计算机 C/C++语言系列丛书

科学计算的技巧与程序库

C语言数值计算方法

林 君
李文全 编著

学苑出版社

计算机 C/C++语言系列丛书

C 语言

数值计算方法:科学计算的技巧与程序库

林　君　李文全等　编
张　让·审校

学　苑　出　版　社
1993

(京)新登字 151 号

内 容 提 要

C 语言应用日趋广泛，本书以 C 语言为基础，较全面地给出了 C 语言在数值计算领域中的应用与实例。

本书共分为十七章，介绍科学计算基础、线性代数方程组的解法、内插法和外推法、函数的积分、求值、特殊函数、随机数、分类、求根及非线性方程组、求函数的极大值和极小值、本征系统、傅里叶变换谱方法、数据的统计描述、模型化、常微分方程的积分、两点边值问题和偏微分方程。本书中给出了大量的实用程序例子，读者可直接引用。

本书适合于使用 C 语言进行科学计算的工程技术人员、C 语言程序开发人员、计算机应用技术人员、大专院校师生参考使用，也可以作为参考手册或教学参考书。

需购本书的用户，可直接与北京 8721 信箱联系，邮码 100080，电话 2562329

计算机 C/C++语言系列丛书

C 语言

数值计算方法：科学计算的技巧与程序库

编　　著：林　君　李文全等

审　　校：张　让

责任编辑：徐建军

出版发行：学苑出版社　**邮政编码**：100032

社　　址：北京市西城区成方街 33 号

印　　刷：双青印刷厂

开　　本：787×1092　1/16

印　　张：40.3　**字　　数**：941 千字

印　　数：1—5000 册

版　　次：1993 年 12 月北京第 1 版第 1 次

ISBN7—5077—0875—6/TP・24

本册定价：49.00 元

学苑版图书印、装错误可随时退换

前　言

C 语言是一种应用日趋广泛的高级语言，它兼顾了多种高级语言的特点，并具备汇编语言的功能。C 语言程序处理功能强，运算速度快，有良好的移植性，而且可以直接实现对系统硬件及外围接口的控制，具有较强的系统处理能力，已经成为软件开发的一种主要语言。有关 C 语言及其应用方面的书近年来已出版不少，但讨论 C 语言编程技巧、编程语言以及在接口、图形/图象处理等方面的较多，有关科学计算方面的较少，因为传统的 FORTRAN 语言是专门适合于科学计算的语言，流行甚广。C 语言是一种较新的语言，尽管它提供了较强的功能，但人们的重点是放在用 C 语言来实现系统软件与高速实时应用软件的开发方法方面。随着 C 语言的广泛普及，它必将深入到各个领域。数值计算方法技巧是计算机应用的传统领域之一，是工程师、科学家、大专院校师生都要涉及和使用的工具，本书正是为了适应这方面的需要而编写的。

全书共分十七章，分别向读者介绍了科学计算基础、线性代数方程组的解法、内插法和外推法、函数的积分、函数的求值、特殊函数、随机数、分类、求根及非线性方程组、求函数的极大值和极小值、本征系统、傅里叶变换谱方法、数据的统计描述、数据的模型化、常微分方程的积分、两点边值问题和偏微分方程。

本书对上述课题进行了一般讨论、在一定程度上进行了数学分析和算法讨论，更重要的是，以可执行的计算机程序来实现这些思想。

本书的大部分内容可供科学或工程专业作为高年级大学生的数值计算课程，也有一部分内容适合有关专业的研究生及技术人员参考。读者可以通过使用本书来提高自己解决复杂问题的能力，增长才干和经验。即便是没有经验的读者，也能够使用本书提供的大多数先进的程序。当然，希望这些读者能够回过头再学习这些程序内部的奥秘，以便修改它们适应自己特定的需要。

本书假定读者具有数学基础，即具有与物理学或工程学或经济学或定量社会科学有关的数学基础。本书还假定读者具有数值分析或数值方法的任何先前知识。

本书几乎包括了所有数值分析课程的内容。有些章节内容较深入，有些内容仍需读者深入去研究。需要声明的是，虽然本书提供的程序都经过测试，但它们并不是没有错误，甚至不能满足您的特定应用要求；这就需要读者自己去探索，通过本书提供的程序和思想进行扩展，直至满足您的要求为止。

本书由林君、李文全主编。参加本书编写的还有项葵葵、王春民、张世娟、别红霞、秦瑞杰、刘蓉。本书由张让审校。

在本书编写过程中，得到北京希望电脑公司的大力支持和帮助，在此表示衷心的感谢。

由于编者水平所限，再加上时间较紧，书稿虽经过多次审核，缺点和错误在所难免，恳请读者批评指正。

目 录

第一章　预备知识

1.0　引言

本书向读者介绍实际有效的并且内容尽可能广泛的数值计算方法。贯穿本书，我们一直都假定读者具有特定的任务要去完成。根据我们的工作经验来教给读者如何去做。偶尔，我们也可能简洁地指出一种特定的漂亮的解决问题的途径，但是本书的绝大部分我们都将带领读者沿着主要的途径使读者到达预定的目的地。

遍及本书，读者都会发现我们精心的编辑，告诉读者应该做什么而不应该做什么。我们希望读者对本书的这种做法不感到厌倦。不能断言我们的建议总是正确的！相反，在数值计算的教科书编写中，我们反对要讨论每一种已经发现的可能的方法，但不努力去提供一种根据相对的优点进行实际判断的倾向。因此，我们确实为读者提供我们所能裁断的实用方法。伴随着实践经验的增长，对我们的建议是否可靠，读者会有自己的看法。

我们假定读者能够阅读 C 语言计算机程序，在本书中，C 语言是所有的“决窍方法”所使用的语言。如果读者对 FORTRAN 或者对 Pascal 语言更熟悉，可以阅读本书的续篇。在数学和算法内容上与本书相同（故略去，只给出程序）。况且，我们所选取的编程习惯是要强调各语言之间的相似性。如果读者熟悉多种计算机语言，只要掌握任一语言版本就足够了。

文中包含的程序，在印刷上其形式看上去如下：

```
#include <math.h>
#define RAD (3.14159265/180.0)
void flmoon(n,nph,jd,frac)
int n,nph;
long *jd;
float *frac;
```

我们的程序用一个简述它们的目的并且解释它们的调用顺序的介绍开始。这个程序用于计算月相。给定一个整数 n 和一个所希望的月相代码 nph（对一个新月 nph＝0，对第一季度为 1，第二季度为 2，第三季度为 3），本程序返回 Julian Day 数字 jd，以及要加到其上的一天的分数部分 frac，这是从 1900 年 1 月开始计算的第 n 个这种月相。假定为格林威治时间。

```
{
    int i;
    float am,as,c,t,t2,xtra;
    void nrerror();
    c=n+nph/4.0;                    This is how we comment an individual line
    t=c/1236.85;
    t2=t*t;
    as=359.2242+29.105356*c;                You aren't really intended to understand this
    am=306.0253+385.816918*c+0.010730*t2;           algorith but it does work!
```

```
    *jd=2415020+28L*n+7L*nph;
    xtra=0.75933+1.53058868*+((1.178e-4)-(1.55e-7)*t)*t2;
    if(nph==0 ||nph==2)
        xtra+=(0.1734-3.93e-4*t)*sin(RAD*as)-0.4068*sin(RAD*am);
    else if (nph==1||nph==3)
      xtra+=(0.1721-4.0e-4*t)*sin(RAD*as)-0.6280*sin(RAD*am);
    else nrerror("nph is unknown in FLMOON");    This is how we will indicate
    i=(xtra>=0.0 ? (int)floor(stra):(int)ceil(xtra-1.0));                    error conditions
    *jd+=i;
    *frac=xtra-i;
}
```

注意，我们处理所有的误差和例外情况的惯例是用一个像 nrerror("some error message");这样的语句来指出。函数 nrerror 是实用程序 nrutil.C 的一个小文件。nrutil.C 由本书后面的附录 D 列出。这个附录包括许多其它的我们将在后面描述的应用程序。函数 nrerror()在用户的 stderr 设备上打印出指明的错误信息(这个设备通常是终端屏幕)，然后调用函数 exit()，该函数结束执行。函数 exit()存在于我们已检查的每一种 C 函数库中，但是如果用户发现它丢掉了，可以修改 nrerror()以便于它完成任务，否则将终止执行。例如，用户可以使其暂停，以便从键盘输入，然后手工地中断执行。在有些应用场合，用户要修改 nrerror()来完成更复杂的错误处理，例如，要把控制转向某处，用一个错误标志或者错误代码设置。

我们有许多关于 C 语言程序设计要介绍，有关它们的习惯和风格，在 1.1 和 1.2 两节中叙述。

1.0.1 计算环境和程序的效率

我们试图使本书中的程序尽可能地通用，不只是表现在所包括的内容方面，而且还根据不同计算机的可移植性程序。特别是，我们要使所有的程序 都应该在多用户计算机和个人计算机上工作。C 就是有目的地用这种可移植性设计的。然而，我们发现没有用各种编译器来实际检验所有的程序，如在库结构或内容上有所区别的过程，甚至有时在所期望的语法上有所区别时，就没有完全的替代。

我们已经使用了带有 VAX/VMS 和 VAX C 的 DEC VAX；IBM PC，XT，或 AT 带有 MS-DOS(3.2 版)和 Microsoft C(3.0 版或以上)；Sun 3/260 工作站带有 UNIX(4.2BSD)，Sun C 编译器和 UNIX lint C 程序检验器。本书中列出的 C 程序在由 ANSI 标准实现的编译器上进行微小改进就可运行，正如在 Harbison 和 Steele's C 中描述的那样：一个参考手册(我们主要推荐的一本书)。少量的修改之后，我们的程序应该在任意编译器上运行。这种编译器是用更早一些的 de facto"Kernighan and Ritchie"标准实现的。这种微小的不兼容性的一个要查看的例子是 VAX 把内存分配函数 malloc()和 free()看成是该语言的内部函数，而 Microsoft C 把它们放在包含文件 malloc.h 中说明。另一方面，Draft Proposed ANSI C 标准用包含文件 stdlib.h 来包括这些函数。

包含本书中全部程序的磁盘或磁带，实际上是在上述机器上有效的。可以从剑桥大学出版社或数值技巧软件(Cambridge University Press and Numerical Recipes Software)得到(取决于使用的格式)。为了证实程序有效，这里直接从可运行的程序文件中取出程序源代码，这样可减少由于印刷造成的错误。我们用作证实部分的"驱动"或演示程序，单独形成《数值技巧实例

(C)》一书，它也是采用机器可读的形式。如果读者计划使用本书中的多个程序，或者计划使用本书中的程序在几个不同的计算机上运行，那么也许会发现获得这些演示程序是有益的。

当然，要声称我们的程序没有缺陷也是愚蠢的，我们不做这种断言。本C语言版已有两年的读者FORTRAN和Pascal版本的意见与改进的益处。我们已经做得很仔细，但是如果读者发现一个缺陷，请记下它并告诉我们。

本章的余下三节，我们回顾编程的某些基本概念(控制结构等)，讨论在本书中所采用的对C语言的某些特定规则，并介绍数值分析中的某些基本概念(舍入误差等)。然后，我们进入到本书的实质内容。

参考文献及进一步阅读：

Meeus, Jean 1982, Astronomical Formulae for Calculators, Second Edition, revised and enlarged (Richmond, Virginia: Willmann-Bell).

Harbison, Samuel P., and Steele, Guy L., Jr. 1987, C: A Reference Manual, Second Edition (Englewood Cliffs, N. J.: Prentice-Hall).

Kernighan, B., and Ritchie, D 1978, The C Programming Language (Englewood Cliffs, N. J Prentice-Hall).

You will find references listed like this at the end of most sections of this book.

由于计算机算法在出版之前常常需要经过相当一段时间，因此"原始的文献"是区别较大的。我们没有做这方面的努力，而且我们也不自称本书任何程序的完整性。对于一个实际上辅助的文献的论题(在教课书、综述等讨论的)，我们对一些较有用的辅助文献上，尤其是那些对原始文献参考有益的文献有意地限定参考文献。在辅助文献存在较少处，我们给出一些原始参考文献源，这些旨在作为进一步阅读的起点，并不作为该领域的完整目录索引。

每一节末的参考是按顺序安排的，一般地根据它们的作用或者相对于那一节的顺序。对于多于一节的参考也在本书的后面单独列出。

1.1 程序的组织和控制结构

"结构化程序设计"这一术语已变成一个吸引人们注意的术语，它对不同的人来说有不同的含义。有些人不正确地认为结构化程序设计就是"把每一个程序都分为多个子程序并且含有大量的注释"。有些人认为结构化程序设计就是可能用某些计算机语言编程(例如用C)而不是用其它的语言(例如用FORTRAN)，这也是错误的，尽管某些语言进行结构化程序设计的确是相对容易或相对困难些。

有时，我们一方面想指出计算机程序之间的相似性，另一方面又写诗或作曲。所有这三种情况都把本身作为可见的介质在一个纸上或计算机屏幕上二维地表现出来。然而，对这三种情况，可见的、二维的和时间不动的表示要交流或假定要交流某些东西区别很大，这是一个在时间上展开的过程。一首诗意味着要朗读，一首乐曲意味着要演奏，一个程序意味着作为一系列计算机指令来执行。

在所有的三种情况，按视觉直观形式传送的目标是人类。其目的是尽可能有效地实现给人类传递信息，事先理解其过程是如何随时间最大程度地展开。诗歌对人类的目标是读者，音乐

的目标是演奏家，程序设计的目标是程序的用户。

现在，读者也许提出反对，一个程序传递的目标不是给一个人而是给一个计算机，因为程序的使用者仅是一个无关的中介人，一个伺服机器的仆人。也许是这样一种情况，即合法的用户把软盘插入个人计算机并且给该计算机提供一个二进制的可执行的黑盒子程序。此计算机并不关心这个程序是否是“结构化的”。然而，我们想象本书的读者完全不同于这种情况。读者需要或者是想要知道的不只是程序做什么，而且是怎么做，这样才能调整并且修正该程序来完成特定应用。读者需要别人能够了解你已经做了什么，这样他们才能批评或者赞同该程序。在这种情况下，一个程序传递的目标仅是人而不是机器。

我们还没有描绘乐曲、诗歌和程序设计的相似处。事实上，它们都作为人脑信号思维的产物，自然地构筑成具有许多不同层次的统治者。语音(音素)的构成是：小的有意义的单元(词素)依次构成单词，再由单词构成短语，再由短语构成句子，再由句子构成段落，这样组成高层次的意义。注意构成乐曲的情况：由音乐短句构成主旋律，配合旋律，和声等等；它们构成乐章，由乐章构成成协奏曲、交响曲等等。

程序中的结构也有同样的结构层次，如果不是这样普遍承认的话。低层是 ASCII 字符集，然后是常量、标识符、操作数、运算符，然后是程序语句，如：a[j+1]=b+c/3.0;。

在下一层，其术语变成模糊的，没有标准的词汇可使用。语句常常来自“组”或“块”，这些“组”或“块”只是作为一个整体来考虑其意义。例如：

```
swap=a[j];
a[j]=b[j];
b[j]=swap;
```

这对于任一程序员而言，作为两个变量的交换具有直接意义，而

```
ans=sum=0.0;
n=1;
```

很可能是在某些迭代过程之前的变量初始化。在一个程序中的这个统治层常常是易于观察到的。为了清楚起见，程序员在这一层放上注释，即放上“初始化”或“变量交换”的字样。

下一层是控制结构层。这些是 for 循环，if 判断等语句。它们是如此重要，我们在下面将专门介绍。然后，我们具有函数和模块层，以及全体的、要进行计算任务的“全局”组织。在音乐的相似情形下，我们现在是在乐章和完整乐曲的层次。在这个层次的组织是与作曲者或程序员所希望完成的什么紧密相连的，而不是与他们的如何完成的有关，这种情况我们只能提供很少的通用原理(如果有的话)。在这个层次，我们假定读者已经是这方面的专家。

1.1.1 控制结构

一个执行程序随时间展开，但并不是严格地按照其语句所写的顺序运行。影响程序语句执行顺序或者影响程序语句是否执行的程序语句称作控制语句。控制语句本身没有什么意义。它们仅对依次控制的语句组或语句块的上下文起作用。如果读者把那些块作为包含语句的段落的话，那么控制语句也许最好看作为段落的空格和句子之间的标点，而不是句子内的单词。

我们现在可以说明什么是结构化编程的目标。它的目标就是使程序控制在程序的形象化表示中更加清楚。读者要知道这个目标对计算机如何看待这个程序来说是不做任何事情。正如已谈论的那样，计算机不关心用户是否使用结构化程序设计。然而，程序的读者却关注它。一旦用户发现要完善一个结构化良好的程序或调试一个结构化良好的程序比起完善或调试一个

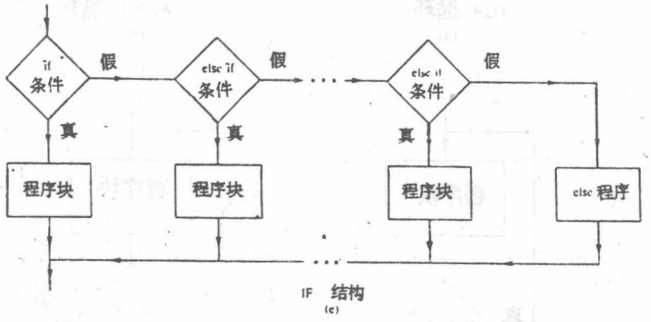
控制结构模糊的程序要容易得多时，用户自己也就会关注结构化程序设计了。

用户以两种基本的方式实现结构化程序设计的目标。首先，用户获取程序中反复发生的必不可少的控制结构的数量，这在大多数程序设计语言中会给出方便的表示方法。用户应该学习考虑你的程序设计任务，尽可能地使用这些标准控制结构。在编写程序中，用户应该养成用一致的、常规的方式来表示这些标准控制结构的习惯。

“这不会限制创造性吗？”我们的学生有时提出这个问题。是的，正如奏鸣曲的形式限制了莫扎特(Mozart)的创造性，或者是十四行诗的韵律限制了莎士比亚(Shakespeare)的创造性。关键在于当要进行交流时，创造性确实受到格式化的某些限制。

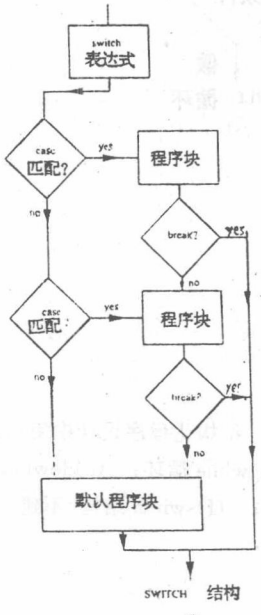
其次，语句的控制块或目标难于一眼辨认出时，要尽可能地避免控制语句。实际上这是指必须试图避免语句的命名标号和 goto 语句。并不是 goto 语句有害(尽管它们确实打断人们的程序阅读)；命名的语句标号是危险的。事实上，当读者阅读一个程序时，无论何时碰到一个命名的语句标号，心里立即会条件反射地有一个想法。为什么？因为习惯上下面的问题会立即浮现在脑海：控制从哪里来到这个标号？它可以在这个程序的任一个地方！一个分支转到这个标号会产生什么结果？其结果难以预测！当然变成不确定性，认识陷入可能性的泥坑。

为了使这些考虑更加具体，现在给出某些例子(见图 1.1.1)。



IF 结构

(e)



SWITCH 结构

(f)

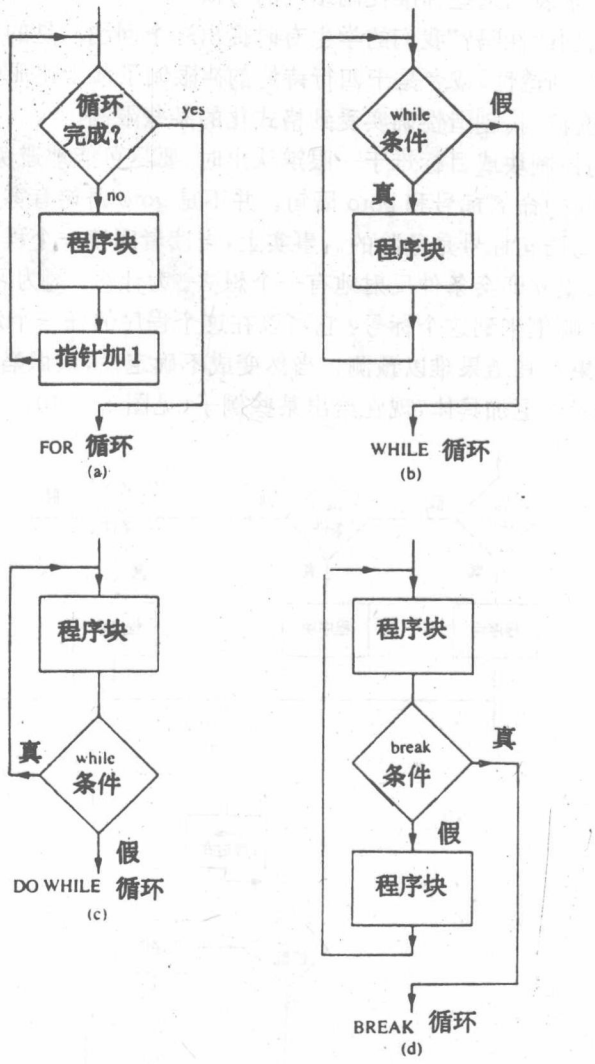


图 1.1.1　结构化程序设计中使用的标准控制结构

(a)for 循环；　(b)while 循环；　(c)dowhile 循环；　(d)break 循环；

(e)if 结构；　(f)switch 结构(不建议使用)

1.1.2 标准结构目录

"For"循环。 C语言中，简单的循环用一个 for 循环执行，例如：

```
for (j=2;j<=1000;j++) {
    b[j]=a[j-1];
    a[j-1]=j;
}
```

注意对于控制结构所执行的代码块，我们是怎样按缩进式排的，留下不缩 进排的结构。还要注意我们习惯上把前花括号放在 for 语句的同一行上，而不是放在下一行。这节省一整行的书写空间，我们的出版商喜欢这种写法。

"If"结构。 C语言中的这个结构类似于 Pascal ，Algol ，FORTRAN 和其它语言中建立的结构，典型的结构如下：

```
if(...){
    ...
}
else if(...){
    ...
else{
    ...
}
```

由于仅当在一个块中多于一个语句时才需要复合语句的花括号，那么C语言中的 if 语句结构比起 FORTRAN 或 Pascal 中对应的结构多少有点不清晰。在建造单个的 if 分句时必须引起某些注意。例如，考虑下面的情况：

```
if (b>3)
if (a>3) b+=1;
else b-=1;          /* questionable! */
```

由于通过使用后续行缩格来判断，这个代码的书写者的目的如下："如果 b 大于 3 且 a 大于 3，则 b 增加 1。如果 b 不大于 3，则 b 减少 1。"然而根据C语言的规则，其实际含义是："如果 b 大于 3，那么 a 小于或者等于 3，则减少 1。"关键问题在于一个 else 分句是与其最近的开放式 if 语句相关联，而不管读者如何把它写到什么地方。这种含义上的混淆可以通过添加花括号来方便地解决。它们在某些情况下也许是技术上多余的，然而，它们清楚地实现读者的目的并且改善读者的程序。上面的程序段应该写为：

```
if (b>3) {
if(a>3) b+=1;
} else }
b-=1;
}
```

下面是一个主要含有 if 控制语句的工作程序：

```
#include <math.h>
#define IGREG(15+31L*(10+12L*1582)) Gregorian Calendar was adopted on Oct. 15,1582
```

```
long julday(mm,id,iyyy)
int mm,id,iyyy;
```

在这个过程中，变量 julday 返回由月份 mm，日 id 和年 iyyy 指定的历法日期的中午开始的儒略(Julian)日期数，mm，id 和 iyyy 都是整型变量。正的年份意指 A. D.（公元），负的年份意指 B. C.（公元前）。记住公元前 1 年之后的年份是公元 1 年。

```
{
long jul;
int ja,jy,jm;
void nrerror();
if (iyyy==0) nrerror("JULDAY:there is no year zero.");
if (iyyy<0) ++iyyy;
if (mm>2){              Here is an example of a block IF-structure.
    jy=iyyy;
    jm=mm+1;
} else {
    jy=iyyy-1;
    jm=mm+13;
}
jul=(long)(floor(365.25*jy)+floor(30.6001*jm)+id+1720995);
if (id+31L*(mm+12L*iyyy)>= YGREG) { Test whether to change to Gregorian Calendar
    ja=0.01*jy;
    jul += 2-ja+(int) (0.25*ja);
}
return jul;
}
```

（天文学家每 24 小时记为一日，开始和结束都在中午，用一个单一的整数，即儒略日期数。儒略日的零是很久以前；一个方便的参考点是儒略日 2440000 开始在 1968 年 5 月 23 日的中午。如果读者知道一个给定日期的中午开始的儒略日期数，那么该日期的星期几可以通过加 1 且获取基于 7 的模数结果来获得；一个零结果对应星期日，1 对应星期一，……，6 对应星期 6。）

“While”循环。 除 FORTRAN 之外，大多数良好的语言都提供了如下 C 语言例子的结构：

```
while (n<1000) {
    n *=2;
    j +=1;
}
```

这个结构的突出特征是控制分支(此情况 n<1000)在每个循环之前进行判断。如果该分支语句非真，则花括号中的语句不会被执行。尤其是，如果当 n 大于 1000 时首次碰到本代码段，则花括号中的语句甚至连一次也不被执行。

“Do-While”循环。 这是与 While 循环相对应的循环控制结构，它在每一次循环结束时

检测它的控制分支。在C中，它看上去如同：

```
do {
    n *=2;
    j +=1;
} while (n<1000);
```

在这种情况下，花括号中的语句至少会执行一次，不管n的初始值如何。

"Break"循环。 在这种情况下，有一个循环，它不确定地重复，直到某些条件变为真，这些条件是在循环圈的中间某处被测试(而且可能在多于一处测试)。在该点希望跳出循环并且进行到其后的程序。在Pascal和标准的FORTRAN中，这种结构需要使用语句标号，这不利于简明的程序设计。在C语言中，这种结构用简单的break语句实现，break中断for，while，do或switch结构内部的执行且进行到后续的指令。中断语句的一个典型用法是：

```
for(;;) {
    [statements before the test]
    if (…)break;
    [statements after the test]
}
[next sequential instruction]
```

这里是一个使用for和break循环结构的程序。为一次打猎，要我们找出一个月的第十三天是星期五，该日子是圆月日。这是一个实现该目标的程序，它还顺便给出了所有其它月的第十三天是星期五的圆月日。

```
include <stdio.h>
include <math.h>
define ZON -5.0                          Time zone-5 is Eastern Standard Time.
define IYBEG 1900                        The range of dates to be searched.
define IYEND 2000
main()                                   /* Program BADLUK */
    int ic,icon,idwk,im,iyyy,n;
    float timzon=ZON/24.0,frac;
    long jd,jday;
    void flmoon();
    long julday();

printf("\nFull moons on Friday the 13th from %5d to %5d\n",IYBEG,IYEND);
for (iyyy=IYBEG;iyyy<=IYEND;iyyy++){     Loop over each year,
    for (im=1;im<=12,im++) {             and each month.
        jday=julday (im,13,iyyy);        is the thirteenth a Friday?
        idwk=(int) ((jday+1) % 7);
        if (idwk==5) {

        n=12.37 * (iyyy-1900+(im-0.5)/12.0);
```

这个n值是从1900年以来已经产生了多少个圆月日的第一次近似值。我们会把它带给此段程序并且调整它向上或向下，直到确定我们所希望的每月第十三天或该日不是一个圆月日。