$，则 $P=(S1+S2+\cdots+Si+\cdots+Sn)/n$，其中：$1\leqslant i\leqslant n$，$n$ 为大于等于 1 的正整数。

某课程的学生平均分数定义为：设 S$i$ 为某学生本课程的成绩，$n$ 为学习本课程的学生数量，某课程的学生平均分数为 $P$，则 $P=(S1+S2+\cdots+Si+\cdots+Sn)/n$，其中：$1\leqslant i\leqslant n$，$n$ 为大于等于 1 的正整数。

如果，我们喜欢用同一个定义，并使之同时适用于上述两种或更多种情况，借助图 3-1，可以这样来定义“平均数”的概念：

（1）输入各门课程分数。

（2）输入这些分数的个数。

（3）计算这些分数之和。

(4) 用分数之和除以分数个数。

(5) 输出平均分数。

(6) 解释这个输出：如果输入的分数是一个学生各科的课程分数，那么，这个平均数就是该学生的当前课程的平均分数；如果输入的是某课程所有学生的分数和对应的学生数，那么，这个平均数就是该课程的学生平均分数；如果输入的仅仅是一般意义上的数据和数据的个数，那么，输出的就是输入数据的平均数。

给出这个“多解”的颇似数学定义的描述也不困难：

设 S$i$ 为 0 和整数集合 $S$ 的数据元素，$S=\{S1,S2,\cdots,Si,\cdots,Sn\}$；$i=1,2,\cdots,n$；$n$ 为大于等于 1 的正整数。数据集合 $S$ 的平均数为 $P$，定义 $P=(S1+S2+\cdots+Si+\cdots+Sn)/n$。

显然，当集合 $S$ 是由某学生所有课程分数构成时，则 $P$ 是该学生的课程平均分数；当 $S$ 是由某课程学生的学习成绩分数构成时，则 $P$ 是该课程的平均分数。

也许，你认为上述讨论已经清楚地描述了相关内容，但是，这些描述有很多局限性，实际上我们自觉不自觉地忽略了某些“看似不重要”或是“无须赘述”的内容。

例如，“个人的课程平均分数是把个人各科的分数求和，然后除以科目数”这个文字描述看似没有什么疑义，但是，真的是这样的吗？或者说，这个描述对什么课程都可以这样定义吗？遇到下面的情况怎么办？——如果某门课程采用 4 个阶段考核的方法，前两个阶段是及格/不及格的记分方法，后两个阶段是百分制的方法，如何计算平均分数呢？如果某门课程没有参加考试，平均分数如何计算呢？

是的，用文字描述概念时，容易“视而不见”地忽略某些“情理之中、意料之外”的情况，而用数学的方法定义一个概念时，运用概念的定义对事物作出判断之前一定要检验是否符合定义的前提假设，只有符合前提条件的事物才能够进行判断，否则就是错误地运用了“数据处理规则”。对于前面几种例外情况，从数学的逻辑角度出发结论是十分清楚的，这些情况不在本概念运用情境之内，如要运用本概念还需要对这些情境进一步地进行补充定义。

至此，应该用开篇时得到的有关“算法”的定义来回顾一下上面的讨论了。可以看到：前面讨论了几种不同用途的平均数的定义，可以把对各个分数求和以及进行除法计算的规则理解为算法定义中所谓的“方法”，而判定运用这个计算规则是否适用的过程就是算法定义中所谓的“步骤”。常常在这些“步骤”里蕴含着问题领域的背景知识，即所谓的“语义”，当我们的问题在具体步骤中遇到了“语义”范畴的“情理之中、意料之外”的情况时，我们定义的算法就会经受一次“完整性”的考验。

用语言和文字描述一个概念时常常与问题背景、语义环境有着密切的关系，脱离了问题背景和语义环境而孤立地理解一段文字(或语言)就容易产生歧义的理解。当然，我们可以用语言和文字严格地描述一个概念，但是，这种描述的结果经常是撰写了一段令人费解的“绕口令”，而所要表达的意义早已经被“绕口令”搅得“云山雾罩”了。

将数学语言、生活口语、书面语言——这些文字和语言与算法描述用语对比地研究一下是具有启发性的。数学语言是十分严谨的，它的上下文之间的联系是明确而简洁的；日常生活的口语也可以做到精准而简洁，但是，交流的各方必须对当前的交流背景有一个共同而精准的理解，否则，稍有差异就会对谈话的内容产生歧义的理解；书面语言的形式更加丰富而灵活，除非必要，较少使用呆板、苛刻的数学语言。算法描述用语实际上具有数学语言的本质特点，但是，在达到数学语言的严谨程度之前，描述算法的文字和思维过程都将经过一番

耗费精力的“提炼”，最后，在摒弃日常口语的省略和疏漏之后，在简化书面语言丰富多彩的表达形式之后，在仔细审视各个概念的数量关系之后，才能够成为描述算法的优美的、有效的文字。当然，尽管我们希望描述算法的文字是优美的，但在追求优美与通俗易懂之间，我们应该坚持的质量准则首先是“逻辑严谨”！否则，逻辑不严谨的算法，文字再优美也是混乱和令人费解的。

或许，伽利略感慨于宇宙充满着各种各样的规律，感慨于这些规律是那样的严谨、那样的不可混淆才说过这样的话吧：“数学是上帝用来书写宇宙的文字”。

从阅读文章、理解文章精神的过程角度看，仅有数学公式的文章是难以阅读的。因此，上帝在创造宇宙的工程过程中（参见图 3-2 米开朗基罗的《创世纪》局部）自己也需要一些帮助记忆某些事情的其他语言，比如自然语言。我们知道理解数学公式从来都离不开自然语言的辅助说明，更何况，上帝在运用这些规律来创造事物呢。

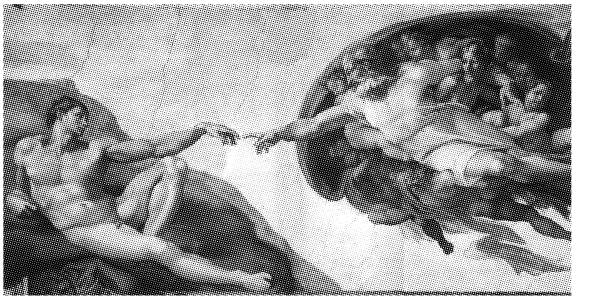

图 3-2　米开朗基罗的《创世纪》局部

如此说来，算法是什么？当把数学公式描述的内容对应地理解为算法定义中的“方法”，把有关运用这些公式的条件说明对应地理解为算法定义中的“步骤”，可以看到，算法与上帝创造宇宙时的“自言自语”是多么的一致呀！

算法是什么？算法是求解问题的方法和步骤，算法是上帝书写宇宙时的自言自语！

《创世纪》是米开朗基罗画在梵蒂冈西斯廷教堂礼拜堂天花板上的壁画，作品场面宏大，人物刻画震撼人心，它是米开朗基罗的代表作之一。

《创造亚当》是《创世纪》中最动人心弦的一幕，这一幕没有直接画上帝塑造亚当，而是画出神圣的火花即将触及亚当这一瞬间：亚当慵倦地斜卧在一个山坡上，他健壮的体格在深重的土色中衬托出来，充满着青春的力与柔和。他的右臂依在山坡上，右腿伸展，左腿自然地弯曲着。他的头，悲哀中透露着一丝渴望，无力地微俯，左臂依在左膝上伸向上帝……上帝飞腾而来，左臂围着几个小天使。他的脸色不再是发号施令时的威严神气，而是怜悯和慈善的情态。从天飞来的上帝，将手指伸向亚当，他正在像要接通电源一样将灵魂传递给亚当……

这一戏剧性的瞬间，将人与上帝奇妙地并列起来，触发着我们无限的敬畏之情。

### 3.1.2　计算机程序是什么

广义的程序是指按时间先后或依次安排的工作步骤，如工作程序、医疗程序。

计算机程序是为了使计算机执行一个或多个操作，或执行某一任务，按序设计的计算机

指令的集合。在软件开发领域里，一般未作特别说明的情况下，程序一词的意义是特指计算机程序。

因为算法是求解问题的方法和步骤，而程序是使得计算机执行操作或任务的指令集合，计算机执行操作的目的是解决问题，或者说是求解问题，所以，程序就是求解问题的方法和步骤。

通常来说，我们会把程序和算法等同起来，在研究求解问题的方法和步骤策略的时候，我们称之为算法，而在研究把求解问题的方法和步骤用编程语言来书写(实现)的时候，我们称之为程序。

在讨论算法是什么的时候，我们重点研究了算法在计算方法层面的内容，如 $P=(S1+S2+\cdots+Si+\cdots+Sn)/n$，以及参与计算的元素 $Si$ 为 0 和整数集合 $S$ 的数据元素，$S=\{S1,S2,\cdots,Si,\cdots,Sn\}$；$i=1,2,\cdots,n$；$n$ 为大于等于 1 的正整数等前提条件。这样的描述在许多时候，人们认为作为“平均数算法描述”来说已经清楚了。而事实上，“求解平均数”的步骤还没有描述出来，不同的人描述其求解的步骤可能差别很大，对应地，也就是说他们“设计”的程序可能大不相同。

如果让计算机较为方便地辅助人们完成这个计算任务，仅仅考虑这些还是不够完善的。也就是说，这样的描述还需要进一步补充内容，才能够作为设计计算机程序的算法。

还需要补充哪些内容呢？

(1) 我们要确认 $Si$ 是整数，还是小数。显然不能是优秀、良好、中等、及格、不及格之类的评语，如果是这种类型的评语，我们需要重新定义“A 个优秀＋B 个良好＝？”之类的算法。

(2) 要设计 $Si$ 输入计算机的方式，是采用一次输入一个数据的人机交互方式吗？那么这个输入界面的布局是怎样的？或者将 $Si$ 以数据文件的形式存储在磁盘中吗？那么以怎样的方式录入这份数据文件呢？数据文件的命名有没有遵守什么规则的必要？

(3) 任何工作的过程不可避免地都可能出现错误，假如把正整数错误地输入成了某个字符，例如，把数字“9”输入成了“)”该怎样处理？如果把“90”输入成了“900”该不该自动识别它是错误的呢？如果不识别这种类型的错误又怎样保证 $Si$ 的正确性呢？

(4) 如果总共输入了 20 个 $Si$，但是错误地把 $n$ 输入成了 30，有没有办法避免这样的错误呢？

(5) 要不要考虑节省存储数据的空间问题？也就是说，是首先输入 $n$，然后根据 $n$ 来定义 $n$ 个存储单元来存储 $n$ 个 $Si$；还是在接收到 $n$ 之后，以 $n$ 来控制进行 $n$ 次的 $Si$ 累加，这样可以节省 $n-1$ 个存储单元；如果 $n\leqslant 200$，你也许不觉得输入方式对使用者有什么影响，但是，如果 $n\geqslant 200$，你可能会改变 $Si$ 的输入方式，采用数据文件的形式而并非是一次输入一个数据的人机交互方式；甚至，你会认为增加一段用于校验数据输入是否正确的程序更有意义。总之，你需要从实际使用时 $n$ 的可能情况出发来决定输入和存贮数据的方式问题，而这些问题在描述算法时通常被忽略了。

(6) 如果需要考核这些 $Si$ 录入者的工作数量和质量，甚至需要追究他们的责任，你还需要记录每个 $Si$ 的录入人姓名和录入时间等情况。

(7) 你可能还需要考虑使用这个平均数的方式，是仅仅使用 1 次？还是可能随时都要使用？抑或是可能使用多次或一次都不会使用？为此，你同样要考虑计算 $P$ 的有关策略以及这个功能与其他功能的配合使用问题：当 $n$ 较小的时候，计算 $P$ 的不同策略对于使用效

果可能不会造成很大影响，但是，当 $n$ 足够大时，就需要考虑计算机的运行时间、存储空间的优化问题以及与其他功能协调使用的问题了，比如，其他功能希望得到瞬间的快速响应，而不是等待 1 秒钟以上的时间。

(8) 也许，你的用户会对这个软件的界面和数据录入的方式有很别致的要求，在你准备好上面的策略之后，还是要认真地、耐心细致地与用户进行交流，把所有用户听得懂的或是必须获得用户意见的(重点问题应该放在可能影响功能和性能的地方)方案以十分有效的方式与用户交流，在获得用户的认可之后才可以开始进一步的程序设计工作。

程序作为“使计算机执行一个或多个操作或执行某一任务，按序设计的计算机指令的集合”，上面 8 个方面的讨论都从根本上决定了这个指令集合的不同内容和次序，也决定了计算机执行操作任务时的性能和状态。在这些补充的讨论中，我们关注的仍然是如何实现 $P=(S1+S2+\cdots+Si+\cdots+Sn)/n$，但是，集中讨论的内容是如何界定计算 $P$ 时的前提条件和计算过程的“有效性”。

仔细审视上面的讨论内容，严格地说来，我们还不能把算法和程序简单地等同起来。实际上，从程序设计者角度看，从一般意义的算法过渡到程序状态还有很多问题需要解决，甚至讨论这些问题的重要性比起严格地定义算法工作的重要性毫不逊色，只有诸如上面 8 个小问题都一一解决了，周详地制定了求解问题的方法和步骤，才可以开始下一步的编程工作。说不定，你可能还会发现需要限定的其他问题呢。

综上所述，计算机程序是算法的实现，或者说，是把用“自然语言＋数学定义”撰写的算法再用程序设计语言翻译给计算机的过程。这个过程需要更加精细地表述解决问题的方法和步骤，不仅要考虑数学意义的逻辑关系，还需要考虑其他“不言而喻”的种种约束条件，既包括计算机软硬件的技术约束条件，也包括用户对求解问题的使用性要求，诸如界面样式和响应速度、方式等。

### 3.1.3 程序是怎样的

早期的计算机程序是图 3-3 所示的样子。

00100000 01101100 01110101 01100011 01101011 01111001 00001010

图 3-3 二进制程序图示

这是一段二进制编写的计算机程序，通常，它们是长长的一串二进制的编码。把它们穿孔在纸带上或卡片上，用纸带机或读卡机输入到计算机磁盘，在操作系统控制下就可以运行了。

每个人都会在这长长的由“0”、“1”构成的编码面前迷惑，初看起来什么也看不懂，但是，这就是早期广泛采用的二进制计算机程序的样子。

这样的计算机程序一直使用到现在，并称之为“机器语言”编写的程序。实际上，它还将存在相当长的一段时间，迄今为止，计算机硬件设备主要是，甚至可以说仅仅是可以运行这样的计算机程序。

那么，这些 0、1 到底是怎样表达解决问题的方法和步骤的呢？

那一串串的由 0、1 组成的符号，我们把它叫做编码。00100000 和 01101100 是两个不同的编码，原因是 0、1 字符所在的位置不同。实际上我们能够做到：不论一个集合是由多

少个元素构成的，只要它是有限的，我们就能够给每个不同的元素指定一个唯一表征它的编码。

既然由 0、1 字符构成的编码难以记忆、难以阅读，人们自然想到采用便于记忆和阅读的编码进一步替代二进制编码。

图 3-4 所示为“汇编语言”编写的计算机程序。左侧是程序代码，右侧“；”号后面的文字是有关代码的解释。它已经具有了便于记忆和阅读的特点了，但是仍然不是十分理想。

```
LOOP:              ; 标号
CLR P2.6           ;选中p2.6 数码管左边的8字使能
SETB P2.7          ;p2.7不使能。 右边的数码管消隐
MOV P0,#28H        ;把28h送p0口；数码管显示 0    ;28为1010000
LCALL DELAY        ;  延时程序
MOV P0,#0FFH       ;0ffh 送p0口，数码管清除；P0口为11111111
CLR P1.0           ;点亮p1.0发光管； P1。0为电平，P0口为11111110
MOV P0,#7EH        ;把7eh送p0口；数码管显示 1； P1。0为低电平，P0口为11
111110
LCALL DELAY        ;  延时程序
MOV P0,#0FFH
MOV P0,#0FFH       ;0ffh 送p0口，数码管清除；P0口为11111111 清一次显示
```

图 3-4　汇编程序图示

下面是一段用“高级语言”编写的计算机程序。这段程序的任务是：对于数列 1、1、2、3、5、8、13、21、34…，求第 30 位数是多少。

我们用递归算法实现，程序如下：

```
public static int Recursion(int N)
    {
        If(N>0)
        {
          if(1>=N||N<=2)
            return 1;
          if(N>2)
            return Recursion(N-1)+Recursion(N-2);
        }
        else
          return 0;
    }
```

在英文环境下，这是一段十分接近自然语言的程序了。Recursion($N$)是递归地调用自身的函数，Recursion($N-1$)计算出前一个数，Recursion($N-2$)计算出前两个数。当我们分析出这个数列的规律时，再看这段程序，它几乎就是数列通项公式“编程语言”化的描述。

至此，计算机程序设计语言从二进制编码的“意义指代”阶段发展到汇编语言的“助记符”阶段，再发展到近似于“自然语言”的高级语言阶段，方便地编写程序的性能逐步获得了提高，编写计算机程序的工具环境日趋人性化。

下面是摘自 http://bbs.elephantbase.net/viewthread.php?tid=775 的一则“程序员随笔”，读一读，看看你能感觉到什么？

大约在年前，公司通知我2月21日到苏州工业园区面试，接到面试通知后的几天我把一些专业课温习了一遍，特别是C++和数据结构，由于大学几年里，我一直钻研这些方面，加上通过了高级程序员的考试，对于一些常用的算法我差不多也达到了烂熟于胸的地步，当时的感觉是如果问了我这些方面的问题我应该是没有问题的！21日那天我被安排在4:30面试，由一位技术人员单独给我面试，在问了一些简单的问题之后他给我出了一道编程题，题目是这样的：

（由于具体面试的题目比较烦琐，我将其核心思想提取出来，写一个函数计算当参数为$n$（$n$很大）时的值1－2＋3－4＋5－6＋7－…＋$n$）

哼，我的心里冷笑一声！没想到这么简单，我有点紧张的心情顿时放松起来！于是很快给出我的解法：

```
long fn(long n)
{
    long temp=0;
    int i,flag=1;
    if(n<=0)
    {
        printf("error: n must>0);
        exit(1);
    }
    for(i=1;i<=n;i++)
    {
        temp=temp+flag*i;
        flag=(-1)*flag;
    }
    return temp;
}
```

搞定！当我用期待的目光看着面试官的时候，他微笑着跟我说，执行结果肯定是没有问题！但当$n$很大的时候我这个程序执行效率很低，在嵌入式系统的开发中，程序的运行效率很重要，能让CPU少执行一条指令都是好的，他让我看看这个程序还有什么可以修改的地方，把程序优化一下！听了这些话，我的心情顿时变得有点沉重，没想到他的要求很严格，之后我对程序进行了严格的分析，给出了改进的方案！

```
long fn(long n)
{
    long temp=0;
    int j=1,i=1,flag=1;
    if(n<=0)
    {
        printf("error: n must >0);
        exit(1);
    }
        while(j<=n)
```

```
    {
        temp=temp+i;
        i=-i;
        i>0?i++:i--;
        j++;
    }
    return temp;
}
```

虽然我不敢保证我这个算法是最优的，但是比起上一个程序，我将所有涉及乘法指令的语句改为执行加法指令，既达到了题目的要求而且运算时间上缩短了很多！而代价仅仅是增加了一个整型变量！但是我现在的信心已经受了一点打击，我将信将疑地看着面试官，他还是微笑着跟我说："不错，这个程序确实在效率上有了很大的提高！"我心里一阵暗喜！但他接着说这个程序仍然不能达到他的要求，要我给出更优的方案！天啊！还要优化！我当时真的有点崩溃了，想了一会儿后，我请求他给出他的方案！然后他很爽快地给出了他的程序！

```
long fn(long n)
{
    if(n<=0)
    {
        printf("error: n must>0);
        exit(1);
    }
        if(0==n%2)
        return(n/2)*(-1);
        else
        return(n/2)*(-1)+n;
}
```

搞笑，当时我目瞪口呆，没想到他是这个意思，这么简单的代码我真的不会写吗，但是我为什么没有往那方面想呢！他说的没有错，在 $n$ 很大很大的时候这三个程序运行时间的差别简直是天壤之别！当我刚想开口说点什么的时候，他却先开口了："不要认为 CPU 运算速度快就把所有的问题都推给它去做，程序员应该将代码优化再优化，我们自己能做的决不要让 CPU 做，因为 CPU 是为用户服务的，不是为我们程序员服务的！"多么精辟的语言，我已经不想再说什么了！

接着是第二个问题……

这段故事需要你静下心来读，希望你能够读懂那三段代码的差异。现将一位同学用程序框图（如图 3-5～图 3-7 所示）的方法表述了他对这三段程序的理解，推荐给你作参考。

也许，你会在这位同学的框图中发现他的疏漏或者是错误，那么，请细心地给出你的框图并和同学们讨论你的想法，这样做是很有益处的。你会品味出那段故事中人物的不同个性和职业韵味，你会懂得软件质量的职业内涵……实际上，我们的学习、生活又何尝不是如此呢。

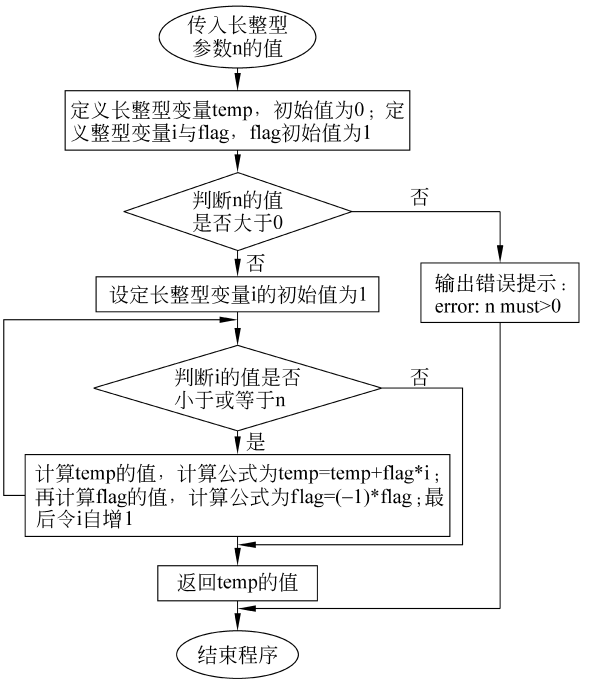


图 3-5　第一段程序框图

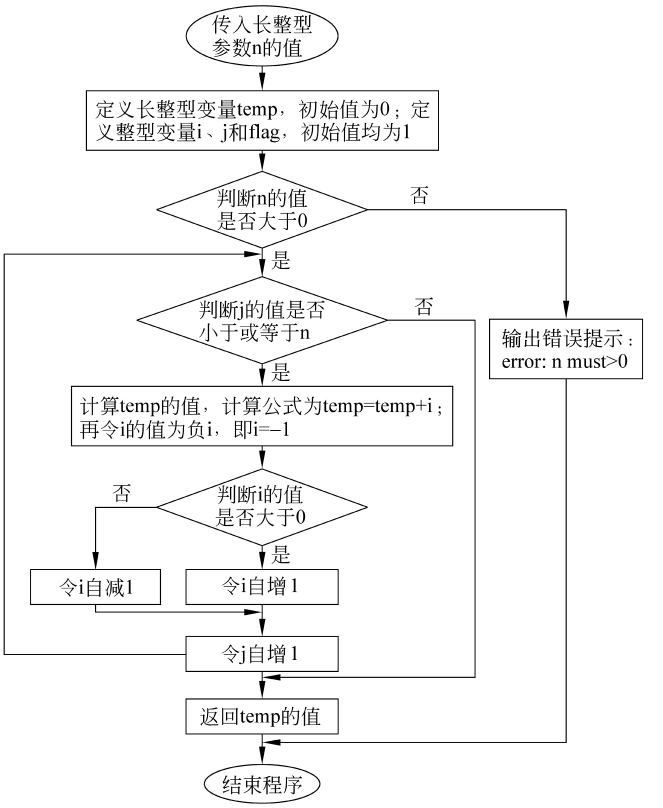


图 3-6　第二段程序框图