程图花费了一个月之久，但是对照着流程图专心写程序只需要两天的时间。

话说回来，诸位都善于画流程图吗？是不是有很多人会觉得在流程图中有那么多的符号，在画图时要把这些符号都用上很麻烦呢？

实际上用于表示程序流程的最基础的符号并没有多少。只要先记住表 4.1 中的符号就足够了。就连笔者也很少使用这张表以外的符号。虽然有时也能见到形如显示器或者打印纸的符号，但是可以认为这些只是为了丰富流程图的表现所附加的符号。

只使用表 4.1 中所示的符号，就可以画出程序的三种流程（如图 4.5 所示）。顺序执行只需用直线将矩形框连接起来 (a)。条件分支用菱形表示 (b)。循环的表示方法是通过条件分支回到前面的处理步骤 (c)。这样就能将所有的流程都表示出来了。

作为程序员必须要学会灵活地运用流程图。在思考程序流程的时候，也要首先在头脑中画出流程图。

表 4.1 最低限度所需的流程图符号

符号	含义
	表示流程的开始和结束
	表示处理步骤
	表示条件分支
	用直线把符号连接起来表示流程。在需要明确流程的走向时使用末端带有箭头的直线

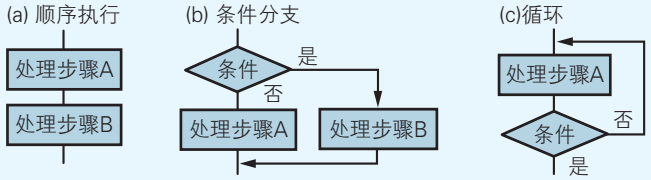

图 4.5　用流程图表示的顺序执行、条件分支、循环三种流程

4.3　表示循环程序块的“帽子”和“短裤”

再继续介绍一些有关流程图的内容吧。如果诸位曾经备考过“信息技术水平考试”，就应该见过用如图 4.6 所示的符号表示循环的流程图。笔者将这一对符号称作“帽子和短裤”（这当然不是正式的名称）。

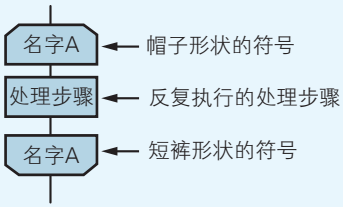

图 4.6　表示循环的符号

对于帽子形状和短裤形状的符号，为了表示它们是成对出现的，要在上面写下适当的名字。然后用“帽子”和“短裤”把需要反复执行的步骤包围起来。如果要在循环中嵌套循环，就需要对每个循环分别使用一对“帽子”和“短裤”。为了区分成对出现的“帽子”和“短裤”，要为每一对起不同的名字。

稍微说一点题外话。笔者的名字是久雄，有一个叫康男的哥哥。洗衣服时，如果把哥哥的帽子和短裤和我的混在一起洗的话，就不知道哪件是哥哥的、哪件是我的了。于是，母亲就在我们哥俩儿的帽子和短裤上分别写上了个人的名字。在流程图的“帽子”和“短裤”符号上写名字也出于同样的目的（如图 4.7 所示）。

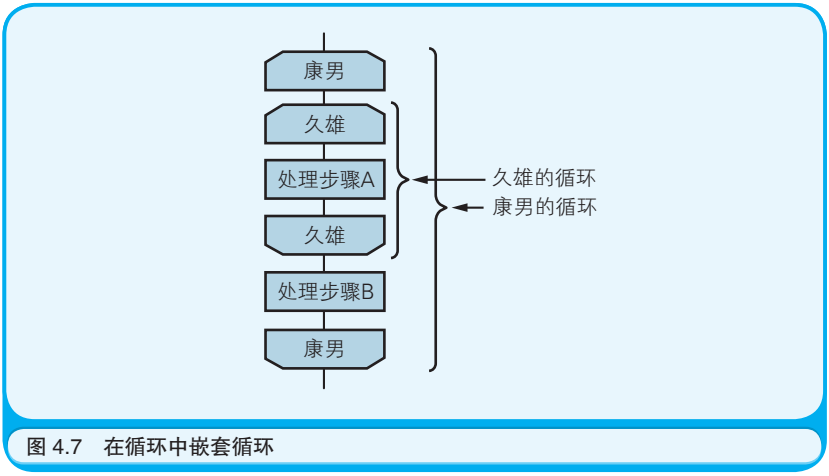

图 4.7 在循环中嵌套循环

上面的内容稍微有点跑题，下面我们回到正题。在计算机硬件上的操作中，循环是通过当满足条件时就返回到之前处理过的步骤来实现的。一旦使用了机器语言或汇编语言所提供的跳转指令，就可以将 PC 寄存器的值设置为任意的内存地址。如果将它的值设为之前执行过的步骤所对应的内存地址，那么就构成了循环。因此，在表示循环的时候，正如图 4.5(c) 所示的那样，仅仅使用带有菱形符号的流程图也就足够了。用机器语言或者汇编语言表示循环时，都是先进行某种比较，再根据比较结果，跳转到之前的地址（如图 4.8 所示）。

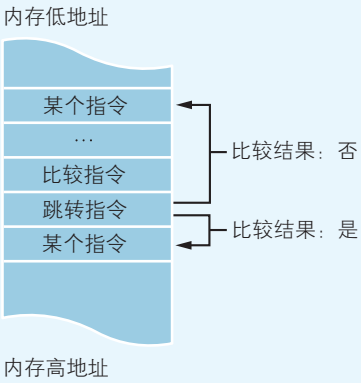

图 4.8　从硬件上看循环的过程

但是，现在还在使用机器语言或汇编语言的人已经不多了。程序员使用的都是能够更加高效地编写程序的高级语言，如 BASIC、C 语言和 Java 等。在这些高级语言中，程序员使用“程序块”表示循环而不是跳转指令。所谓“程序块”就是程序中代码的集合。程序中要被循环处理的部分，就是一种程序块。如图 4.6 所示的用帽子和短裤符号表示循环的方法就适用于使用了程序块的高级语言。

代码清单 4.2 列出了从之前的“石头剪刀布游戏”中摘录出的程序块，这段代码用于循环双方的比试过程。由此可见，在 VBScript 中，是用 For 和 Next 两个关键字表示循环的程序块的。For 对应着“帽子”，Next 则对应着“短裤”。For 的后面写有循环条件。“For i = 1 To 5”表示用变量 i 存储循环次数，将 i 的值从 1 加到 5，每进行 1 次循环就增加 1，如果 i 的值超过了 5 循环就终止。画图时循环条件也要写在“帽子”中（如图 4.9 所示）。

代码清单 4.2 用高级语言表示循环

```
' 进行 5 轮比试
For i = 1 To 5 —— 相当于“帽子”
    ' 处理步骤
    …
Next ———————— 相当于“短裤”
```

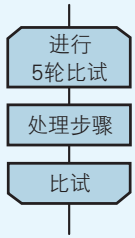

图 4.9 用流程图表示代码清单 4.2 中的程序

用“帽子”和“短裤”表示循环结构没有什么问题，也适用于使用高级语言编写的程序。但是在直接表示硬件操作的机器语言和汇编语言中，是通过条件分支返回到之前处理过的指令来实现循环的，并没有相当于 For 或者 Next 的指令。条件分支本身也是通过跳转指令实现的。根据比较操作的结果，跳转到之前处理过的步骤就是循环；跳转到之后尚未处理的步骤就是条件分支（如图 4.10 所示）。

在高级语言中，条件分支也是由程序块表示的。在 VBScript 中，使用 If、ElseIf、Else、End If 表示条件分支的程序块。通过这几个关键字就可以形成一个被分成三个区域的程序块（如代码清单 4.3 所示）。如果 If 关键字后面所写的条件成立，区域 (1) 中所写的代码就会被执行，形成分支。如果 ElseIf 后面所写的条件成立，区域 (2) 中所写的代码就会被执行，形成分支。当这两个条件都不成立时，区域 (3) 中所写的代码就会被执行，形成分支。高级语言的条件分支代码块，可以用

画有菱形符号的流程图表示。

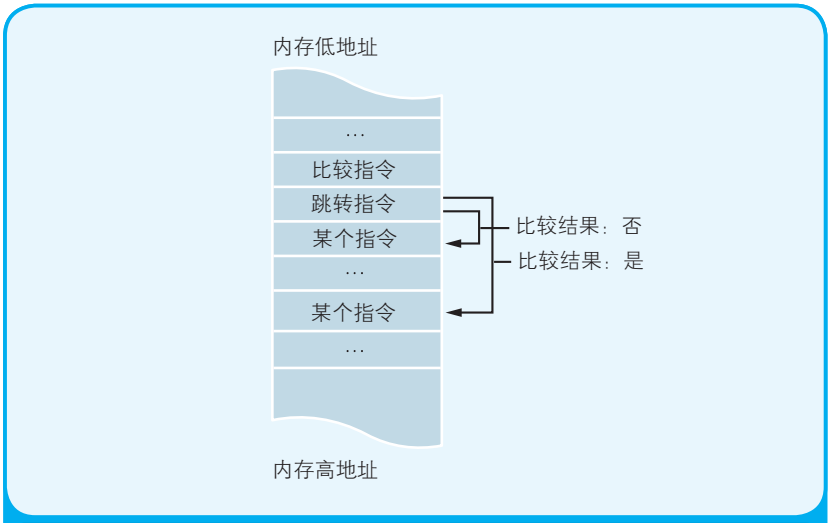

图 4.10　从硬件上看条件分支的过程

代码片段 4.3　用高级语言表示的条件分支

```
' 判定胜负，显示结果
If use = computer Then
    MsgBox s & "... 平局！ "                    区域 (1)
ElseIf computer = (user + 1) Mod 3 Then
    MsgBOx s & "... 玩家获胜！ "                区域 (2)
    wins = wins + 1
Else
    MsgBox s % "... 计算机获胜！ "              区域 (3)
End If
```

4.4　结构化程序设计

既然谈到了程序块，就再介绍一下结构化程序设计吧。诸位即使不曾亲身经历，也应该在什么地方听说过这个词吧。结构化程序设计是由学者戴克斯特拉提倡的一种编程风格。简单地说，所谓结构化程

序设计就是“为了把程序编写得具备结构性，仅使用顺序执行、条件分支和循环表示程序的流程即可，而不再使用跳转指令”。“仅用顺序执行、条件分支和循环表示程序的流程”这一点是不言自明的，需要请诸位注意的是“不使用跳转指令”这一点。

作为计算机硬件上的行为，无论是条件分支还是循环都必须使用跳转指令实现。但是在 VBScript 等高级语言中，可以用 If～ElseIf～Else～End If 程序块表示条件分支，用 For～Next 程序块表示循环。跳转指令因此就变得可有可无了。但是即便如此，在很多高级语言中，还是提供了与机器语言中跳转指令相当的语句，例如 VBScript 中的 GoTo 语句。其实戴克斯特拉想表达的是“既然好不容易使用上了高级语言，就别再使用相当于跳转指令的语句了。即使不使用跳转语句，程序的所有流程仍然可以表述出来”。他这样说是因为跳转指令所带来的危害性不小，会使程序陷入到流程错综复杂的状态，就像意大利面条那样缠绕在一起（如图 4.11 所示）。

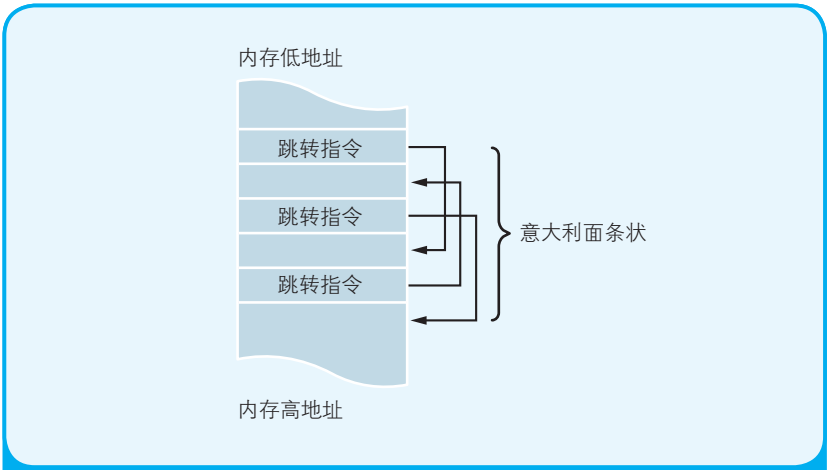

图 4.11　跳转指令（GoTo 语句）使程序陷入“意大利面条”的状态

在程序设计的世界中，如果看到了以“结构化”开头的术语，就可以这样认为：程序的流程是由程序块表示的，而不是用 GoTo 语句等跳转指令实现的。例如，微软的 .NET 框架所提供的新版 BASIC 语言 Visual Basic.NET 中，就以增加新语法的方式加入了被称作“结构化异常处理”的错误处理机制。这里所说的异常类似于错误。

在旧版本的 Visual Basic 中，一旦发生了错误，程序的流程就会跳转到执行错误处理的地方。用程序块来表示这种错误处理方式的机制，就是结构化异常处理。在 Visual Basic.NET 中，用 Try～Catch～End Try 程序块来表示结构化异常处理（如代码清单 4.4 所示）。但是即使使用了结构化异常处理，在硬件上使用的也还是跳转指令，只是说在高级语言中不用再写相当于跳转指令的语句了。如果把用高级语言所编写的程序转换成机器语言，像结构化异常处理这样的语句还是会被转换为跳转指令。

代码清单 4.4　原始的错误处理机制和结构化异常处理的区别

（1）旧版本的 Visual Basic 用跳转指令（GoTo 语句）实现错误处理

```
On Error GoTo ERR_HANDLER
                ← 使用了 GoTo 语句
' 做某些处理
…

' 处理错误的部分          ← 一旦发生了错误，流程就会跳转
ERR_HANDLER:
MsgBox "出错了！"
```

（2）Visual Basic.NET 用程序块实现错误处理

```
Try
    ' 做某些处理
    …

Catch e As Exception      ← 一旦发生了错误，流程就会条转，这一点并没有改变。但是现在这种流程是通过程序块表示，而不再使用 GoTo 语句
    ' 处理错误的部分
    MsgBox "出错了！"
End Try
```

4.5 画流程图来思考算法

为了充分体现流程图的用途，下面稍微涉及一些有关算法的内容。所谓算法（Algorithm），就是解决既定问题的步骤。想让计算机解决问题，就需要把问题的解法转换成程序的流程。

仅用一条语句就能实现出“石头剪刀布游戏”的编程语言是不存在的。如果眼下待解决的问题是如何编写“石头剪刀布游戏”，那么就必须考虑如何把若干条指令组合起来并形成一个解决问题的流程。如果能够想出可以巧妙实现“石头剪刀布游戏”的流程，那么这个问题也就解决了，换言之算法也就实现了。要是诸位被前辈问到：“这个程序的算法是怎样的呢?”那么只要回答清楚程序的流程就可以了。或者画出流程图也是可以的，因为表示程序流程的流程图本身就能解释算法。

思考算法时的要点是要分两步走，先从整体上考虑程序的粗略流程，再考虑程序各个部分细节的流程。有关细节上的流程将在下一章介绍，在这里笔者先介绍粗略的流程。这是一种相当简单的流程，虽然或多或少会有例外，但是几乎所有的程序从整体来看都具有一个一成不变的流程，那就是“初始化处理”→“循环处理”→“收尾处理”。

请试想，用户是怎样使用程序的呢？首先，用户启动了程序（程序执行初始化处理）。接下来用户根据自己的需求操作程序（程序进入循环处理阶段）。最后用户关闭了程序（程序执行收尾处理）。这样的使用方法就可以直接作为程序的整体流程。还是以“石头剪刀布游戏”为例，分出初始化处理、循环处理、收尾处理之后，就可以画出如图4.12那样的粗略的流程图。图中把5次循环处理看作是一个整体，当成是一次处理（用矩形表示）。

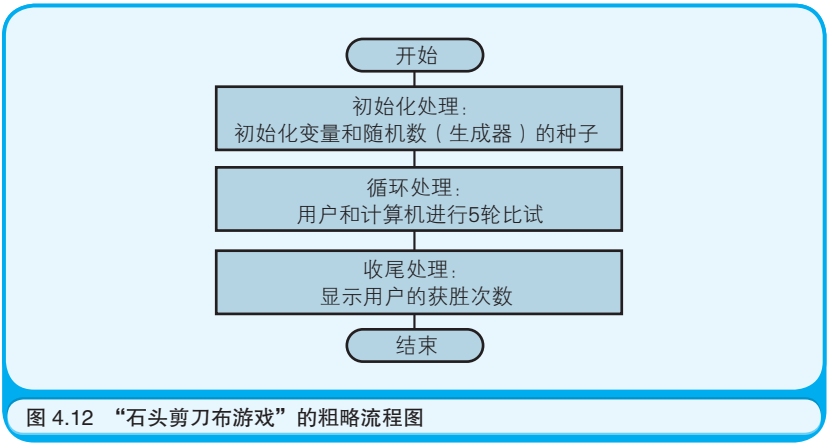

图 4.12　“石头剪刀布游戏”的粗略流程图

反映程序整体流程的粗略流程图还可以用来描述笔者写作本书时的流程（如图 4.13 所示）。首先，启动文字处理机，加载已经写到一半的稿件（初始化处理）。接下来，不断地输入文字（循环处理）。最后，保存稿件（收尾处理）。

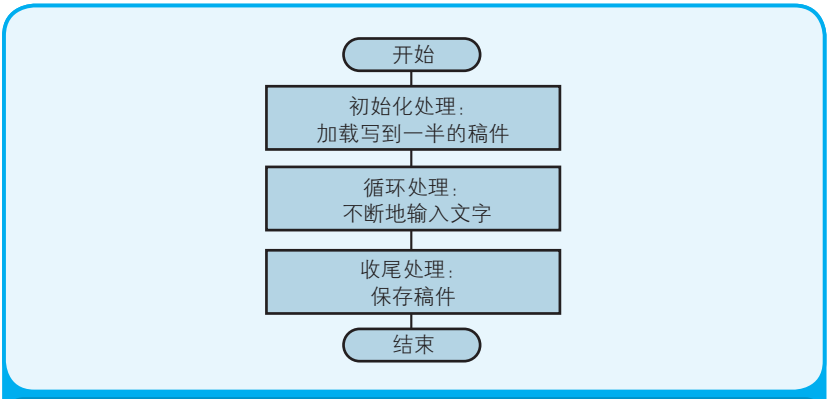

图 4.13　使用“文字处理机”的粗略流程图

我建议那些因为程序没有按照自己的想法来工作而烦恼的人，不