面向21世纪非计算机专业计算机系列教材

C语言程序设计教程

景 红 王国强 主编
尹治本 主审

西南交通大学出版社

面向21世纪非计算机专业计算机系列教材

C语言程序设计教程

景　红　王国强　主　编
尹治本　主　审

西南交通大学出版社
·成　都·

内 容 摘 要

本书对程序设计的三个组成部分：算法、C 语言数据结构、C 语言语法规则做了比较全面、详细的介绍。同时介绍了涉及音乐、图像等方面的高级程序设计方法及技术。

本书程序设计例题多、内容丰富、深入浅出。叙述较为系统而全面。可作为本科、专科学生 C 语言课程的教材。也可作为广大工程技术人员的 C 语言参考书。

图书在版编目（CIP）数据

C 语言程序设计教程/景红，王国强主编. —成都：西南交通大学出版社，2000.7
ISBN 7-81057-409-4
Ⅰ.C... Ⅱ.①景... ②王... Ⅲ.C 语言-程序设计
Ⅳ.TP312
中国版本图书馆 CIP 数据核字（1999）第 56796 号

C 语 言 程 序 设 计 教 程
景 红 王国强 主编
尹治本 主审
*
出 版 人 宋绍南
责任编辑 王 旻
封面设计 唐利群
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 发行科电话：7600564）
http://press.swjtu.edu.cn
E-mail:cbs@center2.swjtu.edu.cn
四川森林印务有限责任公司印刷
*
开本：787mm×1092mm 1/16 印张：16.25
字数：340 千字 印数：1～5000 册
2000 年 7 月第 1 版 2000 年 7 月第 1 次印刷
ISBN 7-81057-409-4/TP·223
定价：22.00 元

前　言

随着科学技术的飞速发展，计算机已经成为各行各业不可缺少的应用工具，计算机文化和应用已成为培养当代高质量大学生的一个重要组成部分，是 21 世纪的高等教育最需要加强的教学环节之一。为了满足我校非计算机专业计算机系列课程的教学改革需求，我们编写了系列课程教材，该书是其中一本。

程序设计是利用计算机解决实际问题最基本、最重要的方法，也是计算机文化的一种思想方法。程序设计语言很多，其中的 C 语言是一种结构化、模块化的高级程序设计语言，被广泛地应用于系统软件的开发和应用程序的设计中。C 语言有着良好的可移植性，用 C 语言编写的程序在不同的计算机系统间很容易转换。因此，C 语言是目前最流行的高级程序设计语言之一，它也成为本书所选用的高级语言。

本书是在我校本科试用教材的基础上，根据教学效果对其内容进行了更新和调整。与传统教材相比，本书的特点是突出程序设计主线，可切实提高读者应用能力。全书在内容的编写上作了较大改动：① 增加了算法内容。首先从编程的主线上介绍算法的基本概念、算法的表示与设计方法，尽可能地使读者能够在分析和解决问题的能力上切实得到提高；② 将介绍 C 语言各种数据结构的内容合并为一章，并增加了对这些数据结构的比较与应用归纳。使读者对 C 语言的数据结构有一个整体了解，编程时能够合理表示问题中涉及到的数据结构；③ 书中各章节有针对性地加入了大量的程序设计题，且每个例题都从算法的分析、数据的表示两个方面做了比较认真、仔细和全面的讨论与介绍。

参加编写的工作人员都是在高校长期从事计算机基础教学的一线教师，有着丰富的教学经验。力图从程序设计主线上来提高读者程序设计的思维方法和编程能力，但由于时间仓促、错误难免，敬请读者不吝赐教。

参加本教材编写工作的教师有：景红、王国强、尹治本、戴克俭、钱晓群、刘军、凯定吉、张丽梅、辛勤。

编　者

2000年5月

目　录

第 1 章　算法概述

[主要内容]　本章将着重叙述程序设计的一个重要环节——计算机算法。其中包括计算机算法的基本概念、算法的设计原则和算法的表示方法等。同时，对算法的复杂性我们也作了一些有意义的讨论。

§1.1　算法基本概念

广义的讲，算法就是解决某类具体问题的方法和步骤，即处理事宜的全过程。简单地说，**计算机算法**就是利用计算机解决问题的方法。我们知道，计算机的每一步工作（运算）都是人们预先为它编制好的，它仅仅是快速、高效地执行人们的“指令”而已。编制一组实现确定算法的指令集称之为**程序设计**。著名的计算机科学家沃思（Niklaus Wirth）说过：“程序就是在数据的某些特定的表示方式和结构的基础上，对抽象算法的具体描述。”他用以下的公式表达程序的实质：

算法 + 数据结构 = 程序

算法和数据结构是程序设计的重要环节。现在的程序设计不仅仅局限于数值计算，也包含了大量的数据加工和信息处理。因此，程序设计的一个策略是：当我们处理数值计算或信息处理问题时，首先建立出反映问题本质的数学模型，并选择合适的算法，然后根据算法确定合理的数据结构进行编程，才能正确地利用计算机求出问题的解。

算法是计算机科学与技术的核心问题，我们可以通过日常生活中的一些例子对算法深入地了解。

[例 1-1]　比如看电影，我们需要如下的步骤来实现：

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

这是一个众所周知的例子，在我们的日常生活中这样的事例随处可见。

§1.2 算法设计初步

1.2.1 算法的特征

利用计算机来解决实际问题，需要找出解决这些问题的算法，再将其按照计算机程序设计语言的要求付诸实施。

算法是求解一个问题类的无二义性的有穷过程。一个算法应具有下列五个重要特征：

• 有穷性　一个算法总是可以在执行有穷步之后完成，且每一步都可在有穷的(合理的，可接受的)时间内完成。

• 确定性　算法中每一条指令都有确切的含义，不会产生二义性。且在任何条件下，算法只有唯一的一条执行路径，对于相同的输入只能得出相同的结果。

• 可行性　算法中描述的所有操作都可通过已经实现的基本运算执行有限次来实现。

• 输　入　一个算法可有零个或多个输入。通过输入给算法中的某些变量赋予特定的值。

• 输　出　一个算法可以有一个或多个输出。通过算法的输出把算法的中间结果、最终执行结果等在输出设备上输出。任何一个算法至少应有一个输出。

1.2.2 算法设计的主要原则

通常要设计一个“好”的算法应考虑以下因素：

• 正确性　算法应满足具体问题的要求。算法正确性的验证，可以通过验证表达该算法的程序的正确性来实现。但这种验证是有局限性的，对于一个大型的比较复杂的算法，很难做到把每种可能的输入及其执行情况举完，因此这种验证并不能够保证算法完全正确。另一种方法是从理论上证明算法的正确性，目前还很困难。

• 可读性　算法的一个主要目的是为了人们阅读与交流。一个可读性好的算法有助于对算法的理解，易于调试和修改等。

• 健壮性　当输入非法数据时，算法也能够作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。

• 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算时间和计算机的存储空间。简言之，占用时间越短效率越高，占用内存空间越少效率越高。

1.2.3 算法设计的步骤

• 分析问题　分析问题是解决问题的第一步。深入地研究分析问题可以加深对问题的认识和理解。弄清楚未加工的原始表达，弄清所求，从中找出有用的信息等等。

• 建立数学模型　模型选择是否恰当，对解决问题的影响极大。利用计算机解决科学计算问题必须有恰当的数学模型，否则无从下手。应考虑最适合这个问题的数学结构是什么，有无已解决的问题借鉴等等。

• 算法的详细设计　算法的详细设计是指采用计算机的操作来设计解决问题的一系列步骤。不同的模型有不同的算法，相同的模型也可有不同的算法。

• 算法的正确性　算法的正确性可以通过对程序正确性的证明和验证两种方法来保证。对程序正确性证明目前仍很困难，而对程序正确性的验证又很难保证对所有的可能进行验证。因此，这是程序设计中一个长期存在的难题。

• 算法的实现　根据算法编制计算机程序，并在计算机上调试、实现。

• 算法分析　算法分析是对算法占用的计算机资源(运算时间和占用存储空间)进行度量或估计,并以此作为标准评价不同算法的优劣性。

1.2.4 应用举例

[例 1-2]　计算 5!。

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

计算 n!

前面我们介绍计算 5！的算法，实际上是对手工操作的描述，所以理解起来并不难。但是，随着 n 值加大，比如：n=100，如果还是采用同样的方法求解，算法的描述就会过于繁琐，1×2×……×100 要写 100 个步骤。所以，该算法并不可取。

分析阶乘的求解过程，我们不难看出：它实际上是一个循环求积的处理过程。所以，我们可以设两个变量：一个变量代表被乘数，一个变量代表乘数，直接将每一步的乘积放在被乘数变量中。

设 t 为被乘数，i 为乘数，可将求解阶乘算法描述为以下一种通用方式：

S1：输入 n 值

S2：将 1⇒t

S3：将 2⇒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

[例 1-3]　判定 2000～2500 年中的每一年是否为闰年。

分析：闰年的条件是：·能被 4 整除，但不能被 100 整除的年份是闰年；

·能被 100 整除，又能被 400 整除的年份也是闰年。

设 y 为年份，则可将解决该类问题的算法表示如下：

S1：2000⇒y

S2：若 y 能被 4 整除，不能被 100 整除，则输出 y“是闰年”，并转到 S5

S3：若 y 能被 100 整除，又能被 400 整除，则输出 y“是闰年”，并转到 S5

S4：输出 y“不是闰年”

S5：y+1⇒y

S6：当 y ≤2500 时，转到 S2 继续执行，否则算法结束

[例 1-4] 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$。

设 sum 表示部分和，deno 表示某项的分母值，term 表示某项的值，sign 表示某项应取的符号

其算法可表示如下：

S1：1⇒sum

S2：2⇒deno

S3：1⇒sign

S4：(-1)×sign⇒sign

S5：sign×(1/deno)⇒term

S6：sum+term⇒sum

S7：deno+1⇒deno

S8：若 deno ≤100，返回 S4 继续执行，否则执行 S9

S9：输出结果 sum，算法结束

[例 1-5] 欲在按非降序排列的 n 个元素 $a_1,a_2,\cdots,a_n(a_i \leqslant a_{i+1})$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，即先用位居‘中点’的元素 $a_{(n/2)}$ 与 b 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 $a_1,a_2,\cdots,a_{(n/2-1)}$ 中采用上述方法继续查找；否则在 $a_{(n/2+1)},a_{(n/2+2)},\cdots,a_n$ 中采用上述方法继续查找。这种算法显然要优越于前一种算法，因为最多也只需要比较 $\log_2 n$ 次。

以上的例子告诉我们：不同类型的问题有不同的算法，同一类型的问题也可有多种算法。当解决一个实际问题时，可能会有若干个算法供选用，我们要对这些算法进行分析，才能知道哪一个算法最好。

§1.3 算法的复杂性分析

一个好的算法标准要求：(1) 需要较少的存储容量；(2) 具有较高的运算效率。

算法的复杂性分析是对算法效率的度量，是评价算法优劣性的重要依据。一个算法复杂性的高低体现在运行该算法所需要的计算机资源的多少上，所需资源越多，则称该算法的复杂性越高；反之，称该算法的复杂性越低。

当问题的规模增长时，不同的算法所需要的计算时间按不同的增长率增长。我们用一个与问题相关的整数 N 来代表问题的规模，如表 1-1 所示：

表 1-1 问 题 的 规 模

问　　题	问题的规模 N
在一个数组中查找 X	数组元素的个数
两方阵相乘	矩阵的阶
对一组数排序	这组数的数据个数
解图的有关问题	图的节点数和图的边数
处理集合问题	集合的元素个数

算法的时间增长率就成为 N 的函数，记为 **T**(N),称为算法时间复杂性函数。算法的存储空间增长率记为 **S**(N)，称为空间复杂性函数。这里我们主要讨论时间复杂性，空间复杂性的分析与之类同。

例如，两个 n 阶矩阵相乘 A × B⇒C 其算法及复杂性分析如下：

```
1  for  i←1  to  n  do
2      for  j←1  to  n  do
           begin
3            c[i,j]←0;
4            for  k←1  to  n  do
5                c[i,j]←c[i,j]+ a[i,k] × b [k,j]
           end;
```

在该程序中，主要有乘运算、加运算和赋值三种基本操作。“乘”和“加”都是 n^3 次，赋值是 n^3+n^2 次。而占用内存是 $3n^2$。因此有：

$$T(n)=t_1 \times n^3 + t_2(n^3+n^2) \qquad S(n)= C \times 3n^2$$

式中的 t_1 为每次乘运算的时间，t_2 为每次赋值运算的时间，C 为每个元素占用内存的字节数。

随着计算机解决的问题越来越复杂，规模越来越大，要对所有的操作进行精确统计、计算是非常困难的。为了简化问题，便于抓住主要矛盾，我们只关心随着规模增长时增长率最快的操作。当问题的规模递增时，时间复杂性的极限称为**渐近时间复杂性**。一般不需要知道精确的时间耗费，只要知道时间耗费的增长率在什么范围内即可，因此我们引入算法复杂性的阶的概念。

如果对某一 C(C > 0),一个算法在时间 Cn^2 内能处理尺度为 n 的规模，则称此算法的时间复杂性是 n^2 阶的，记为 $\mathbf{0}(n^2)$。可定义如下：

若存在 C > 0 和 n_0,使当 $n \geqslant n_0$ 时有 $T(n) \leqslant f(n)$,则称 T(n)是 0(f(n))的,记为 T(n)= 0(f(n))。此时，f(n)是 T(n)增长率的一个上界。

在矩阵相乘的上例中，$T(n)=0(n^3)$,是 n^3 阶的。又如，二分查找法是 $0(\log_2 n)$的。顺序逐个查找法是 0(n)的。常见的阶有：

$\mathbf{0}(1)$，$\mathbf{0}(\log n)$，$\mathbf{0}(n)$，$\mathbf{0}(n\log n)$，$\mathbf{0}(n^2)$，$\mathbf{0}(n^3)$，$\mathbf{0}(2^n)$，$\mathbf{0}(n!)$等。

在相同的运行环境下，同一个算法也会有不同的时间复杂性。例如，在 n 个元素中查找

是否有与 x 相同的元素，采用顺序查找算法。最好情况是第 1 个元素就是，只需查找 1 次；最坏情况是找到最后一个元素，查找 n 次；平均情况是查找 n/2 次。可记为 Tmax(n)=n,Targ(n)=n/2,Tmin(n)=1。而我们主要关心**平均情况和最坏情况**。

有人认为随着计算机运算速度的快速增长，算法时间复杂性问题已不成其问题，这是十分错误的观点。事实上对许许多多的问题，仍然不能把希望寄托在提高计算机的运算速度上，而应着眼于寻求更高效(更低阶)的算法。

表 1-2 给出了 A1,A2,A3,A4,和 A5 五个不同阶的算法，比较它们在计算机速度为 10^7/s 和 10^9/s（即增长 100 倍）的情况下，单位时间(每秒)内能处理问题的最大规模：

表 1-2 五个不同阶的算法

算法	时间复杂性 T(n)	N	
		10^7/s	10^9/s
A1	N	10^7	10^9
A2	n^2	3162	31623
A3	n^3	215	1000
A4	2^n	23	30
A5	n !	10 ~ 11	12 ~ 13

可以看出对于算法 A5，计算速度增长了 100 倍，而处理规模仅增加了大约 1。

例如，著名的货郎担问题(Traveling Salesman Problem，TSP)，货郎担要访遍各个城市，沿途不走重复路，要求选择一条最短的周游路线。设想货郎担从起点城市出发，第一个目的城市将有 n-1 种选择，第二个目的城市将有 n-2 种选择，等等。可供选择的回路共有(n-1)!条，考虑其中有一半只是走向不同，实际上是相同的回路，即有(n-1)!/2 条。再考虑每条回路做 n 次加运算，需要 n!/2 次加运算。若货郎担问题采用这样的算法，将有：

$$T(n)=n!/2$$

若 n=30(约为全国省会城市)，采用每秒 10^{12}(1 万亿)次加的计算机计算，将有

$$30!/2=1.32\times10^{32} \qquad 1.32\times10^{32}/10^{12}=1.32\times10^{20}\text{秒}=4.2\times10^{12}\text{年}$$

即需要 4.2 万亿年！

一个算法的时间复杂性,如果是 $O(n^2)$的,称之为**好算法**；如果是 $O(n^k)$(k 为有理数)，称之为**多项式时间复杂性算法**,或称有效算法；而如果是 $O(2^n)$，则称之为**指数时间复杂性**。对指数时间复杂性的算法在规模稍大时是很难解决的。分析算法是一个复杂的工作，需要组合分析、图论、集合论、概率论等方面的理论基础。需要时,这方面内容可以参考其它的书籍。

§1.4 表示算法的方法

为了表示一个算法，可以用不同的方法。常用的有：自然语言、流程图、结构化流程图、伪代码等。

1.4.1 用自然语言表示算法

自然语言是人们日常使用的语言，它可以是汉语、也可以是英语或其它文字。用自然语

言表示通俗易懂；但文字冗长，含义往往不太严格,容易出现“歧义性”，要根据上下文才能判断其正确含义。而且用自然语言来描述包含分支和循环的算法很不方便。因此，除了一些简单的问题外，一般不用自然语言描述算法。例 1-1 ~ 例 1-5 的算法都是采用自然语言表示的。

1.4.2 用流程图表示算法

流程图是用一些图框表示各种类型的操作。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) 规定了一些常用的流程图符号 (见图 1-1)，已为世界各国程序设计者普遍采用。

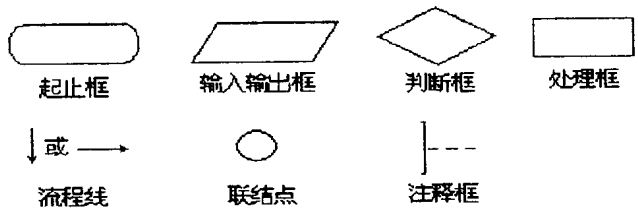

图 1-1 流程图符号

[例 1-6] 用流程图描述求 5!的算法 (见图 1-2)。

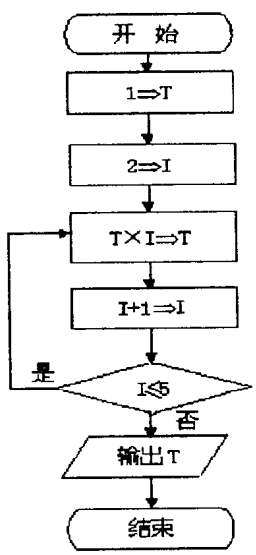

图 1-2 5!的算法流程图

1.4.3 三种基本结构和改进流程图

1. 传统流程图的弊端

传统流程图用流程线指出各框的执行顺序。由于传统流程图对流程线的使用没有严格限

制，所以设计者可以毫无规律地随意地将流程转来转去（如图 1-3），造成算法的逻辑难以理解，大大降低了算法的可靠性和可维护性。

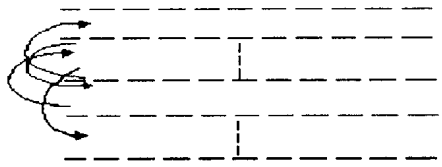

图 1-3　滥用流程线

为了提高算法的质量，必须限制流程线的使用。同时，由于问题的解决难免会包含一些分支（判断）处理和循环（重复）执行，所以流程不可能只是顺序地进行下去。为了解决这些问题，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，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。

2. 三种基本结构

1966 年，Bohra 和 Jacopini 提出了以下三种基本结构，这三种基本结构作为表示一个良好算法的基本单元。

（1）顺序结构（见图 1-4）

从 a 点进入，顺序地先执行 A 框操作，然后执行 B 框操作，最后从 b 点脱离该结构。

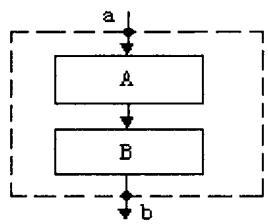

图 1-4　顺序结构

（2）选择结构（又称选取结构）（见图 1-5）

从 a 点进入，并对给出的条件 P 进行判断，如果 P 成立，则执行 A 框操作，否则执行 B 框操作，最后从 b 点脱离该结构。

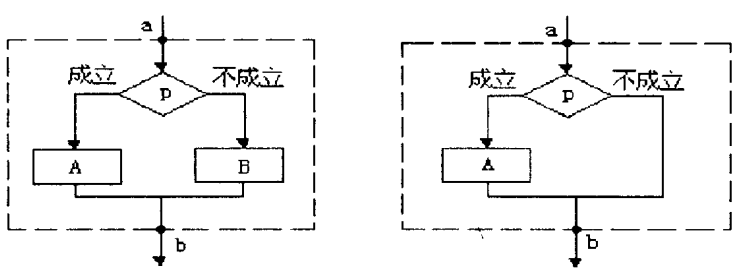

图 1-5　选择结构

需要注意的是：① A 或 B 可有一个为空（即不执行任何操作，称为空操作），如图 1-5

的左图；② 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 。

(3) 循环结构 (又称重复结构)

可分为当型循环结构和直到型循环结构两类

- 当型循环结构（见图 1-6）

从 a 点进入，并对给出的条件 P 进行判断，如果 P 成立，则执行 A 框操作；执行完 A 后重对条件 P 进行判断，如果 P 仍然成立，再次执行 A 框操作；如此反复“判断——执行”，直到条件 P 不成立为止，从 b 点脱离该结构。

- 直到型循环结构（见图 1-7）

从 a 点进入，首先执行 A 框操作，然后对给出的条件 P 进行判断，如果 P 不成立，则执行 A 框操作；执行完 A 后重对条件 P 进行判断，如果 P 仍然不成立，再次执行 A 框操作；如此反复“判断——执行”，直到条件 P 成立为止，从 b 点脱离该结构。

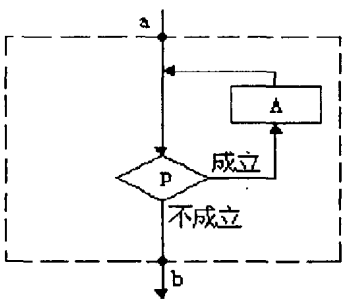

图 1-6 当型循环结构

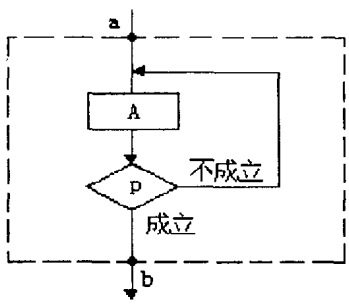

图 1-7 直到型循环结构

- 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，后执行”，而直到型循环是先执行，后判断。

③ 当型循环是给出的条件成立时执行重复操作，而直到型循环是在给出的条件不成立时执行重复操作。

④ 对同一问题，如果分别用当型循环结构和直到型循环结构处理的话，二者结构中的给出条件恰为互逆条件。

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，用直到型循环可解决，反之亦然。

- 以上三种基本结构，有以下共同点：

① 只有一个入口，图 1-4 ~ 图 1-7 中的 a 点为入口点。

② 只有一个出口，图 1-4 ~ 图 1-7 中的 b 点为出口点。

③ 结构内的每一部分都有机会被执行。

④ 结构内不存在“死循环”。

由以上三种基本结构所构成的算法属于“结构化”算法，它不存在无规律的转移，只在本结构内才存在分支和向前或向后的转移。已证明，由这三种基本结构组合而成的算法结构可解决任何复杂问题。