第六章　详细设计

详细设计阶段的根本目标是确定应该怎样具体地实现所要求的系统，也就是说，经过这个阶段的设计工作，应该得出对目标系统的精确描述，从而在编码阶段可以把这个描述直接翻译成用某种程序设计语言书写的程序。

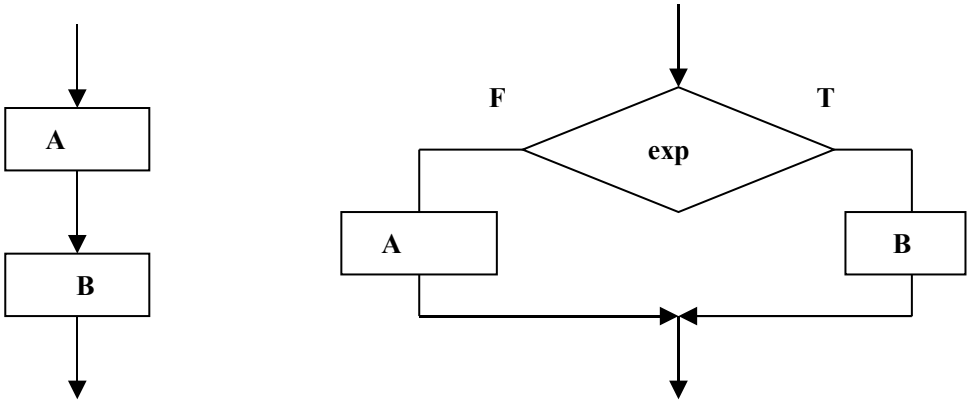
详细设计阶段的任务还不是具体地编写程序，而是要设计出程序的“蓝图”，以后程序员将根据这个蓝图写出实际的程序代码。因此，详细设计的结果基本上决定了最终的程序代码的质量。考虑程序代码的质量时必须注意，程序的“读者”有两个，那就是计算机和人。在软件的生命周期中，设计测试方案、诊断程序错误、修改和改进程序等等都必须首先读懂程序。实际上对于长期使用的软件系统而言，人读程序的时间可能比写程序的时间还要长得多。因此，衡量程序的质量不仅要看它的逻辑是否正确，性能是否满足要求，更主要的是要看它是否容易阅读和理解。详细设计的目标不仅仅是逻辑上正确地实现每个模块的功能，更重要的是设计出的处理过程应该尽可能简明易懂。结构程序设计技术是实现上述目标的关键技术，因此是详细设计的逻辑基础。

6.1 结构程序设计

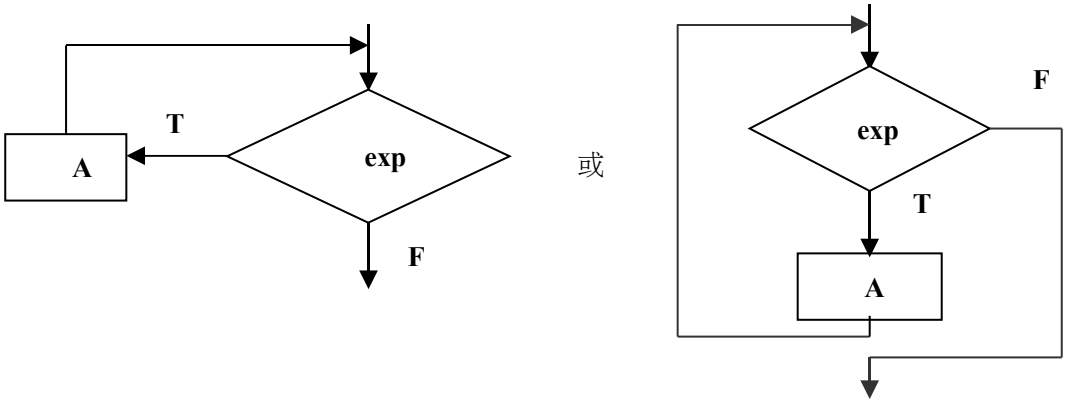
结构程序设计的概念最早由 E.W.D ijkstra 提出。1965 年他在一次会议上指出：“可以从高级语言中取消 GO TO 语句”，“程序的质量与程序中所包含的 GO TO 语句的数量成反比”。1966 年 Bohm 和 Jacopini 证明了，只用 3 种基本的控制结构就能实现任何单入口单出口的程序。这 3 种基
本的控制结构是“顺序”、“选择”和“循环”，它们的流程图分别为图 6.1(a)，6.1(b)和 6.1(c)。

实际上用顺序结构和循环结构(又称 DO_WHILE 结构)完全可以实现选择结构(又称 IF_THEN_ELSE 结构)，因此，理论上最基本的控制结构只有两种。

Bohm 和 Jacopini 的证明给结构程序设计技术奠定了理论基础。



(a)　　（c



(c)

图 6.1 3 种基本的控制结构

（a）顺序结构，先执行 A 再执行 B；（b） IF_THEN_ELSE 型选择（分支）结构；（c） DO_WHILE 型循环结构

1968 年 Dijkstra 再次建议从一切高级语言中取消 GO TO 语句，只使用 3 种基本控制结构写程序。他的建议引起了激烈争论，经过讨论人们认识到，不是简单地去掉 GO TO 语句的问题，而是要创立一种新的程序设计思想、方法和风格，以显著地提高软件生产率和降低软件维护代价。

1971 年 IBM 公司在纽约时报信息库管理系统的设计中成功地使用了结构程序设计技术，随后在美国宇航局空间实验室飞行模拟系统的设计中，结构程序设计技术再次获得圆满成功。这两个系统都相当庞大，前者包含 83000 行高级语言源程序，后者包含 40 万行源程序，而且在设计过程中用户需求又曾有过很多改变，然而两个系统的开发工作都按时并且高质量地完成了。这表明，软件生产率比以前提高了一倍，结构程序设计技术成功地经受了实践的检验。

1972 年 IBM 公司的 Mills 进一步提出，程序应该只有一个入口和一个出口，从而补充了结构程序设计的规则。

那么，什么是结构程序设计呢?

结构程序设计的经典定义如下所述：“如果一个程序的代码块仅仅通过顺序、选择和循环这 3 种基本控制结构进行连接，并且每个代码块只有一个入口和一个出口，则称这个程序是结构化的。”

上述经典定义过于狭隘了，结构程序设计本质上并不是无 GO TO 语句的编程方法，而是一种使程序代码容易阅读、容易理解的编程方法。在多数情况下，无 GO TO 语句的代码确实是容易阅读、容易理解的代码，但是，在某些情况下，为了达到容易阅读和容易理解的目的，反而需要使用 GO TO 语句。例如，当出现了错误条件时，重要的是在数据库崩溃或栈溢出之前，尽可能快地从当前程序转到一个出错处理程序，实现这个目标的最好方法就是使用前向 GO TO 语句(或与之等效的专用语句)，机械地使用 3 种基本控制结构实现这个目标，反而会使程序晦涩难懂。因此，下述的结构程序设计的定义可能更全面一些：

“结构程序设计是尽可能少用 GO TO 语句的程序设计方法。最好仅在检测出错误时才使用 GO TO 语句，而且应该总是使用前向 GO TO 语句。”

虽然从理论上说只用上述 3 种基本控制结构就可以实现任何单入口单出口的程序，

但是为了实际使用方便起见，常常还允许使用 DO_UNTIL 和 DO_CASE 两种控制结构，它们的流程图分别是图 6.2(a)和图 6.2(b)。

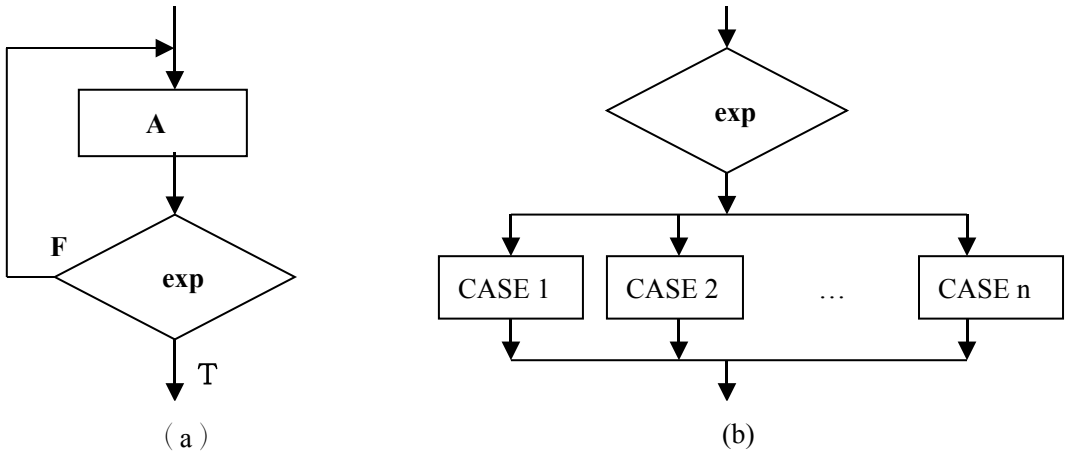


图 6.2 其他常用的控制结构

(a)DO_WHILE 型循环结构； (b) 多分支结构

有时需要立即从循环(甚至嵌套的循环)中转移出来，如果允许使用 LEAVE(或 BREAK)结构，则不仅方便而且会使效率提高很多。LEAVE 或 BREAK 结构实质上是受限制的 GO TO 语句，用于转移到循环结构后面的语句。

如果只允许使用顺序、IF_THEN_ELSE 型分支和 DO_WHILE 型循环这 3 种基本控制结构，则称为经典的结构程序设计；如果除了上述 3 种基本控制结构之外，还允许使用 DO_CASE 型多分支结构和 DO_UNTIL 型循环结构，则称为扩展的结构程序设计；如果再加上允许使用 LEAVE(或 BREAK)结构，则称为修正的结构程序设计。

6.2 人机界面设计

人机界面设计是接口设计的一个重要的组成部分。对于交互式系统来说，人机界面设计和数据设计、体系结构设计及过程设计一样重要。近年来，人机界面在系统中所占的比例越来越大，在个别系统中人机界面的设计工作量甚至占总设计量的一半以上。

人机界面的设计质量，直接影响用户对软件产品的评价，从而影响软件产品的竞争力和寿命因此，必须对人机界面设计给予足够重视。

6.2.1 设计问题

在设计人机界面的过程中，几乎总会遇到下述 4 个问题：系统响应时间、用户帮助设施、出错信息处理和命令交互。不幸的是，许多设计者直到设计过程后期才开始考虑这些问题，这样做往往导致出现不必要的设计反复、项目延期和用户产生挫折感。最好在设计初期就把这些问题作为重要的设计问题来考虑，这时修改比较容易，代价也低。下面讨论这 4 个设计问题。

1. 系统响应时间

系统响应时间是许多交互式系统用户经常抱怨的问题。一般说来，系统响应时间指从用户完成某个控制动作(例如，按回车键或点击鼠标)，到软件给出预期的响应(输出信息或做动作)之间的这段时间。

系统响应时间有两个重要属性，分别是长度和易变性。如果系统响应时间过长，用户就会感到紧张和沮丧。但是，当用户工作速度是由人机界面决定的时候，系统响应时间过短也不好，这会迫使用户加快操作节奏，从而可能会犯错误。

易变性指系统响应时间相对于平均响应时间的偏差，在许多情况下，这是系统响应时间的更重要的属性。即使系统响应时间较长，响应时间易变性低也有助于用户建立起稳定的工作节奏。例如，稳定在 1 秒的响应时间比从 0.1 秒到 2.5 秒变化的响应时间要好。用户往往比较敏感，他们总是担心响应时间变化暗示系统工作出现了异常。

2.用户帮助设施

几乎交互式系统的每个用户都需要帮助，当遇到复杂问题时甚至需要查看用户手册以寻找答案。大多数现代软件都提供联机帮助设施，这使得用户无须离开用户界面就能解决自己的问题。

常见的帮助设施可分为集成的和附加的两类。集成的帮助设施从一开始就设计在软件里面，通常，它对用户工作内容是敏感的，因此用户可以从与刚刚完成的操作有关的主题中选择一个请求帮助。显然，这可以缩短用户获得帮助的时间，增加界面的友好性。附加的帮助设施是在系统建成后再添加到软件中的，在多数情况下它实际上是一种查询能力有限的联机用户手册，人们普遍认为，集成的帮助设施优于附加的帮助设施。

具体设计帮助设施时，必须解决下述的一系列问题。

(1)在用户与系统交互期间，是否在任何时候都能获得关于系统任何功能的帮助信息?有两种选择：提供部分功能的帮助信息和提供全部功能的帮助信息。

(2)用户怎样请求帮助? 有 3 种选择：帮助菜单，特殊功能键和 HELP 命令。

(3)怎样显示帮助信息? 有 3 种选择：在独立的窗口中，指出参考某个文档(不理想)和在屏幕固定位置显示简短提示。

(4)用户怎样返回到正常的交互方式中?有两种选择：屏幕上的返回按钮和功能键。

(5)怎样组织帮助信息?有 3 种选择：平面结构(所有信息都通过关键字访问)，信息的层次结构(用户可在该结构中查到更详细的信息)和超文本结构。

3.出错信息处理

出错信息和警告信息，是出现问题时交互式系统给出的“坏消息”。出错信息设计得不好，将向用户提供无用的甚至误导的信息，反而会加重用户的挫折感。

一般说来，交互式系统给出的出错信息或警告信息，应该具有下述属性。

(1)信息应该用用户可以理解的术语描述问题。

(2)信息应该提供有助于从错误中恢复的建设性意见。

(3)信息应该指出错误可能导致哪些负面后果(例如，破坏数据文件)，以便用户检查是否出现了这些问题，并在确实出现问题时及时解决。

(4)信息应该伴随着听觉上或视觉上的提示，例如，在显示信息时同时发出警告铃声，或者信息用闪烁方式显示，或者信息用明显表示出错的颜色显示。

(5)信息不能带有指责色彩，也就是说，不能责怪用户。

当确实出现了问题的时候，有效的出错信息能提高交互式系统的质量，减轻用户的挫折感。

4.命令交互

命令行曾经是用户和系统软件交互的最常用的方式，并且也曾经广泛地用于各种应用软件中。现在，面向窗口的、点击和拾取方式的界面已经减少了用户对命令行的依赖，但是，许多高级用户仍然偏爱面向命令行的交互方式。在多数情况下，用户既可以从菜单中选择软件功能，也可以通过键盘命令序列调用软件功能。

在提供命令交互方式时，必须考虑下列设计问题。

(1)是否每个菜单选项都有对应的命令?

(2)采用何种命令形式?有 3 种选择：控制序列(例如，Ctrl+P)，功能键和键入命令。

(3)学习和记忆命令的难度有多大?忘记了命令怎么办?

(4)用户是否可以定制或缩写命令?

在越来越多的应用软件中，人机界面设计者都提供了“命令宏机制”，利用这种机制用户可以用自己定义的名字代表一个常用的命令序列。需要使用这个命令序列时，用户无须依次键入每个命令，只需输入命令宏的名字就可以顺序执行它所代表的全部命令。

在理想的情况下，所有应用软件都有一致的命令使用方法。如果在一个应用软件中命令 Ctrl+D 表示复制一个图形对象，而在另一个应用软件中 Ctrl+D 命令的含义是删除一个图形对象，显然会使用户感到困惑，并且往往会导致用错命令。

6.2.2 设计过程

用户界面设计是一个迭代的过程，也就是说，通常先创建设计模型，再用原型实现这个设计模型，并由用户试用和评估，然后根据用户意见进行修改。为了支持上述迭代过程，各种用于界面设计和原型开发的软件工具应运而生。这些工具被称为用户界面工具箱或用户界面开发系统，它们为简化窗口、菜单、设备交互、出错信息、命令及交互环境的许多其他元素的创建，提供了各种例程或对象。这些工具所提供的功能，既可以用基于语言的方式也可以用基于图形的方式来实现。

一旦建立起用户界面的原型，就必须对它进行评估，以确定其是否满足用户的需求。评估可以是非正式的，例如，用户即兴发表一些反馈意见；评估也可以十分正式，例如，运用统计学方法评价全体终端用户填写的调查表。

用户界面的评估周期如下所述：完成初步设计之后就创建第一级原型；用户试用并评估该原型，直接向设计者表述对界面的评价；设计者根据用户意见修改设计并实现下一级原型。上述评估过程持续进行下去，直到用户感到满意，不需要再修改界面设计时为止。

当然，也可以在创建原型之前就对用户界面的设计质量进行初步评估。如果能及早发现并改正潜在的问题，就可以减少评估周期的执行次数，从而缩短软件的开发时间。在创建了用户界面的设计模型之后，可以运用下述评估标准对设计进行早期复审。

(1)系统及其界面的规格说明书的长度和复杂程度，预示了用户学习使用该系统所需要的工作量。

(2)命令或动作的数量、命令的平均参数个数或动作中单个操作的个数，预示了系统的交互时间和总体效率。

(3)设计模型中包含的动作、命令和系统状态的数量，预示了用户学习使用该系统时需要记忆

的内容的多少。

(4)界面风格、帮助设施和出错处理协议，预示了界面的复杂程度及用户接受该界面的程度。

6.2.3 人机界面设计指南

用户界面设计主要依靠设计者的经验。总结众多设计者的经验得出的设计指南，有助于设计者设计出友好、高效的人机界面。下面介绍 3 类人机界面设计指南。

1.一般交互指南

一般交互指南涉及信息显示、数据输入和系统整体控制，因此，这类指南是全局性的，忽略它们将承担较大风险。下面讲述一般交互指南。

(1)保持一致性。应该为人机界面中的菜单选择、命令输入、数据显示以及众多的其他功能，使用一致的格式。

(2)提供有意义的反馈。应向用户提供视觉的和听觉的反馈，以保证在用户和系统之间建立双向通信。

(3)在执行有较大破坏性的动作之前要求用户确认。如果用户要删除一个文件，或覆盖一些重要信息，或终止一个程序的运行，应该给出“您是否确实要……”的信息，以请求用户确认他的命令。

(4)允许取消绝大多数操作。UNDO 或 REVERSE 功能曾经使众多终端用户避免了大量时间浪费。每个交互式系统都应该能方便地取消已完成的操作。

(5)减少在两次操作之间必须记忆的信息量。不应该期望用户能记住在下一步操作中需使用的一大串数字或标识符。应该尽量减少记忆量。

(6)提高对话、移动和思考的效率。应该尽量减少用户击键的次数，设计屏幕布局时应该考虑尽量减少鼠标移动的距离，应该尽量避免出现用户问“这是什么意思?”的情况。

(7)允许犯错误。系统应该能保护自己不受严重错误的破坏。

(8)按功能对动作分类，并据此设计屏幕布局。下拉菜单的一个主要优点就是能按动作类型组织命令。实际上，设计者应该尽力提高命令和动作组织的“内聚性”。

(9)提供对用户工作内容敏感的帮助设施(参见 6.2.1 节)。

(10)用简单动词或动词短语作为命令名。过长的命令名难于识别和记忆，也会占用过多的菜单空间。

2.信息显示指南

如果人机界面显示的信息是不完整的、含糊的或难于理解的，则该应用系统显然不能满足用户的需求。可以用多种不同方式“显示”信息：用文字、图形和声音；按位置、移动和大小；使用颜色、分辨率和省略。下面是关于信息显示的设计指南。

(1)只显示与当前工作内容有关的信息。用户在获得有关系统的特定功能的信息时，不必看到与之无关的数据、菜单和图形。

(2)不要用数据淹没用户，应该用便于用户迅速吸取信息的方式来表示数据。例如，可以用图形或图表来取代庞大的表格。

(3)使用一致的标记、标准的缩写和可预知的颜色。显示的含义应该非常明确，用户无须参照其他信息源就能理解。

(4)允许用户保持可视化的语境。如果对所显示的图形进行缩放，原始的图像应该一直显示着(以缩小的形式放在显示屏的一角)，以使用户知道当前看到的图像部分在原图中所处的相对位置。

(5)产生有意义的出错信息(参见 6.2.1 节)。

(6)使用大小写、缩进和文本分组以帮助理解。人机界面显示的信息大部分是文字
文字的布局和形式对用户从中提取信息的难易程度有很大影响。

(7)使用窗口分隔不同类型的信息。利用窗口用户能够方便地“保存”多种不同类型的信息。

(8)使用“模拟”显示方式表示信息，以使信息更容易被用户提取。例如，显示炼油厂储油罐的压力时，如果简单地用数字表示压力，则不易引起用户注意。但是，如果用类似温度计的形式来表示压力，用垂直移动和颜色变化来指示危险的压力状况，就容易引起用户的警觉，因为这样做为用户提供了绝对和相对两方面的信息。

(9)高效率地使用显示屏。当使用多窗口时，应该有足够的空间使得每个窗口至少都能显示出一部分。此外，屏幕大小应该选得和应用系统的类型相配套(这实际上是一个系统工程问题)。

3.数据输入指南

用户的大部分时间用在选择命令、键入数据和向系统提供输入。在许多应用系统中键盘仍然是主要的输入介质，但是，鼠标、数字化仪和语音识别系统正迅速地成为重要的输入手段。下面是关于数据输入的设计指南。

(1)尽量减少用户的输入动作。最重要的是减少击键次数，这可以用下列方法实现：用鼠标从预定义的一组输入中选一个；用“滑动标尺”在给定的值域中指定输入值；利用宏把一次击键转变成更复杂的输入数据集合。

(2)保持信息显示和数据输入之间的一致性。显示的视觉特征(例如，文字大小、颜色和位置)应该与输入域一致。

(3)允许用户自定义输入。专家级的用户可能希望定义自己专用的命令或略去某些类型的警告信息和动作确认，人机界面应该为用户提供这样做的机制。

(4)交互应该是灵活的，并且可调整成用户最喜欢的输入方式。用户类型与喜好的输入方式有关，例如，秘书可能非常喜欢键盘输入，而经理可能更喜欢使用鼠标之类的点击设备。

(5)使在当前动作语境中不适用的命令不起作用。这可使得用户不去做那些肯定会导致错误的动作。

(6)让用户控制交互流。用户应该能够跳过不必要的动作，改变所需做的动作的顺序(在应用环境允许的前提下)，以及在不退出程序的情况下从错误状态中恢复正常。

(7)对所有输入动作都提供帮助(参见 6.2.1 节)。

(8)消除冗余的输入。除非可能发生误解，否则不要要求用户指定输入数据的单位；尽可能提供默认值；绝对不要要求用户提供程序可以自动获得或计算出来的信息。

6.3 过程设计的工具

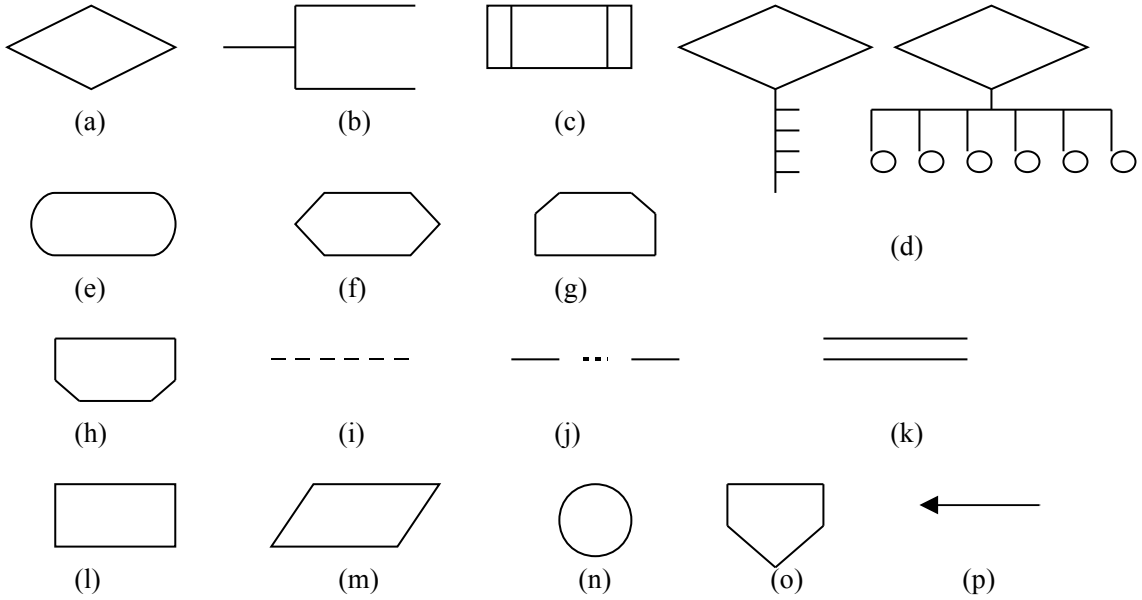
描述程序处理过程的工具称为过程设计的工具，它们可以分为图形、表格和语言 3 类。不论是哪类工具，对它们的基本要求都是能提供对设计的无歧义的描述，也就是应该能指明控制流程、处理功能、数据组织以及其他方面的实现细节，从而在编码阶段能把对设计的描述直接翻译成程序代码。

6.3.1 程序流程图

程序流程图又称为程序框图，它是历史最悠久、使用最广泛的描述过程设计的方法，然而它

也是用得最混乱的一种方法。

在 6.1 节中已经用程序流程图描绘了一些常用的控制结构，相信读者对程序流程图中使用的基本符号已经有了一些了解。图 6.3 中列出了程序流程图中使用的各种符号。



(a)选择(分支);(b)注释;(c)预先定义过程;(d)并行方式;(e)终端;(f)输入输出;(g)循环上界限;(h)循环下界限;(i)虚线;(j)省略符;(k)并行方式;(l)处理;(m)数据;(n)连接;(o)换页连接;(p)控制流

图 6.3 程序流程图中使用的符号

从 20 世纪 40 年代末到 70 年代中期，程序流程图一直是软件设计的主要工具。它的主要优点是对控制流程的描绘很直观，便于初学者掌握。由于程序流程图历史悠久，为最广泛的人所熟悉，尽管它有种种缺点，许多人建议停止使用它，但至今仍在广泛使用着。不过总的趋势是越来越多的人不再使用程序流程图了。

程序流程图的主要缺点如下：

(1)程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

(2)程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制。

(3)程序流程图不易表示数据结构。

6.3.2 盒图(N--S 图)

出于要有一种不允许违背结构程序设计精神的图形工具的考虑，Nassi 和 Shneiderman 提出了盒图，又称为 N-S 图。它有下述特点：

(1)功能域(即，个特定控制结构的作用域)明确，可以从盒图上一眼就看出来。

(2)不可能任意转移控制。

(3)很容易确定局部和全程数据的作用域。

(4)很容易表现嵌套关系，也可以表示模块的层次结构。

图 6.4 给出了结构化控制结构的盒图表示，也给出了调用子程序的盒图表示方法：

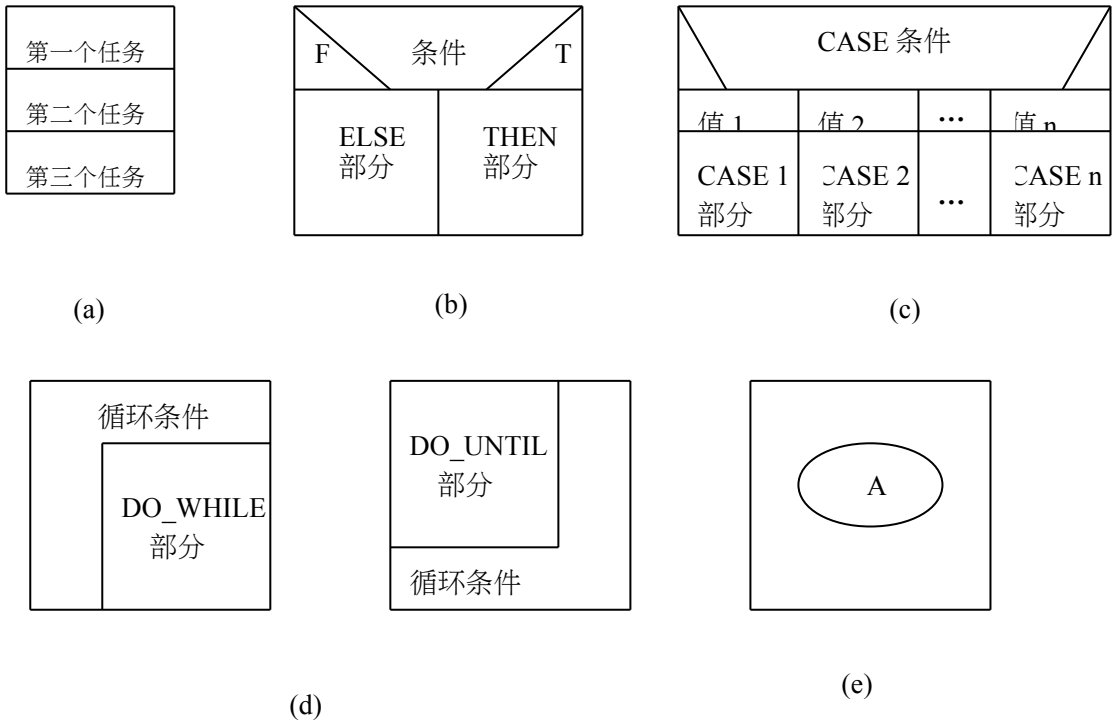


图 6.4 盒图的基本符号

盒图没有 (a) 顺序 ;(b)I P1 N_ELSE 型分支 ;(c)CASE 型多分支 ;(d) 循环 ;(e) 调用子程序 A 「以使程序员逐步养成用结构化的方式思考问题和解决问题的习惯。

P2

WHILE C P

6.3.3 PAD 图

(a) (d)

P1

PAD 是问题分析图 C diagram)的英文缩 年由 公司发明以后，已得到一定程度的推广 已用于形结构的图来表示程序的控制流，将这种图翻译成程序代码比较容易。图 6.5 给出 的基本符号。

P2

UNTIL C P

(b) (e)

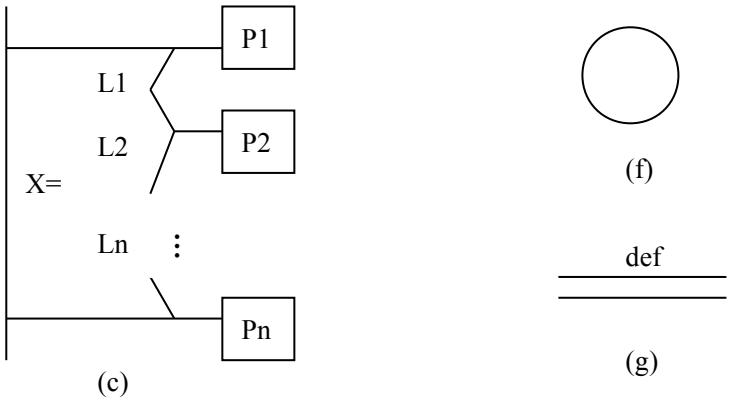


图 6.5 PAD 图的基本符号

(a) 顺序 (先执行 P1 后执行 P2);(b) 选择 (IF C THEN P1 ELSE P2);(c)CASE 型多分支 ;(d)WHILE 型循环 (WHILE C DO P);(e)UNTIL 型循环 (REPEAT P UNTIL C);(f) 语句标号 ;(g) 定义

PAD 图的主要优点如下：

(1)使用表示结构化控制结构的 PAD 符号所设计出来的程序必然是结构化程序。

(2)PAD 图所描绘的程序结构十分清晰。图中最左面的竖线是程序的主线，即第一层结构。随着程序层次的增加，PAD 图逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线。PAD 图中竖线的总条数就是程序的层次数。

(3)用 PAD 图表现程序逻辑，易读、易懂、易记。PAD 图是二维树形结构的图形，程序从图中最左竖线上端的结点开始执行，自上而下，从左向右顺序执行，遍历所有结点。

(4)容易将 PAD 图转换成高级语言源程序，这种转换可用软件工具自动完成，从而可省去人工编码的工作，有利于提高软件可靠性和软件生产率。

(5)即可用于表示程序逻辑，也可用于描绘数据结构。

(6)PAD 图的符号支持.自顶向下、逐步求精方法的使用。开始时设计者可以定义一个抽象的程序，随着设计工作的深入而使用 def 符号逐步增加细节，直至完成详细设计，如图 6.6 所示。

PAD 图是面向高级程序设计语言的，为 FORTRAN，COBOL 和 PASCAL 等每种常用的高级程序设计语言都提供了一整套相应的图形符号。由于每种控制语句都有一个图形符号与之对应，显然将 PAD 图转换成与之对应的高级语言程序比较容易。

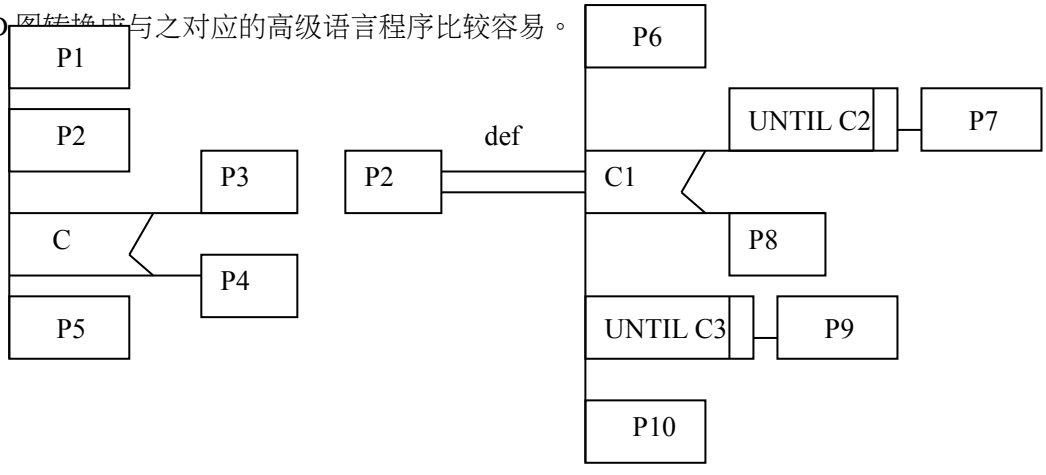


图 6.6　使用 PAD 图提供的定义功能来逐步求精的例子

(a) 初始的 PAD 图 ;(b) 使用 def 符号细化处理框

6.3.4 判定表

当算法中包含多重嵌套的条件选择时，用程序流程图、盒图、PAD 图或后面即将介绍的过程设计语言(PDL)都不易清楚地描述 ◦然而判定表却能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

一张判定表由 4 部分组成，左上部列出所有条件，左下部是所有可能做的动作，右上部是表示各种条件组合的一个矩阵，右下部是和每种条件组合相对应的动作。判定表右半部的每一列实质上是一条规则，规定了与特定的条件组合相对应的动作。

下面以行李托运费的算法为例说明判定表的组织方法。假设某航空公司规定，乘客可以免费托运重量不超过 30kg 的行李。当行李重量超过 30kg 时，对头等舱的国内乘客超重部分每公斤收费 4 元，对其他舱的国内乘客超重部分每公斤收费 6 元，对外国乘客超重部分每公斤收费比国内乘客多一倍，对残疾乘客超重部分每公斤收费比正常乘客少一半。用判定表可以清楚地表示与上述每种条件组合相对应的计算行李费的算法，如表 6.1 所示。

表 6.1 用 判定表表示计算行李费的算法

	规 则								
	1	2	3	4	5	6	7	8	9
国内乘客		T	T	T	T	F	F	F	F
头等舱		T	F	T	F	T	F	T	F
残疾乘客		F	F	T	T	F	F	T	T
行李重量 W≤30kg	T	F	F	F	F	F	F	F	F
免费	╳								

（W-30）×2				X					
（W-30）×3					X				
（W-30）×4		X						X	
（W-30）×6			X						X
（W-30）×8						X			
（W-30）×12							X		

在表的右上部分中“T”表示它左边那个条件成立，“F”表示条件不成立，空白表示这个条件成立与否并不影响对动作的选择。判定表右下部分中画“X”表示做它左边的那项动作(在本例中就是用该公式计算行李费)，空白表示不做这项动作。从表 6.1 可以看出，只要行李重量不超过 30kg，不论这位乘客持有何种机票，是中国人还是外国人，是残疾人还是正常人，一律免收行李费，这就是表右部第一列(规则 1)表示的内容。当行李重量超过 30kg 时，根据乘客机票的等级、乘客国籍及是否残疾人而使用不同算法计算行李费，这就是从规则 2 到规则 9 所表示的内容。

从上面这个例子可以看出，判定表能够简洁而又无歧义地描述处理规则。当把判定表和布尔代数或卡诺图结合起来使用时，可以对判定表进行校验或化简。但是，判定表并不适于作为一种通用的设计工具，没有一种简单的方法使它能同时清晰地表示顺序和重复等处理特性。

6.3.5 判定树

判定表虽然能清晰地表示复杂的条件组合与应做的动作之间的对应关系，但其含义却不是一眼就能看出来的，初次接触这种工具的人理解它需要有一个简短的学习过程。此外，当数据元素的值多于两个时(例如，6.3.4 例子中假设对机票需细分为头等舱、二等舱和经济舱等多种级别时)，判定表的简洁程度也将下降。

判定树是判定表的变种，也能清晰地表示复杂的条件组合与应做的动作之间的对应关系。判定树的优点在于，它的形式简单到不需任何说明，一眼就可以看出其含义，因此易于掌握和使用。多年来判定树一直受到人们的重视，是一种比较常用的系统分析和设计的工具。图 6.7 是和表 6.1 等价的判定树。从图 6.7 可以看出，虽然判定树比判定表更直观，但简洁性却不如判定表，数据元素的同一个值往往要重复写多遍，而且越接近树的叶端重复次数越多。此外还可以看出，画判定树时分枝的次序可能对最终画出的判定树的简洁程度有较大影响，在这个例子中如果不是把行李重量做为第一个分枝，而是将它作为最后一个分枝，则画出的判定树将有 16 片树叶而不是只有 9 片树叶。显然判定表并不存在这样的问题。

6.3.6 过程设计语言

过程设计语言(PDL)也称为伪码，这是一个笼统的名称，现在有许多种不同的过程设计语言在使用。它是用正文形式表示数据和处理过程的设计工具。

PDL 具有严格的关键字外部语法，用于定义控制结构和数据结构；另一方面，PDL 表示实际操作和条件的内部语法通常又是灵活自由的，可以适应各种工程项目的需要。因此，一般说来，PDL 是一种“混杂”语言，它使用一种语言(通常是某种自然语言)的词汇，同时却使用另一种语言(某种结构化的程序设计语言)的语法。

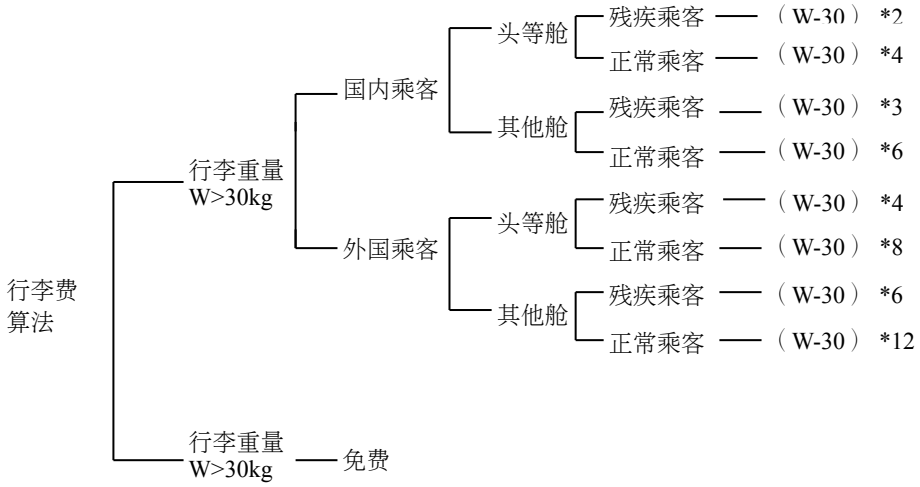


图 6.7 用判定树表示计算行李的算法

PDL 应该具有下述特点：

(1)关键字的固定语法，它提供了结构化控制结构、数据说明和模块化的特点。为了使结构清晰和可读性好，通常在所有可能嵌套使用的控制结构的头和尾都有关键字，例如，if...fi(或 endif)等等。

(2)自然语言的自由语法，它描述处理特点。

(3)数据说明的手段。应该既包括简单的数据结构(例如纯量和数组)，又包括复杂的数据结构(例如，链表或层次的数据结构)。

(4)模块定义和调用的技术，应该提供各种接口描述模式。

PDL 作为一种设计工具有如下一些优点：

(1)可以作为注释直接插在源程序中间。这样做能促使维护人员在修改程序代码的同时也相应地修改 PDL 注释，因此有助于保持文档和程序的一致性，提高了文档的质量。

(2)可以使用普通的正文编辑程序或文字处理系统，很方便地完成 PDL 的书写和编辑工作。

(3)已经有自动处理程序存在，而且可以自动由 PDL 生成程序代码。

PDL 的缺点是不如图形工具形象直观，描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单。

6.4 面向数据结构的设计方法

计算机软件本质上是信息处理系统，因此，可以根据软件所处理的信息的特征来设计软件。第5章曾经介绍了面向数据流的设计方法，也就是根据数据流确定软件结构的方法，本节介绍面向数据结构的设计方法，也就是根据数据结构设计程序处理过程的方法。

在许多应用领域中信息都有清楚的层次结构，输入数据、内部存储的信息(数据库或文件)以及输出数据都可能有独特的结构。数据结构既影响程序的结构又影响程序的处理过程，重复出现的数据通常由具有循环控制结构的程序来处理，选择数据(即，可能出现也可能不出现的信息)要用带有分支控制结构的程序来处理。层次的数据组织通常和使用这些数据的程序的层次结构十分相似。

面向数据结构的设计方法的最终目标是得出对程序处理过程的描述。这种设计方法并不明显地使用软件结构的概念，模块是设计过程的副产品，对于模块独立原理也没有给予应有的重视。因此，这种方法最适合于在详细设计阶段使用，也就是说，在完成了软件结构设计之后，可以使用面向数据结构的方法来设计每个模块的处理过程。

Jackson方法和Warnier方法是最著名的两个面向数据结构的设计方法，本节结合一个简单例子扼要地介绍Jackson方法，目的是使读者对面向数据结构的设计方法有初步了解。

使用面向数据结构的设计方法，当然首先需要分析确定数据结构，并且用适当的工具清晰地描绘数据结构。本节先介绍Jackson方法的工具——Jackson图，然后介绍Jackson程序设计方法的基本步骤。

6.4.1 Jackson图

虽然程序中实际使用的数据结构种类繁多，但是它们的数据元素彼此间的逻辑关系却只有顺序、选择和重复3类，因此，逻辑数据结构也只有这3类。

1.顺序结构

顺序结构的数据由一个或多个数据元素组成，每个元素按确定次序出现一次。图6.8是表示顺序结构的Jackson图的一个例子。

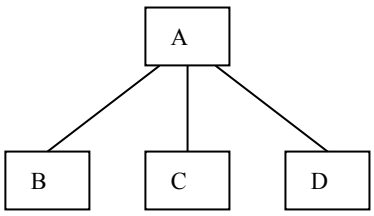


图6.8 A由BCD3个元素顺序组成
（每个元素只出现一次，出现的次序依次是BCH和D）

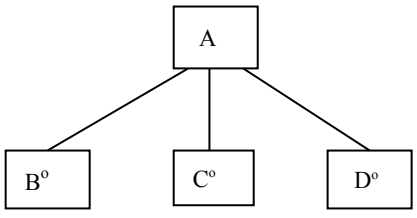


图6.9 根据条件A是B或C或D中某一个
（注意：在B，C和D的右上角有小圆圈做记号）

2.选择结构

选择结构的数据包含两个或多个数据元素，每次使用这个数据时按一定条件从这些数据元素中选择一个。图6.9是表示3个中选1个结构的Jackson图。

3.重复结构

重复结构的数据，根据使用时的条件由一个数据元素出现零次或多次构成是表示重复结构的Jackson 图。

Jackson 图有下述优点：

·便于表示层次结构，而且是对结构进行自顶向下分解的有力工具；

·形象直观可读性好；

·既能表示数据结构也能表示程序结构(因为结构程序设计也只使用上述 3 种基本控制结构)。

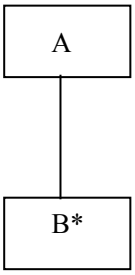


图 6.10 A 由 B 出现 N 次（N≥0）组成

（注意：在 B 的右上角有星号标记）

6.4.2 改进的 Jackson 图

上一小节介绍的 Jackson 图的缺点是，用这种图形工具表示选择或重复结构时，选择条件或循环结束条件不能直接在图上表示出来，影响了图的表达能力，也不易直接把图翻译成程序，此外，框间连线为斜线，不易在行式打印机上输出。为了解决上述问题，本书建议使用图 6.11 中给出的改进的 Jackson 图。

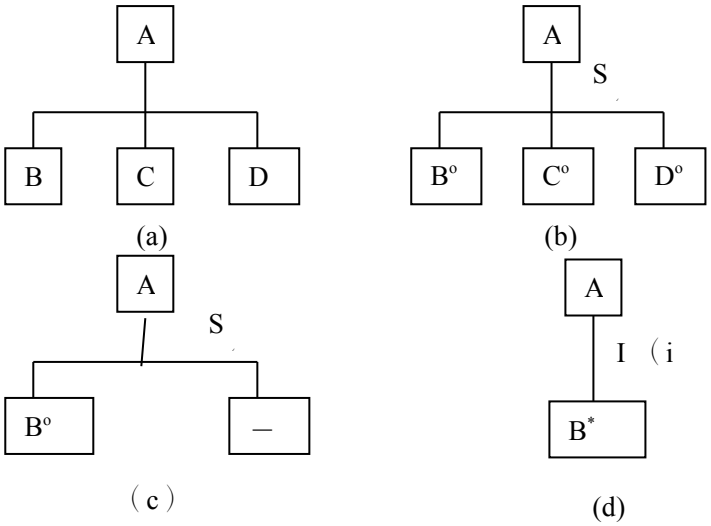


图 6.11 改进的 Jackson 图

(a)顺序结构，B 、C 、D 中任何一个都不能是选择出现或重复出现的数据元素(即，不能是右上角有小圆圈或星　号标记的元素)；

(b)选择结构，S 右面括号中的数字 i 是分支条件的编号；

(c)可选结构，A 或者是元素 B 或者不出现(可选结构是选择结构的一种常见的特殊形式)；

(d)重复结构，循环结束条件的编号为 i。

Jackson 图实质上是对第 3.7 节中介绍的层次方框图的一种精化 。请读者注意，虽然 Jackson 图和描绘软件结构的层次图形式相当类似，但是含义却很不相同，即，层次图中的一个方框通常代表一个模块；而 Jackson 图即使在描绘程序结构时，一个方框也并不代表一个模块，通常一个方框只代表几个语句。层次图表现的是调用关系，通常一个模块除了调用下级模块外，还完成其他操作；而 Jackson 图表现的是组成关系，也就是说, 一个方框中包括的操作仅仅由它下层框中的那些操作组成。

6.4.3 Jackson 方法

Jackson 结构程序设计方法基本上由下述 5 个步骤组成：

(1)分析并确定输入数据和输出数据的逻辑结构，并用 Jackson 图描绘这些数结构。

(2)找出输入数据结构和输出数据结构中有对应关系的数据单元。所谓有对关系是指有直接的因果关系，在程序中可以同时处理的数据单元(对于重复出现数据单元必须重复的次序和次数都相同才可能有对应关系)。

(3)用下述 3 条规则从描绘数据结构的 Jackson 图导出描绘程序结构 Jackson 图：

第一，为每对有对应关系的数据单元，按照它们在数据结构图中的层次在程序构图的相应层次画一个处理框(注意，如果这对数据单元在输入数据结构和输出据结构中所处的层次不同，则和它们对应的处理框在程序结构图中所处的层次与它们之中在数据结构图中层次低的那个对应)；

第二，根据输入数据结构中剩余的每个数据单元所处的层次，在程序结构图的应层次分别为它们画上对应的处理框；

第三，根据输出数据结构中剩余的每个数据单元所处的层次，在程序结构图的
应层次分别为它们画上对应的处理框。

总之，描绘程序结构的 Jackson 图应该综合输入数据结构和输出数据结构的层关系而导出来。在导出程序结构图的过程中，由于改进的 Jackson 图规定在构成序结构的元素中不能有重复出现或选择出现的元素，因此可能需要增加中间层次处理框。

(4)列出所有操作和条件(包括分支条件和循环结束条件)，并且把它们分配程序结构图的适当位置。

(5)用伪码表示程序。

Jackson 方法中使用的伪码和 Jackson 图是完全对应的，下面是和 3 种基本结对应的伪码。

和图 6.1l(a)所示的顺序结构对应的伪码，其中‘seq’和‘end’是关键字：

```
A seq
   B
   C
   D
A end
```

和图 6.1l(b)所示的选择结构对应的伪码,其中‘select’、‘or’和‘end’是关键与 condl 、cond2 和 cond3 分别是执行 B 、C 或 D 的条件：

A select cond1

B

A or cond2

C

A or cond3

D

A end

和图 6.11(d)所示重复结构对应的伪码，其中‘iter’、‘until’,‘while’,和‘end’是关键字(重复结构有 until 和 while 两种形式)，cond 是条件:

A iter until(或 while) cond

B

A end

下面结合一个具体例子进一步说明 Jackson 结构程序设计方法。

[例] 一个正文文件由若干个记录组成，每个记录是一个字符串。要求统计每个记录中空格字符的个数，以及文件中空格字符的总个数。要求的输出数据格式是，每复制一行输入字符串之后，另起一行印出这个字符串中的空格数，最后印出文件中空格的总个数。

对于这个简单例子而言，输入和输出数据的结构很容易确定。图 6.12 是用 Jackson 图描绘的输入输出数据结构。

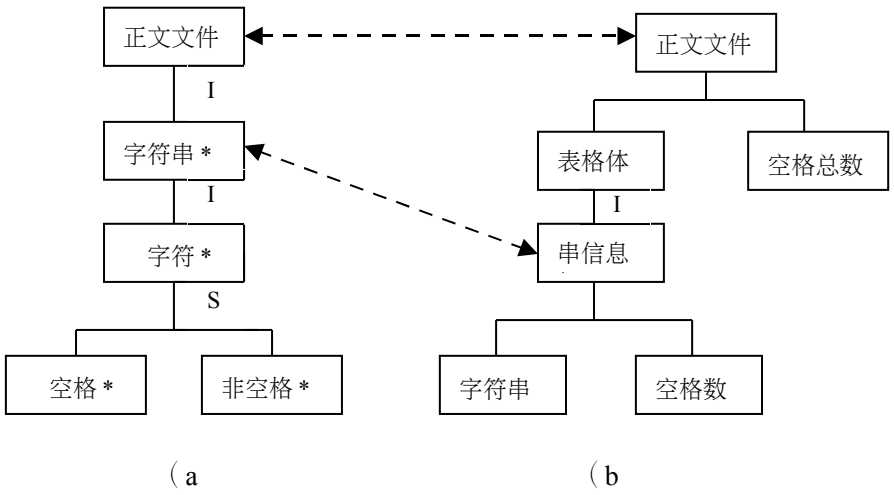


图 6.12　表示输入输出数据结构的 Jackson 图

（a）输入数据结构　　（b）输出数据结构

确定了输入输出数据结构之后，第二步是分析确定在输入数据结构和输出数据结构中有对应关系的数据单元。在这个例子中哪些数据单元有对应关系呢?输出数据总是通过对输入数据的处理而得到的，因此在输入输出数据结构最高层次的两个单元(在这个例子中是“正文文件”和“输出表格”)总是有对应关系的。这一对单元将和程序结构图中最顶层的方框(代表程序)相对应，也就是说经过程序的处理由正文文件得到输出表格。下面还有哪些有对应关系的单元呢?因为每处理输入数据中一个“字符串”之后，就可以得到输出数据中一个“串信息”，它们都是重复出现的数

据单元，而且出现次序和重复次数都完全相同，因此，“字符串”和“串信息”也是一对有对应

关系的单元。

还有其他有对应关系的单元吗?为了回答这个问题依次考察输入数据结构中余下的每个数据单元。“字符”不可能和多个字符组成的“字符串”对应，和输出数据结构中其他数据单元也不能对应。“空格”能和“空格数”对应吗?显然，单个空格并不能决定一个记录中包含的空格个数，因此没有对应关系。通过类似的考察发现，输入数据结构中余下的任何一个单元在输出数据结构中都找不到对应的单元，也就是说，在这个例子中输入输出数据结构中只有上述两对有对应关系的单元。在图 6.12 中用一对虚线箭头把有对应关系的数据单元连接起来，以突出表明这种对应关系。

Jackson 程序设计方法的第三步是从数据结构图导出程序结构图。按照前面已经讲述过的规则，这个步骤的大致过程是：

首先，在描绘程序结构的 Jackson 图的最顶层画一个处理框“统计空格”，它与“正文文件”和“输出表格”这对最顶层的数据单元相对应。但是接下来还不能立即画与另一对数据单元(“字符串”和“串信息”)相对应的处理框，因为在输出数据结构中“串信息”的上层还有“表格体”和“空格总数”两个数据单元，在程序结构图的第二层应该有与这两个单元对应的处理框——“程序体”和“印总数”。因此，在程序结构图的第三层才是与“字符串”和“串信息”相对应的处理框——“处理字符串”。在程序结构图的第四层似乎应该是和“字符串”、“字符”及“空格数”等数据单元对应的处理框“印字符串”、“分析字符”及“印空格数”，这 3 个处理是顺序执行的。但是，“字符”是重复出现的数据单元，因此“分析字符”也应该是重复执行的处理。改进的 Jackson 图规定顺序执行的处理中不允许混有重复执行或选择执行的处理，所以在“分析字符”这个处理框上面又增加了一个处理框“分析字符串”。最后得到的程序结构图为图 6.13。

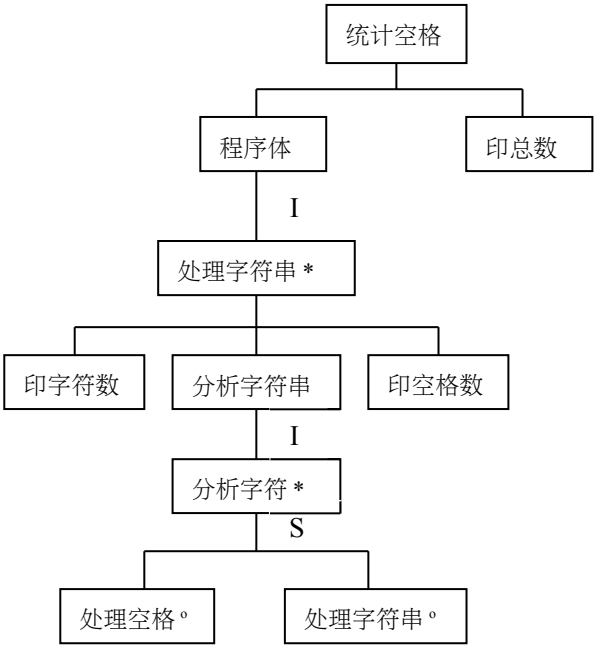


图 6.13 描绘统计空格程序结构的 Jackson 图

Jackson 程序设计方法的第四步是列出所有操作和条件，并且把它们分配到程序结构图的适当位置。首先，列出统计空格个数需要的全部操作和条件如下：

(1) 停止

(2) 打开文件
(3) 关闭文件
(4) 印出字符串
(5) 印出空格数目
(6) 印出空格总数
(7) sum：＝sum+1;
(8) totalsum：＝totalsum+sum;
(9) 读入字符串
(10) sum：＝0
(11)totalsum：＝0
(12) pointer：＝1
(13)pointer：＝pointer+1
I(1)文件结束
I(2)字符串结束
S(3)字符是空格

在上面的操作表中，sum 是保存空格个数的变量，totalsum 是保存空格总数的变量，而 pointer 是用来指示当前分析的字符在字符串中的位置的变量。

经过简单分析不难把这些操作和条件分配到程序结构图的适当位置，结果为图 6.1.4。

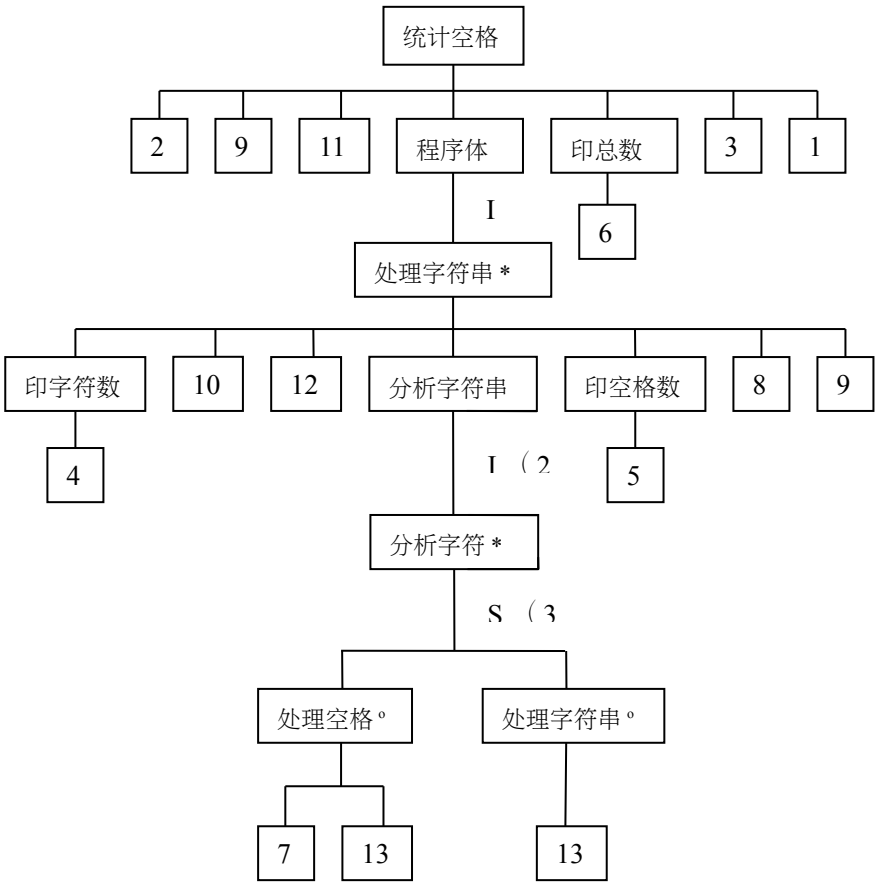


图 6.14 把操作和条件分配到程序结构图的适当位置

Jackson 方法的最后一步是用伪码表示程序处理过程。因为 Jackson 使用的伪码和 Jackson 图之间存在简单的对应关系，所以从图 6.14 很容易得出下面的伪码：

```
统计空格 seq
    打开文件
    读入文件
    读入字符串
    totalsum：=0
    程序体 iter until 文件结束
      处理字符串 seq
      印字符串 seq
        印出字符串
      印字符串 end
      sum：=0
      pointer：=1
      分析字符串 iter until 字符串结束
        分析字符 select 字符是空格
          处理空格 seq
      sum：=sum+1
          pointer：=pointer+1
        处理空格 end
        分析字符 or 字符不是空格
        处理非空格 seq
          pointer：=pointer+1
        处理非空格 end
      分析字符 end
      分析字符串 end
      印空格数 seq
          印出空格数目
      印空格数 end
        totalsum：=totalsum+sum
        读入字符串
      处理字符串 end
          程序体 end
          印总数 seq
          印出空格总数
          印总数 end
          关闭文件
          停止
      统计空格 end
```

以上简单介绍了由英国人 M.Jackson 提出的结构程序设计方法。这个方法在设计比较简单的数据处理系统时特别方便，当设计比较复杂的程序时常常遇到输入数据可能有错、条件不能预先测试、数据结构冲突等问题。为了克服上述困难，把 Jackson 方法应用到更广阔的领域，需要采用一系列比较复杂的辅助技术，详细介绍这些技术已经超出本书的范围。

6.5 程序复杂程度的定量度量

详细设计阶段设计出的模块质量如何呢?第 5 章中曾经讲述了软件设计的基本原理和概念，经过详细设计之后每个模块的内容都非常具体了，因此可以使用这些原理进一步仔细衡量它们的质量。但是，这种衡量毕竟只能是定性的，人们希望能进一步定量度量软件的性质。由于软件工程还是一门很年轻的学科，目前许多定量度量方法还处在研究过程中，本节将要介绍的程序复杂程度定量度量方法是其中比较成熟的一种。

定量度量程序复杂程度的方法很有价值：把程序的复杂程度乘以适当常数即可估算出软件中错误的数量以及软件开发需要用的工作量，定量度量的结果可以用来比较两个不同的设计或两个不同算法的优劣；程序的定量的复杂程度可以作为模块规模的精确限度。

下面着重介绍使用得比较广泛的 McCabe 方法和 Halstead 方法。

6.5.1 McCabe 方法

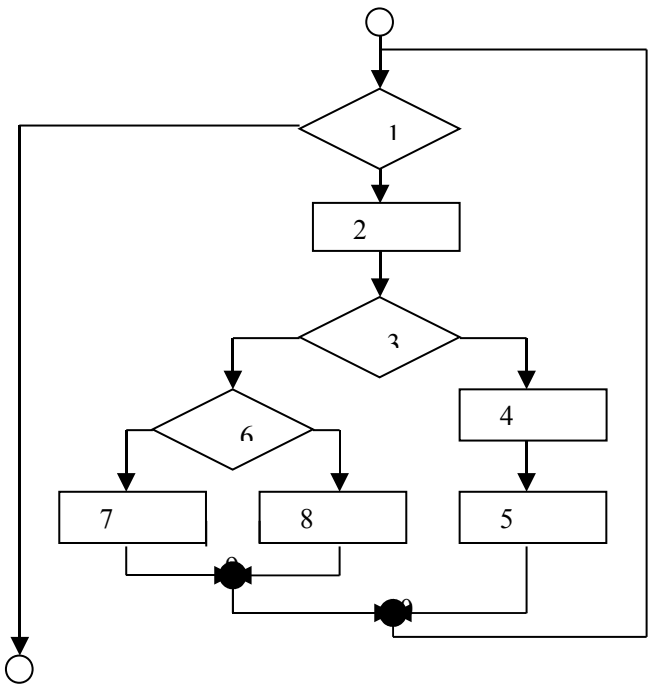
1.流图

McCabe 方法根据程序控制流的复杂程度定量度量程序的复杂程度，这样度量出的结果称为程序的环形复杂度。

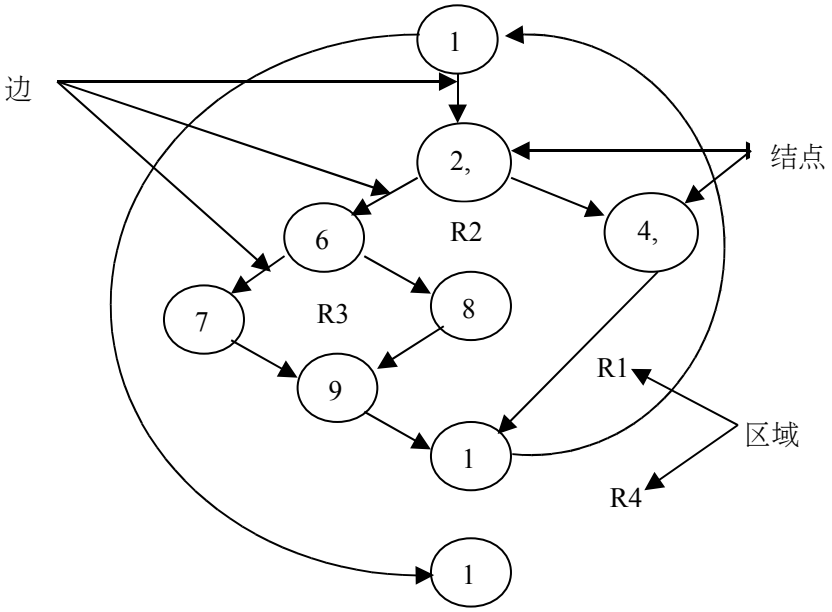
为了突出表示程序的控制流，人们通常使用流图(也称为程序图)。所谓流图实质上是“退化了的”程序流程图，它仅仅描绘程序的控制流程，完全不表现对数据的具体操作以及分支或循环的具体条件。

在流图中用圆表示结点，一个圆代表一条或多条语句。程序流程图中的一个顺序的处理框序列和一个菱形判定框，可以映射成流图中的一个结点。流图中的箭头线称为边，它和程序流程图中的箭头线类似，代表控制流。在流图中一条边必须终止于一个结点，即使这个结点并不代表任何语句(实际上相当于一个空语句)。由边和结点围成的面积称为区域，当计算区域数时应该包括图外部未被围起来的那个区域。

图 6.15 举例说明把程序流程图映射成流图的方法。



(a)



（h

图 6.15 把程序流程图映射成流图

用任何方法表示的过程设计结果，都可以翻译成流图。图 6.16 是用 PDL 表示的处理过程及与之对应的流图。

```
PDL
Procedure:sort
1：  do while records remain
2 :      read record:
              if record field 1=0
3 :       then process record;
                    store in buffer;
                    incremert counter;
4 :      elseif record field 2=0
5 :       then reset counter;
6 :       else process record;
                    store in file;
7a :     endif
             endif
7b : enddo
8 : end
```

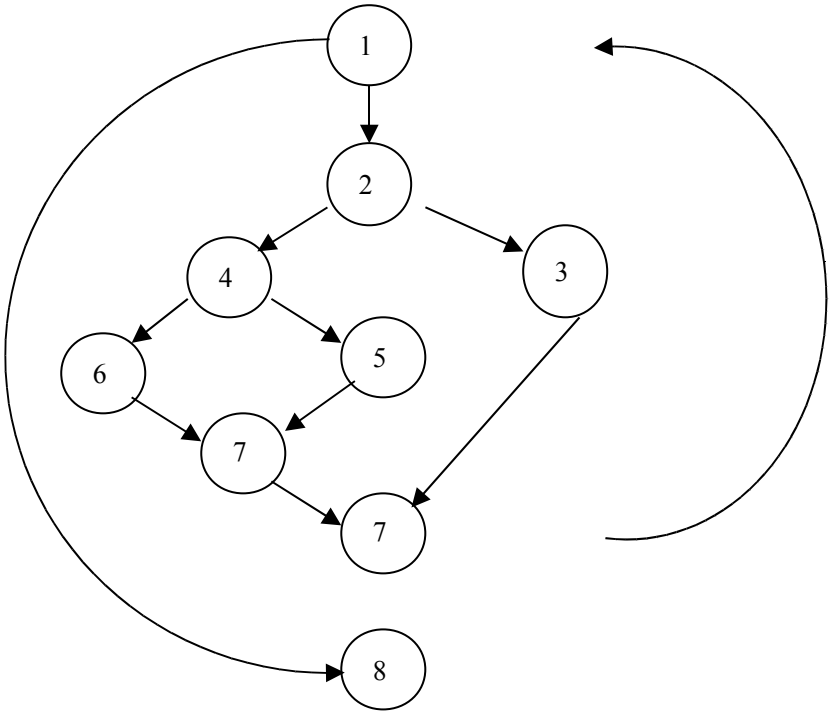


图 6.16 由 PDL 翻译成的流图

当过程设计中包含复合条件时，生成流图的方法稍微复杂一些。所谓复合条件，就是在条件中包含了一个或多个布尔运算符(逻辑 OR，AND，NAND，NOR)。在这种情况下，应该把复合条件分解为若干个简单条件，每个简单条件对应流图中一个结点。包含条件的结点称为判定节点，从每个判定结点引出两条或多条边。图 6.17 是由包含复合条件的 PDL 片断翻译成的流图。

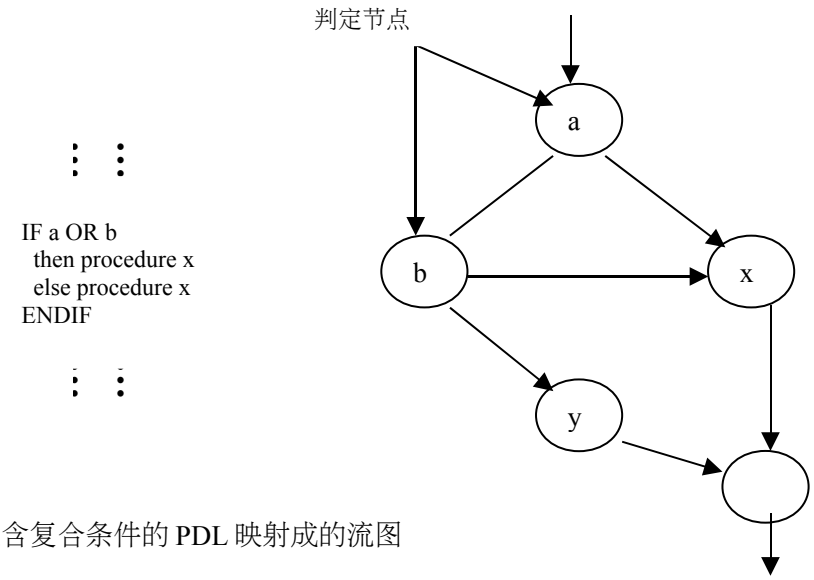


图 6.17 包含复合条件的 PDL 映射成的流图

2.计算环形复杂度的方法

环形复杂度定量度量程序的逻辑复杂度。有了描绘程序控制流的流图之后，可以用下述 3 种方法中的任何一种来计算环形复杂度。

(1)流图中的区域数等于环形复杂度。

(2)流图 G 的环形复杂度 V(G)＝E－N+2，其中，E 是流图中边的条数，N 是结点数。

(3)流图 G 的环形复杂度 V(G)＝P+1，其中，户是流图中判定结点的数目。

例如，使用上述任何一种方法，都可以计算出图 6.16 所示流图的环形复杂度为 4。

3.环形复杂度的用途

程序的环形复杂度取决于程序控制流的复杂程度，也即是取决于程序结构的复杂程度。当程序内分支数或循环个数增加时，环形复杂度也随之增加，因此它是对测试难度的一种定量度量，也能对软件最终的可靠性给出某种预测。

McCabe 研究大量程序后发现，环形复杂度高的程序往往是最困难、最容易出问题的程序。实践表明，模块规模以 V(G)≤10 为宜，也就是说，V(G)＝10 是模块规模的一个更科学更精确的上限。

6.5.2 HaIStead 方法

Halstead 方法是另一个著名的方法，它根据程序中运算符和操作数的总数来度量程序的复杂程度。

令 N1 为程序中运算符出现的总次数，N：为操作数出现的总次数， 程序长度 N 定义为：

$$N = N_1 + N_2$$

详细设计完成之后，可以知道程序中使用的不同运算符(包括关键字)的个数 n1，以及不同操作数(变量和常数)的个数 n2。Halstead 给出预测程序长度的公式如下：

$$H = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

多次验证都表明，预测的长度 H 与实际长度 N 非常接近。

Halstead 还给出了预测程序中包含错误的个数的公式如下：

$$E = N \log_2 (n_1 + n_2)/3000$$

有人曾对从 300 条到 12 000 条语句范围内的程序核实了上述公式，发现预测的错误数与实际错误数相比误差在 8%之内。

6.6 小结

详细设计阶段的关键任务是确定怎样具体地实现用户需要的软件系统，也就是要设计出程序的“蓝图”。除了应该保证软件的可靠性之外，使将来编写出的程序可读性好、容易理解、容易测试、容易修改和维护，是详细设计阶段最重要的目标。结构程序设计技术是实现上述目标的基本保证，是进行详细设计的逻辑基础。

人机界面设计是接口设计的一个重要的组成部分。对于交互式系统来说，人机界面设计和数据设计、体系结构设计及过程设计一样重要。人机界面的质量直接影响用户对软件产品的接受程度，因此，对人机界面设计必须给予足够重视。在设计人机界面的过程中，必须充分重视并认真处理好系统响应时间、用户帮助设施、出错信息处理和命令交互等 4 个设计问题。人机界面设计是一个迭代过程，通常，先创建设计模型，接下来用原型实现这个设计模型并由用户试用和评估原型，然后根据用户意见修改原型，直到用户满意为止。总结人们在设计人机界面过程中积累的经验，得出了一些关于用户界面设计的指南，认真遵守这些指南有助于设计出友好、高效的人机界面。

过程设计应该在数据设计、体系结构设计和接口设计完成之后进行，它的任务是设计解题的详细步骤(即算法)，它是详细设计阶段应完成的主要工作。过程设计的工具可分为图形、表格和语言 3 类，这 3 类工具各有所长，读者应该能够根据需要选用适当的工具。

在许多应用领域中信息都有清楚的层次结构，在开发这类应用系统时可以采用面向数据结构的设计方法完成过程设计。本章以 Jackson 结构程序设计技术为例，对面向数据结构的设计方法做了初步介绍。为了能使用这种方法解决实际问题，还需要进一步钻研有关的专著。

使用环形复杂度可以定量度量程序的复杂程度，实践表明，环形复杂度 V(G)＝10 是模块规模的合理上限。

习 题 6

1.假设只有 SEQUENCE 和 DO_WHILE 两种控制结构，怎样利用它们完成 IF_THEN_ELSE 操作?
2.假设只允许使用 SEQUENCE 和 IF_THEN_ELSE 两种控制结构，怎样利用它们完成 DO_WHILE 操作?

3.画出下列伪码程序的程序流程图和盒图:

```
START
IF p  THEN
          WHILE q DO
                f
          END DO
       ELSE
          BLOCK
            g
            n
          END BLOCK
END IF
STOP
```

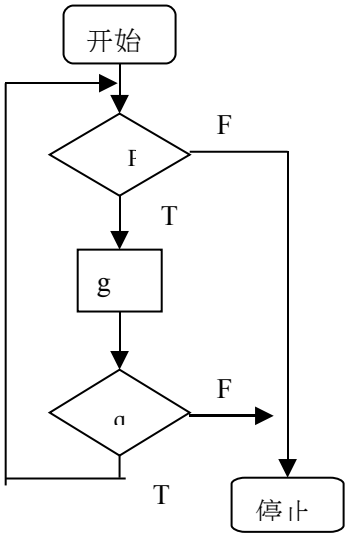


图 6.18 一个非结构化程序

4.图 6.18 给出的程序流程图代表一个非结构化的程序，请问·

(1)为什么说它是非结构化的?

(2)设计一个等价的结构化程序。

(3)在(2)题的设计中你使用附加的标志变量 flag 了吗?若没用，请再设计一个使用 flag 的程序；若用了，再设计一个不用 flag 的程序。

5.研究下面的伪码程序:

```
LOOP: Set I to(START+FINISH)／2
    If TABLE(I)=ITEM goto FOUND
    If TABLE(I)<ITEM Set START to(I+1)
    If TABLE(I)>ITEM Set FINISH to(I-1)
    If(FINISH-START)>1 goto LOOP
    If TABLE(START)=ITEM goto FOUND
    If TABLE(F1NISH)=ITEM goto FOUND
    Set FLAG to 0
    Goto DONE
FOUND: Set FLAG to 1
DONE: Exit
```

要求:

(1)画出程序流程图。

(2)程序是结构化的吗?说明理由。

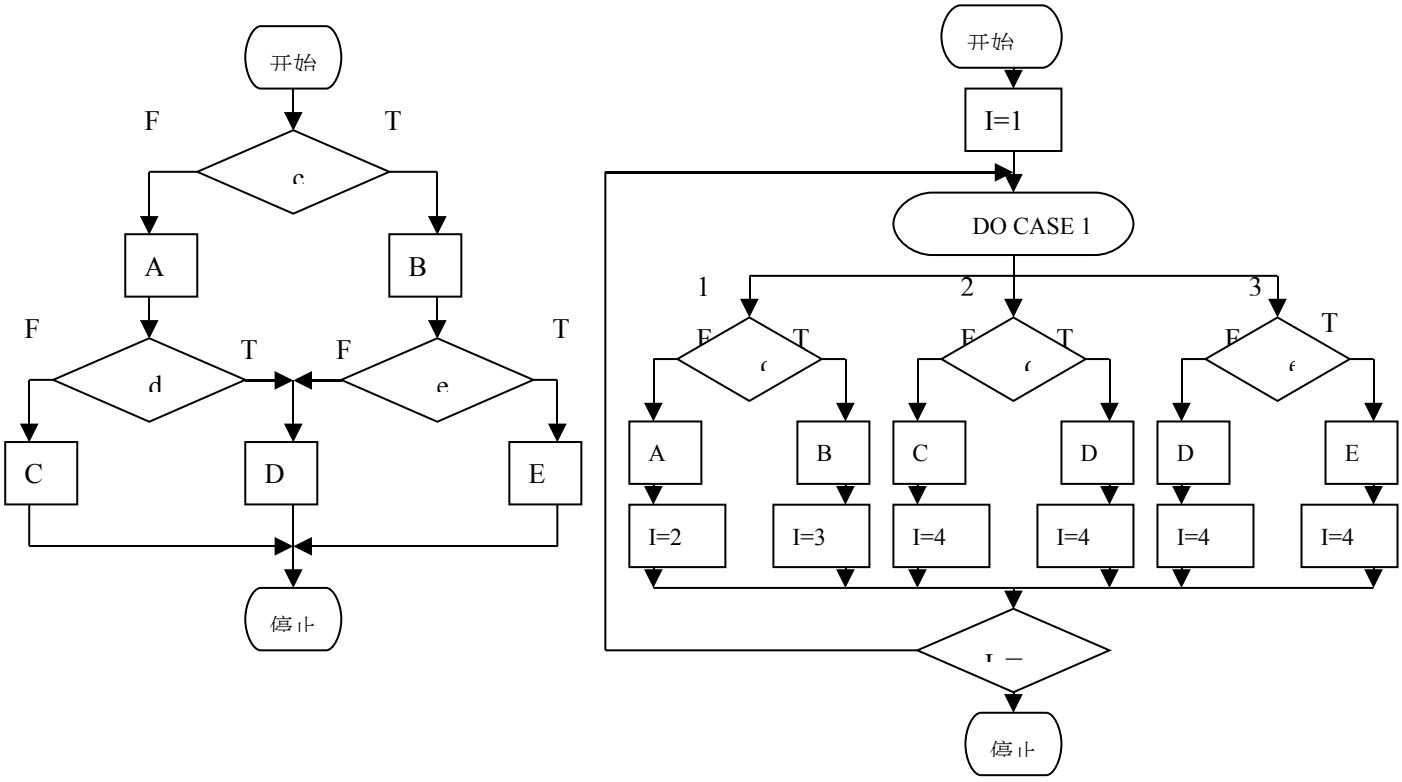
(3)若程序是非结构化的，请设计一个等价的结构化程序并且画出程序流程图。

(4)此程序的功能是什么?它完成预定功能有什么隐含的前提条件吗?

6.用 Ashcroft_Manna 技术可以将非结构化的程序转换为结构化程序，图 6.19 是一个转换的例子。

(1)你能否从这个例子总结出 Ashcroft_Manna 技术的一些基本方法?

(2)进一步简化图 6.19(b)给出的结构化设计。



7.某交易所规定给经纪人的手续费计算方法如下：总手续费等于基本手续费加上与交易中的每股价格和股数有关的附加手续费。如果交易总金额少于 1000 元，则基本手续费为交易金额的 8.4%；如果交易总金额在 1000 元到 10000 元之间，则基本手续费为交易金额的 5%，再加 34 元；如果交易总金额超过 10000 元，则基本手续费为交易金额的 4%加上 134 元。当每股售价低于 14 元时，附加手续费为基本手续费的 5%，除非买进、卖出的股数不是 100 的倍数，在这种情况下附加手续费为基本手续费的 9%。当每股售价在 14 元到 25 元之间时，附加手续费为基本手续费的 2%，除非交易的股数不是 100 的倍数，在这种情况下附加手续费为基本手续费的 6%。当每股售价超过 25 元时，如果交易的股数零散(即，不是 100 的倍数)，则附加手续费为基本手续费的 4%，否则附加手续费为基本手续费的 1%。

要求:

(1)用判定表表示手续费的计算方法;

(2)用判定树表示手续费的计算方法。

8.画出下列伪码程序的流图，计算它的环形复杂度。你觉得这个程序的逻辑有什么问题吗?

```
C     EXAMPLE
LOOP: DO WHILE Z>o
```

```
A=B+1
IF A>10
   THEN X=A
   ELSE Y=Z
END IF
IF Y<5
   THEN PRINT X,Y
   ELSE IF Y=2
        THEN GOTO LOOP
        ELSE C=3
        END IF
 END IF
 G=H+R
 END DO
 IF F>0
    THEN PRINT G
    ELSE PRINT K
 END IF
 STOP
```

9.把统计空格程序的Jackson图(图6.13)改画为等价的程序流程图和盒图。

10.人机对话由操作员信息和系统信息交替组成。假设一段对话总是由操作员信息开始以系统信息结束，请用Jackson图描绘这样的人机对话过程。