第6章 软件详细设计

本章内容：

6.1 详细设计的任务与方法

6.2 设计表示法

6.3 结构化程序设计

6.4 结构化详细设计应用实例

第6章 软件详细设计

在软件的总体设计中，完成了数据和系统结构。已将系统划分为多个模块，并将它们按照一定的原则组装起来，也确定了每个模块的功能及模块与模块之间的外部接口。在理想的情况下，详细设计是软件设计的第二阶段。在这个阶段，由于开发系统内外的人员理论上都使用一种自然语言，因此设计说明最好是用自然语言。很清楚，这个阶段必须定义过程的细节。

如果用自然语言来定义，很容易产生二义性，因此要求对于细节的表达式要十分小心严谨。

6.1 详细设计的任务与方法

在软件的分析与设计中，如果系统很小，一般做法是将总体设计与详细设计合并在一起进行。当系统有一定规模时，这两者是分开进行的。详细设计要完成所有设计的细节。

6.1.1 详细设计的基本任务

1. 数据结构设计

前面的需求分析、总体设计阶段，确定的概念性的数据类型，要进行确切的定义。这一部分的设计内容一般比较多，所以大多数采用小型数据库辅助的方法。

2. 物理设计

对数据库进行物理设计，即确定数据库的物理结构。物理结构主要指数据库的存储记录格式、存储记录安排和存储方法，这些都依赖于具体所使用的数据库系统。

3. 算法设计

在总体设计的结构完成后，结构各个环节的实现是多解的。这就需要用系统设计与分析的技术来描述。可以用某种图形、表格、语言等工具将每个模块处理过程的详细算法描述出来。

4. 界面设计

用户界面的设计现在显得比较重要，可以采用字符用户界面设计、图形用户界面和多媒体人机界面设计。这就要结合具体的系统来处理。

5. 其他设计

根据软件系统的类型，还可能要进行以下设计。

（1）代码设计：为了提高数据的输入、分类、存储及检索等操作的效率，以及节约内存空间，对数据库中的某些数据项的值要进行代码设计。

（2）输入/输出格式设计。

（3）人机对话设计：对于一个实时系统，用户与计算机频繁对话，因此要进行对话方式、内容及格式的具体设计。

（4）网络设计：如果设计的软件是一个分布式系统，那么还要进行网络拓扑结构设计。

6. 编写设计说明书

详细设计说明书有下列的主要内容。

（1）引言：包括编写目的、背景、定义、参考资料。

（2）程序系统的组织结构。

（3）程序1（标识符）设计说明：包括功能、性能、输入、输出、算法、流程逻辑、接口。

（4）程序2（标识符）设计说明。

（5）程序N（标识符）设计说明。

7. 评审

对处理过程的算法和数据库的物理结构都要评审。

6.1.2 详细设计方法

处理详细设计中采用的典型方法是结构化程序设计（SP）方法，最早是由E.W.Dijkstra在20世纪60年代中期提出的。详细设计并不是具体地编写程序，而是将总体设计结构细化成很容易从中产生程序的图纸。这就说明详细设计的结果基本决定了最终程序的质量。

要提高软件的质量，减少软件的故障，延长软件的生存期。要保障软件的可测试性、可维护性。而软件的可测试性、可维护性又与程序的易读性有很大关系。因此，详细设计的目标不仅要在逻辑上正确地实现每个模块的功能，而且还应使设计出的处理过程清晰易读。要实现上述要求，采用的关键技术之一就是结构化程序设计。结构化程序设计方法有以下几个基本要点。

1. 采用自顶向下、逐步求精的程序设计方法

在需求分析、总体设计中，都采用了自顶向下、逐层细化的方法。分析中使用“抽象”这个手段，上层对问题抽象、对模块抽象和对数据抽象。而下层则进一步分解，进入另一个抽象层次。在详细设计中，虽然处于“具体”设计阶段，但在设计某个复杂的模块内部处理过程时，仍可以采用逐步求精的方法。可以将其分解为若干个模块来实现，降低处理细节的复杂度。

2. 使用三种基本控制结构构造程序

任何程序都可由顺序、选择及重复三种基本控制结构构造。这三种基本结构的共同点是单入口、单出口。它不但能有效地限制使用GOTO语句，还创立了一种新的程序设计思想、方法和风格，同时为自顶向下、逐步求精的设计方法提供了具体的实施手段。设计时，如果对一个模块处理过程细化中，开始是模糊的，可以用下面三种方式对模糊过程进行分解

（1）用顺序方式对过程分解，确定各部分的执行顺序。
（2）用选择方式对过程分解，确定某个部分的执行条件。
（3）用循环方式对过程分解，确定某个部分进行重复的开始和结束的条件。
对处理过程仍然模糊的部分反复使用以上分解方法，最终可将所有细节确定下来。

3. 组织形式

在详细设计阶段，当参加设计的人员比较多的时候，有可能因为设计员的技术水平、设计风格不同而影响到系统的质量。因此，要组织以一个负责全部技术活动的三人为核心小组。小组中有负责全部技术的主程序员，协调、支持主程序员的后备程序和负责事务性工作的程序管理员，再加上其他技术人员。这样做的目的是使设计责任集中在少数人身上，利于提高软件质量，并且能有效地提高软件生产率。这种组织形式在IBM公司和其他软件公司纷纷采用，效果很好。

6.2 设计表示法

详细描述处理过程常用三种工具：图形、表格和语言。本节主要介绍可作为详细设计中描述的结构化语言、判定表、判定树。

描述加工逻辑的结构化语言、判定表及判定树。结构化语言、判定表或判定树则详细描述数据流图中不能被再分解的每一个基本加工的处理逻辑。加工逻辑也有对加工点的说明。

6.2.1 结构化语言

结构化语言是介于自然语言和形式化语言之间的一种类自然语言。结构化语言语法结构包括内外两层。内部语法则比较灵活，可以使用数据字典中定义过的词汇、易于理解的一些名词、运算符和关系符；外层语法具有较固定的格式，设定一组符号如IF、THEN、ELSE、DO WHILE…ENDWHILE、DO CASE…ENDCASE等，用于描述顺序、选择和重复的控制结构。用结构化语言描述的处理功能结构清晰、简明易懂。下面是一个行李托运收费功能描述的例子。

例6-1：用户要求的自然语言（中文）含义为：如果行李不超过30公斤，那么可以免费托运；如果行李超过30公斤，那么，对头等舱乘客超过部分每公斤收费4元，对普通舱乘客超重部分每公斤收费6元；如果乘客是残疾人，那么，收费减半。

上述需求用结构化语言表示如下：

```
IF 行李重量W≤30公斤
   免交托运费
ELSE
   IF 是头等舱乘客
      IF是残疾乘客
        托运费=（W-30）*2
      ELSE            /*否则是正常乘客*/
        托运费=（W-30）*4
      ENDIF
   ELSE               /*是普通舱乘客*/
      IF是残疾乘客
        托运费=（W-30）*3
      ELSE            /*否则是正常乘客*/
        托运费=（W-30）*6
      ENDIF
  ENDIF
ENDIF
```

6.2.2 判定表

判定表也是在设计中常用的技术。在有些情况下，数据流图中的某个加工的一组动作依赖于多个逻辑条件的取值。

这时，用自然语言或结构化语言都不易清楚地描述出来，而用判定表就能够清楚地表示复杂的条件组合与应做的动作之间的对应关系。

判定表（Decision Table）是判定树的表格形式，包括四部分：条件定义、条件组合、动作定义和条件组合下的动作。判定表的结构如图6-1所示。

图6-1 判定表的结构

表6-1是结构化语言对应的判定表。

条件组合		1	2	3	4	5	6	7	8
条件	W>30公斤	√	√	√	√				
	头等舱乘客	√	√			√	√		
	残疾乘客	√		√		√		√	
行动	（W-30）*2	√							
	（W-30）*4		√						
	（W-30）*3			√					
	（W-30）*6				√				
	免费					√	√	√	√

表6-1 行李托运费处理判定表

判定表比判定树更严格、更具有逻辑性。判定表的条件严格按二进值取值，不会遗漏任何一种组合。条件组合个数等于2n。但是，原始判定表比较机械，可能会含有一些多余的重复的信息，如表6-1所示中条件组合5，6，7，8的行动都是一样。所以，应对判定表做适当的逻辑优化。表6-2是表6-1优化后的判定表。表中“--”表示该条件满足或不满足均可。

判定表能够把在什么条件下系统应做什么动作准确无误地表示出来，但不能描述循环的处理特性，循环处理还需结构化语言。

条件组合		1	2	3	4	5/6/7/8
条件	W>30公斤	√	√	√	√	
	头等舱乘客	√	√			--
	残疾乘客	√		√		--
行动	（W-30）*2	√				
	（W-30）*4		√			
	（W-30）*3			√		
	（W-30）*6				√	
	免费					√

表6-2 行李托运费处理优化判定

6.2.3 判定树

判定树是判定表的变形，一般情况下它比判定表更直观，且易于理解和使用。如图6-2所示是与表6-1功能等价的判定树。当处理逻辑中含太多判定条件及其组合时，用判定表和判定树描述会比较方便、直观。


行李费算法
行李重量（W>30）
行李重量（W<=30）——免费
头等舱乘客
普通舱乘客
残疾乘客——（W-30）*2
正常乘客——（W-30）*4
残疾乘客——（W-30）*3
正常乘客——（W-30）*6

图6-2 判定树

以上三种逻辑表达工具各有所长和不足，归纳起来可以得出下列结论。

对于顺序执行和循环执行的动作，用结构化语言描述；对于存在多个条件复杂组合的判断问题，用判定表和判定树。判定树较判定表直观易读，判定表进行逻辑验证较严格，能把所有的可能性全部都考虑到。可将两种工具结合起来，先用判定表作底稿，在此基础上产生判定树。

6.3 结构化程序设计

结构程序设计是一种程序设计技术，它采用自顶向下、逐步求精的设计方法和单入口单出口的控制结构。结构程序设计技术的优越性如下。

（1）自顶向下、逐步求精的方法符合人类解决复杂问题的普遍规律，因此可以显著提高软件开发工程的成功率和生产率。

（2）用先全局后局部、先整体后细节、先抽象后具体的逐步求精过程开发出的程序有清晰的层次结构，因此容易阅读和理解。

（3）不使用GO TO语句，仅使用单入口单出口的控制结构，使得程序的静态结构和它的动态执行情况比较一致，易于阅读和理解。

（4）控制结构有确定的逻辑模式，编写程序代码只限于很少几种直截了当的方式，因此源程序清晰流畅。

（5）程序清晰和模块化使得在修改和重新设计一个软件时可以重用的代码量最大。

（6）程序的逻辑结构清晰，有利于程序正确性证明。

6.3.1流程图程序

在软件工程中，一个程序可以用流程图的形式表示出来,用流程图描述客观存在的事物特性。流程图是一个描述程序结构的控制的流程和指令执行情况的有向图。他是程序的一个比较直观的表达形式。一个程序流程图的基本元素是函数结点、谓词结点和汇点。

1）函数结点

如果一个结点有一个入口线和一个出口线，则称为函数结点。函数结点通常表示为如图6--3所示的形式。其中，f是函数结点的名字。由于函数结点一般和赋值语句相对应，

图6--3 流程图的函数结点

所以f也表示了这一个结点对应的函数关系。

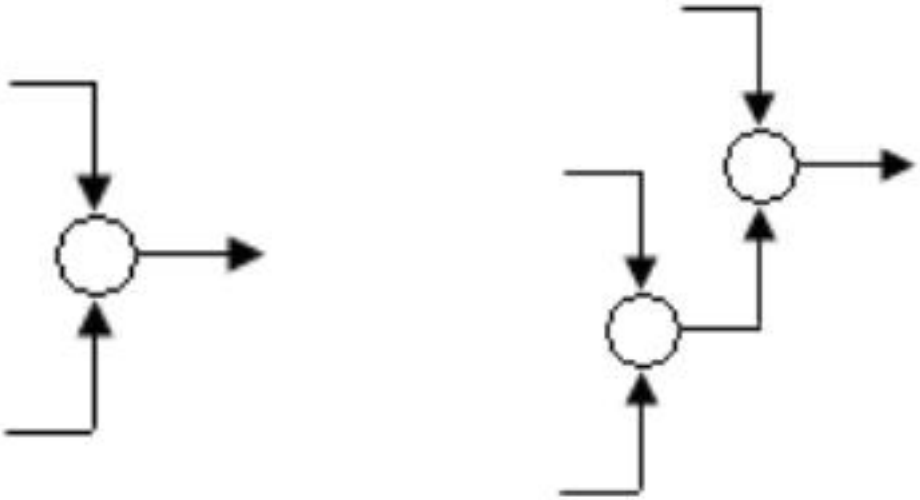
2）谓词结点

如果一个结点有一个入口线和两个出口线，且它不改变程序的数据项的值，则称为谓词结点。谓词结点通常表示为如图6--4所示的形式。其中，p是一个谓词，根据p的逻辑值是真还是假（T或F），结点有不同的出口。或者说有不同的控制流程。这里我们约定:本书讨论中,若没有特别说明,总是约定流程图上方指示线为真,下方为假。

（a） 图6--4谓词结点 （b）

3）汇点

如果一个结点有两个和一个出口线，而且它不执行任何运算，那么称为汇点,通常表示为如图6--5(a)所示。由多个入口线汇集到一点的情形可以用多个汇点的联结表示，如图6--5(b) 所示。



（a）　　（b）

图6--5 流程图中汇点表示

6.3.2 三种基本控制结构

程序流程图是历史最悠久、使用最广泛的一种描述程序逻辑结构的工具，如图6-6所示为用流程图表示的三种程序基本控制结构。

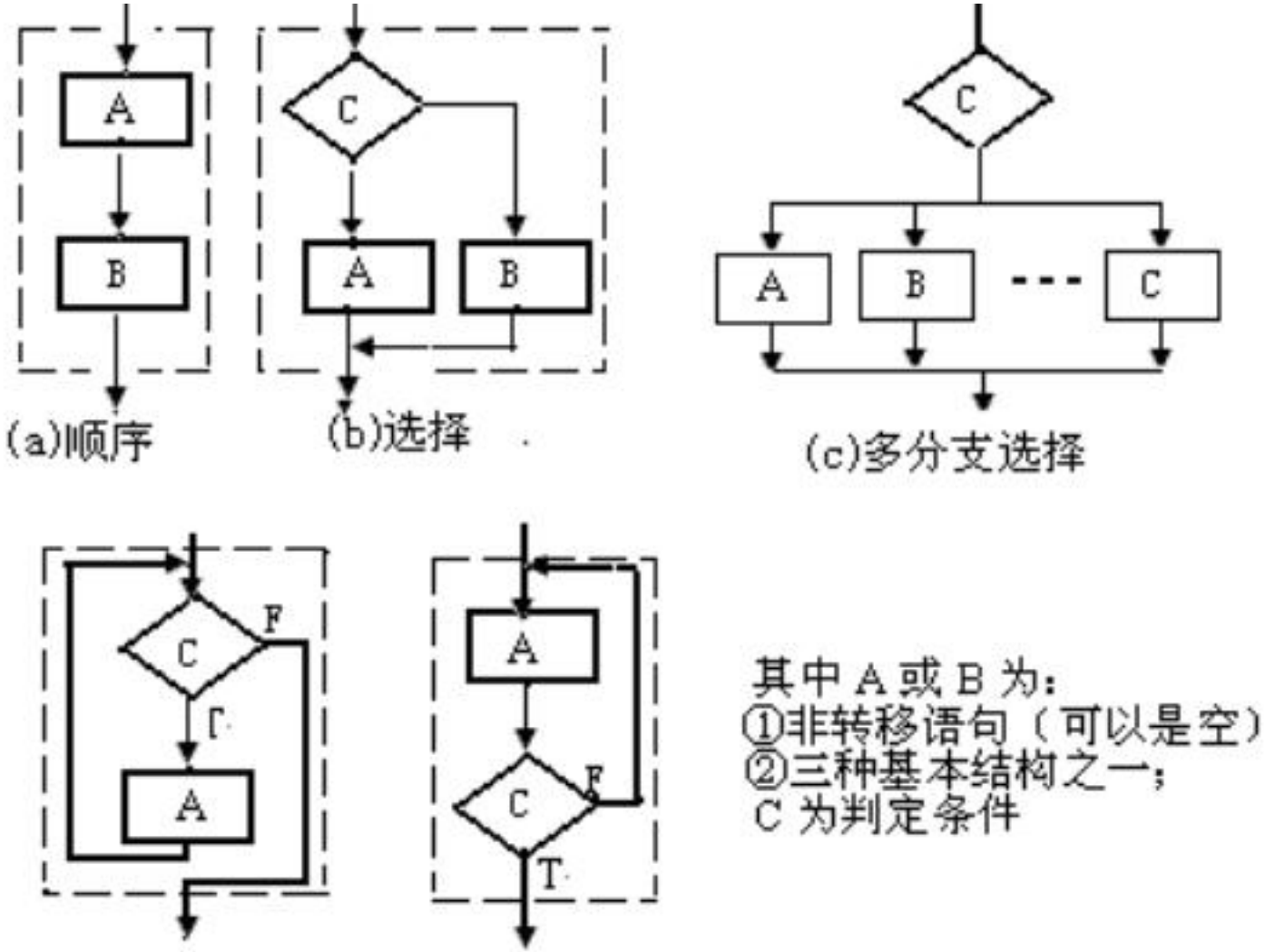


图6-6 程序的三种基本控制结构

（1）顺序型：如图6-6（a）所示，由几个连续的执行步骤依次排列构成。
（2）选择型：常见的选择结构有简单选择与多分支结构。由某个逻辑判断式的取值决定选择两个加工中的一个，这种结构是简单选择，如图6-6（b）所示。多情况（case）多分支选择（如图6-6（c）所示）列举多种执行情况，根据控制变量取值，现在执行其一。此外，还有一种是多分支嵌套的结构。

（3）循环结构：循环结构一般有先判断（while）当型循环和后判断（until）直到型循环。先判断（while）当型循环如图6-6（d）所示，在循环控制条件成立时，重复执行特定的循环；后判断（until）直到型循环如图6-6（e）所示，重复执行特定的循环，直到控制条件成立时。

任何复杂的程序流程图都应由这三种基本控制结构组合嵌套而成。为了说明上述三种基本控制结构的相互组合和嵌套的可用性，给出了如图6-7所示的程序流程图。

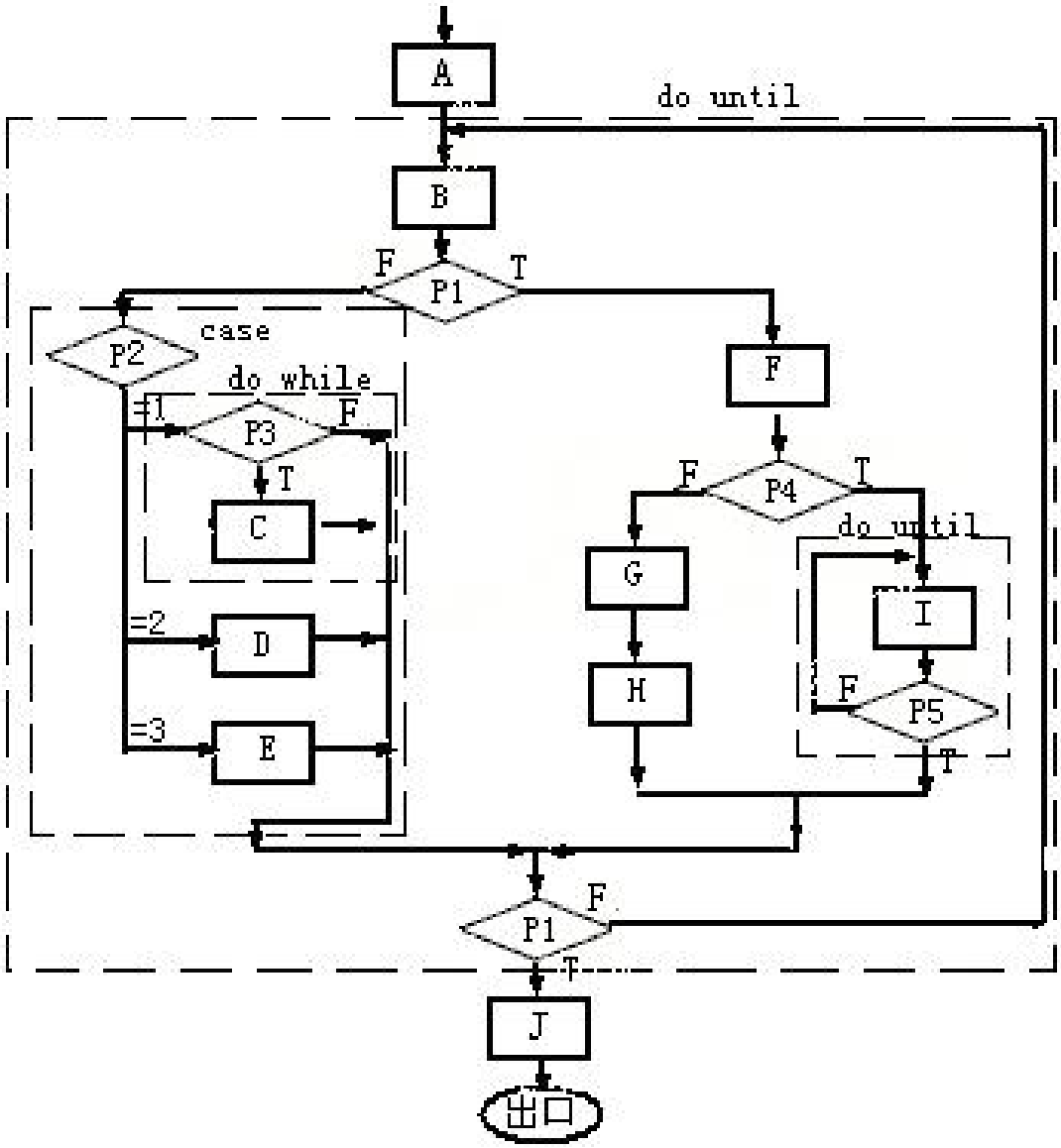


图6-7 嵌套构成的流程图实例

在图6-7中加了一些虚线的框，是为了方便读者理解控制结构的嵌套关系。从图上可以看出，这个图是结构化的流程图。

6.3.3 常用符号

在一个软件的开发初期，一般都要写一个系统标准的说明。其中需要对流程图所用的符号做出确切的规定。在流程图中，除了使用规定的符号外，其他符号是不允许出现在流程图中的。

下面给出国际标准化组织提出的，并由中国国家技术监督局批准的在流程图中的部分标准符号，如图6-8所示。

图6-8 标准程序流程图符号

对于上图，有如下几点需要说明。

（1）循环的上下界设有一对特殊的符号。循环开始符号是两个上角为圆角的矩型，循环结束符号是两个下角为圆角的矩型。其中要注明循环名和进入循环条件或循环终止条件。一般情况下，这对符号要在同一条纵线上，上下对应，循环体在中间，如图6-9所示。

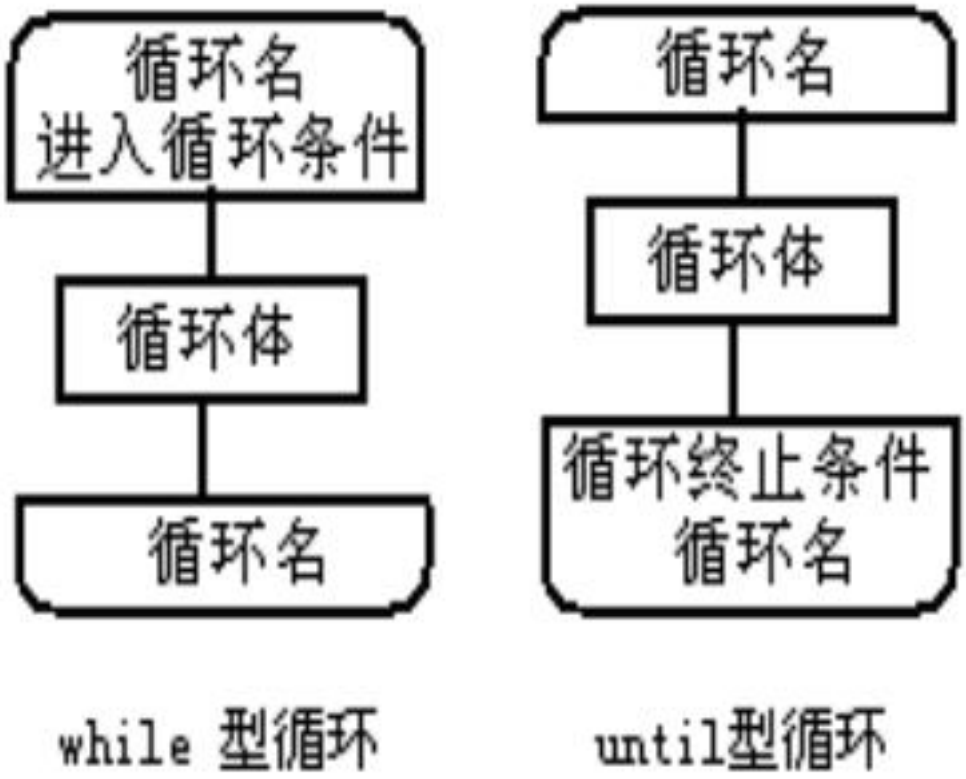


图6-9 循环的标准符号

（2）流线表示控制的流向。自上而下、自左向右的自然形式，流线可以不加箭头。否则必须在流线上加箭头。

（3）注释符号可以用来标识注释的内容，虚线连接在相关符号上。

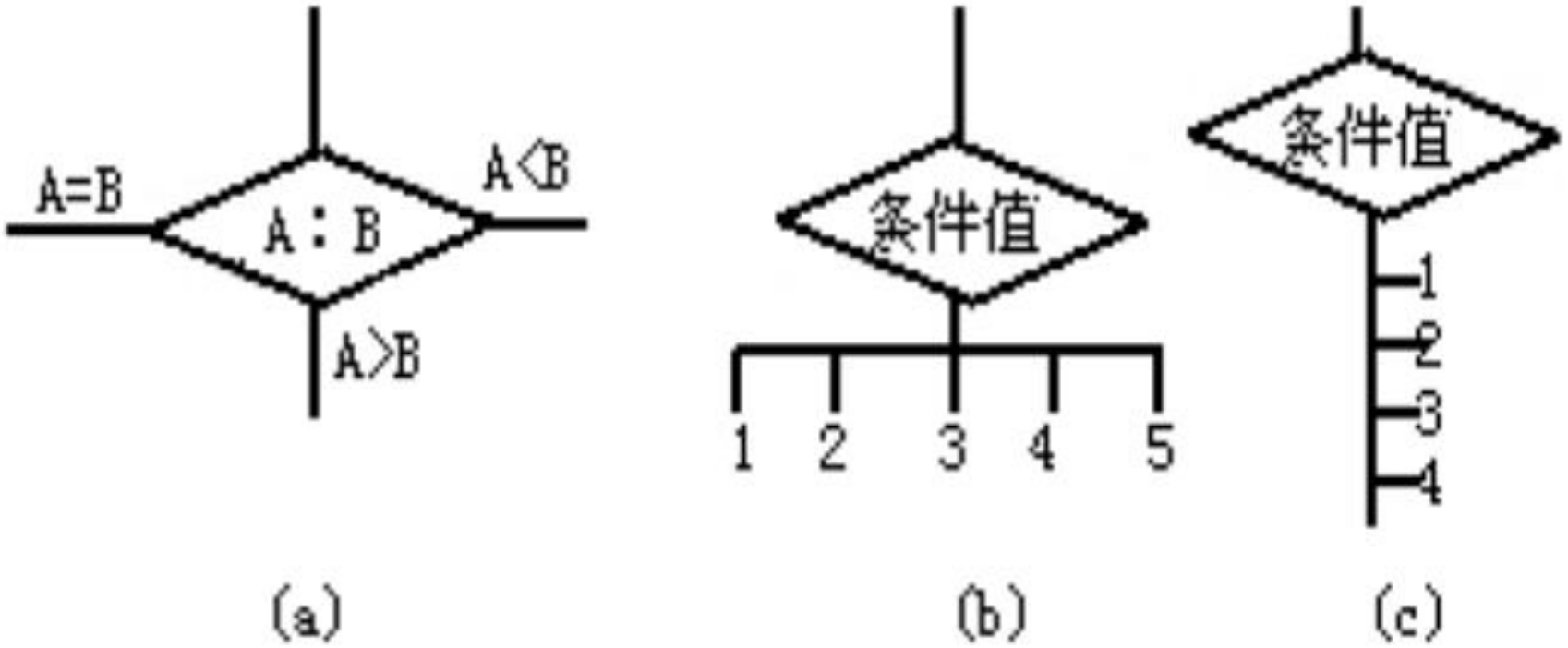


图6-10 多出口判断

（4）判断有一个入口，但是可以有多个可选的出口。在判断条件取值后，有一个且仅有一个出口被激活。多出口的判断即为CASE 结构。图6-10给出多出口判断的表示。图6-10（a）、图6-10（b）和图6-10（c）分别表示具有3、5和4个出口判断。

（5）虚线表示两个或多个条件间的选择关系。

（6）外接符号与内接符号表示流线在另外一个地方连接，或者表示转向外部或从外部转入。

流程图在描述程序控制结构时的优点是直观清晰、易于使用。开发人员都采用这种工具来抽象描述程序的控制过程。但是，流程图有如下严重缺点：

（1）存在的第一个问题是由于用流程图本身没有限制，所以可以随心所欲地画控制流程线的流向。因此也容易造成非结构化的程序结构，编码时势必就要使用GOTO语句实现，导致基本控制块多入口多出口的结果。这样会使软件质量受到影响，与软件设计的原则相违背。

（2）由于使用如图6-6所示流程图的基本结构构造流程图，当遇到多层嵌套的循环，而且每层仅容许一个出口时，那么退出效率就会很差。

（3）高层的宏观控制流程图与低层的微观控制流程的区分问题。

（4）不易表示数据结构。

为了克服流程图的缺陷，要求流程图都应由三种基本控制结构顺序组合和完整嵌套而成，不能有相互交叉的情况，这样的流程图是结构化的流程图。

非结构化转换为结构化(选讲)

上节给出了将任意正规程序转换成结构化程序一种转换算法，其目的是为了证明结构化定理。对一个具体程序来说，这种方法并不是唯一的方法，所得到的结构化程序也不一定是最好的。下面以几个例子说明从非结构化程序转化成结构化程序的一般方法。

一、用结构化定理证明过程提供的方法

例4 图6--23的流程图程序是一个非结构化程序，利用上节结构化定理证明过程给出的方法将其转换成结构化程序的步骤如下。

第一步，结点编号如图6--23；

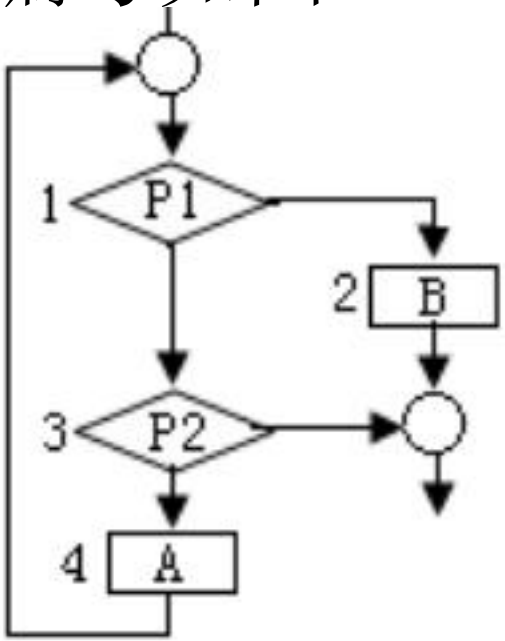


图6--23结点编号非结构化流程图

第二步，图中四个结点构造新程序分别如图6--24；

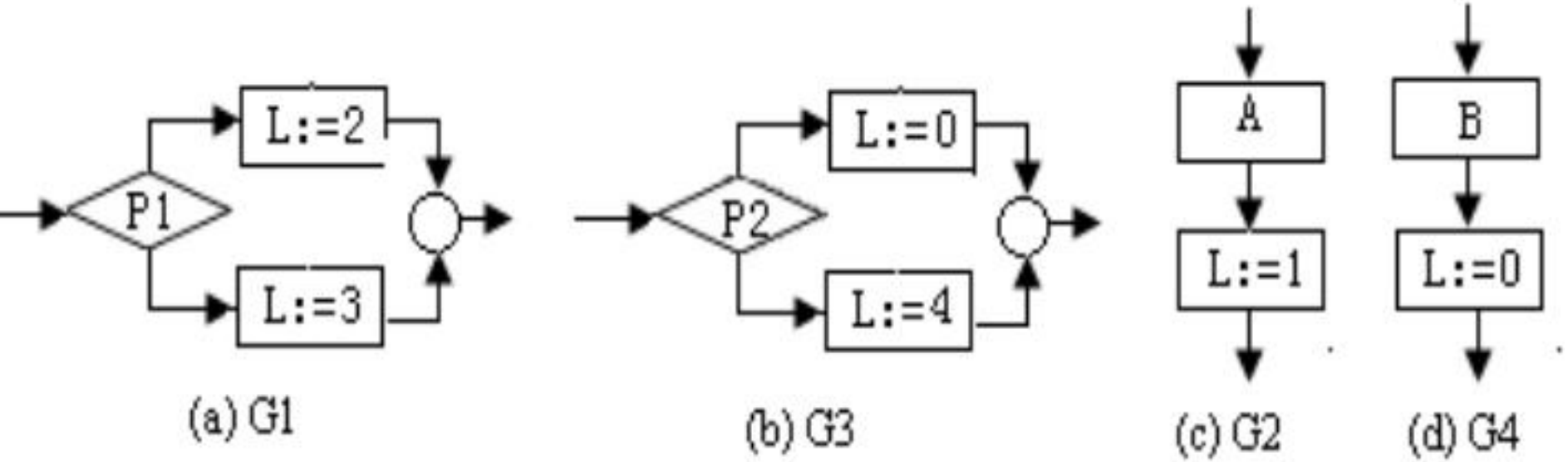


图6—24点构造新流程

第三步，得到如图6--25的等价结构化程序。

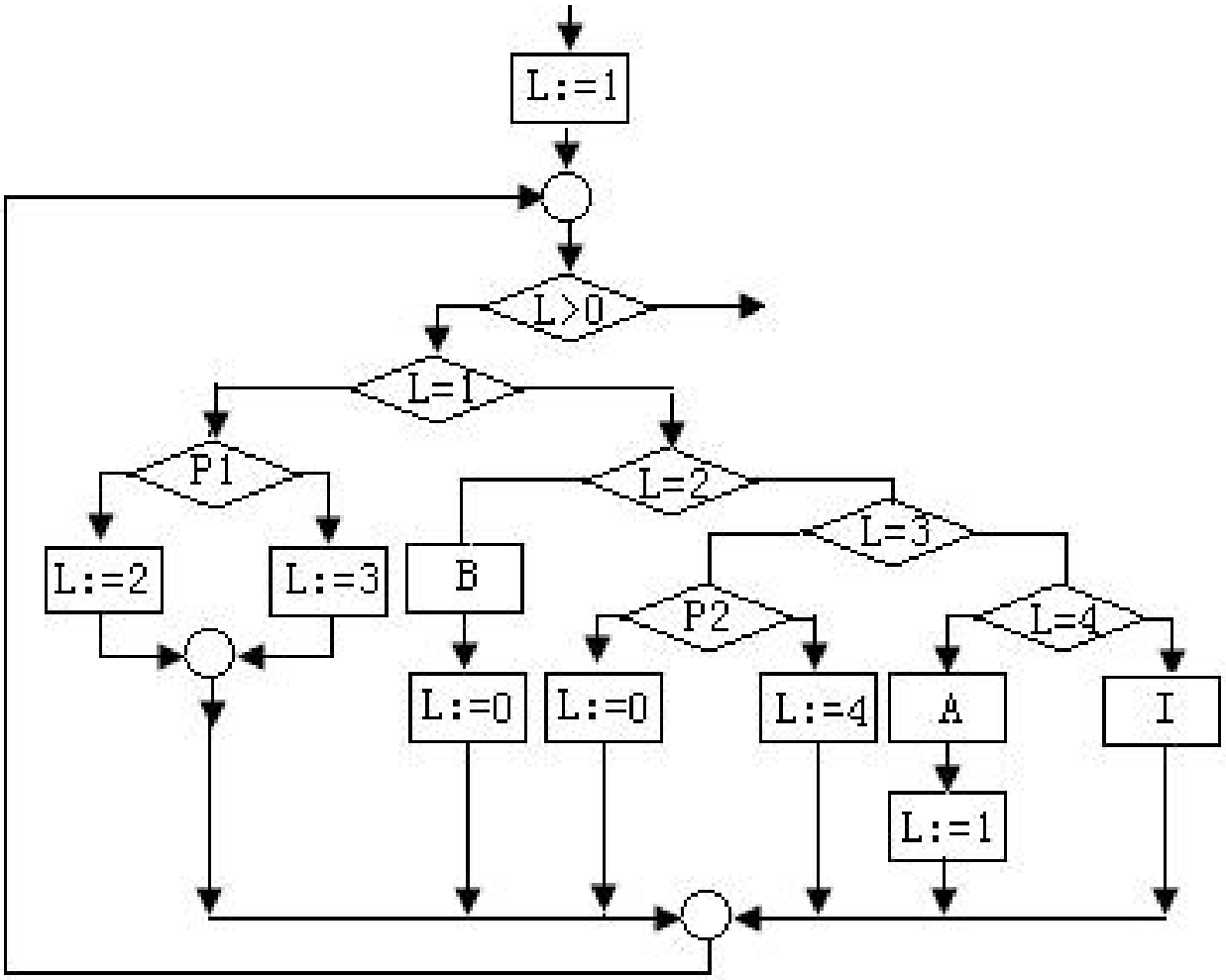
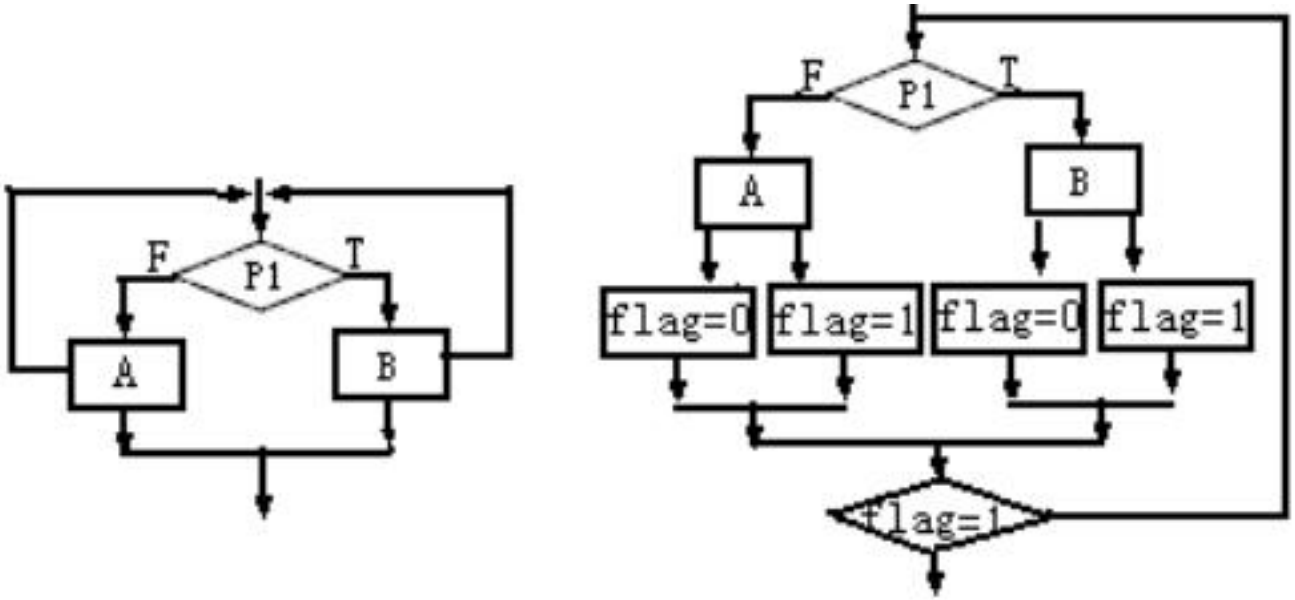


图6—25与图6--23等价结构化流程图

二、其它方法

例6 图2—26 (a)是一个非结构化流程图，图2—26 (b)中利用一个标志逻辑变量FLAG将之改进成等价的结构化程序。



(a)非结构化流程图　　(b)与(a)等价的结构化流程图

图6—26增加辅助变量的转换

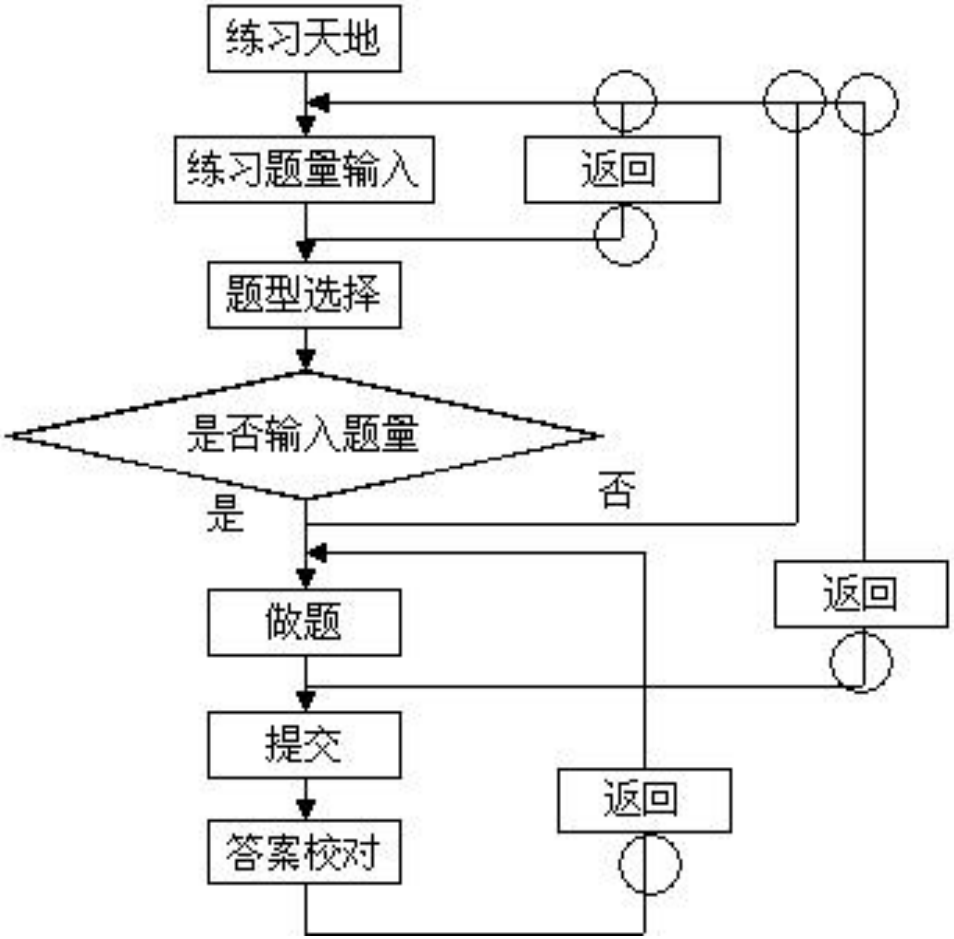
由图可以看出，将非结构化流程图转换为等价的结构化流程图时，可以使用标志技术。标志是一个变量，他记忆过去判定量是“真”还是“假”，并保持信息以后使用。

6.3.4 常见错误

在有些教科书上还给出了将任意正规程序转换成结构化程序的转换算法，其目的是为了使设计出的流程符合结构化理论。对一个具体程序来说，这种方法并不是唯一的方法，所得到的结构化程序也不一定是最好的。

下面用几个例子说明结构化程序图中常见的错误。

（1）在流程图中，漏画流程方向，如图6-27中用圆圈着的地方所示。



该图中还有
其他的问题。
请读者分析。

图6-27 漏画流程方向

（2）在流程图中，没有进行抽象与归纳，如图6-28所示。

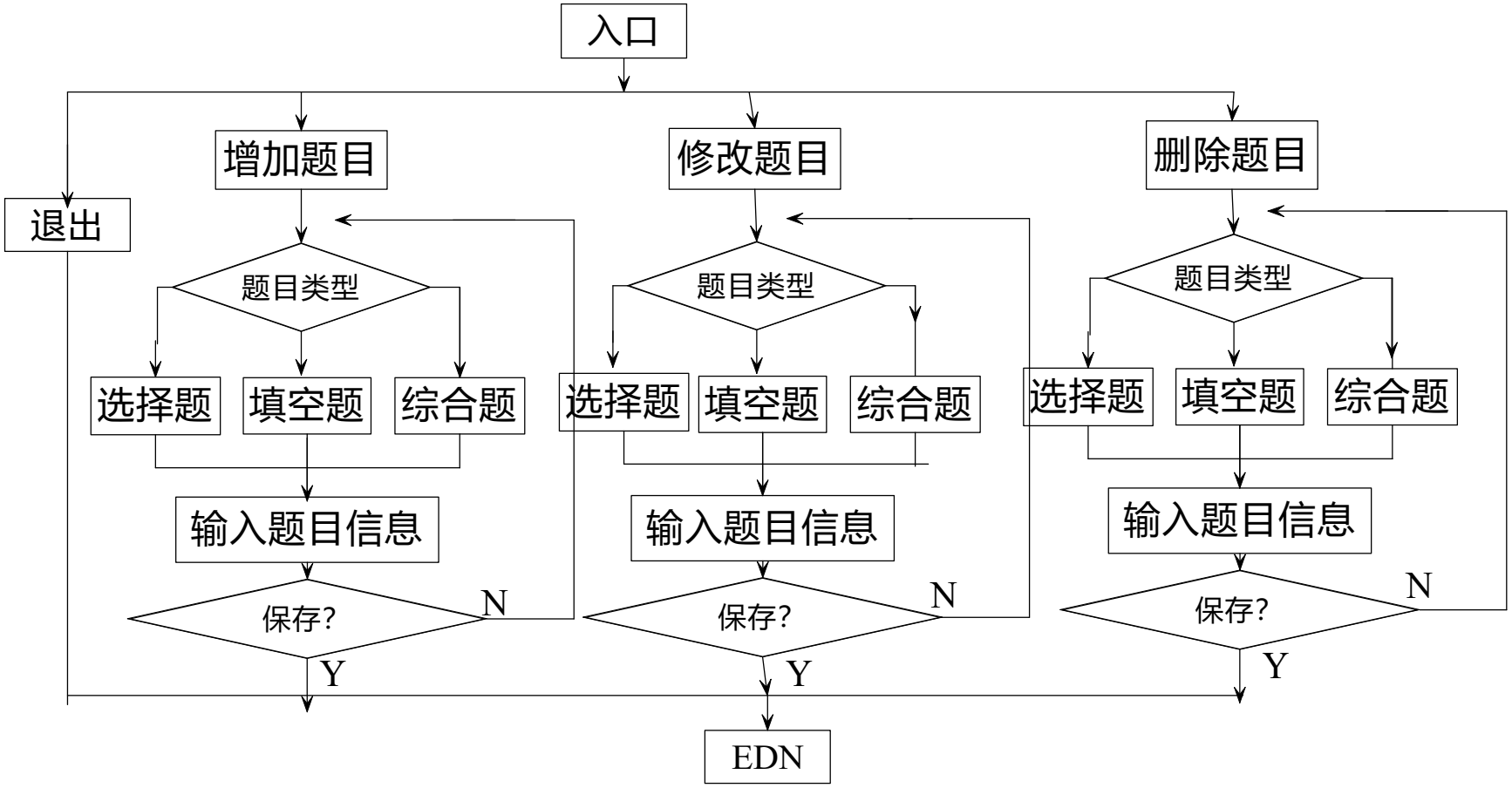


图6-28 没有进行抽象与归纳

（3）在流程图中，随意的用一些图形符号。如图6-29所示，图中用椭圆来代替表示用菱形表示的判断结点。

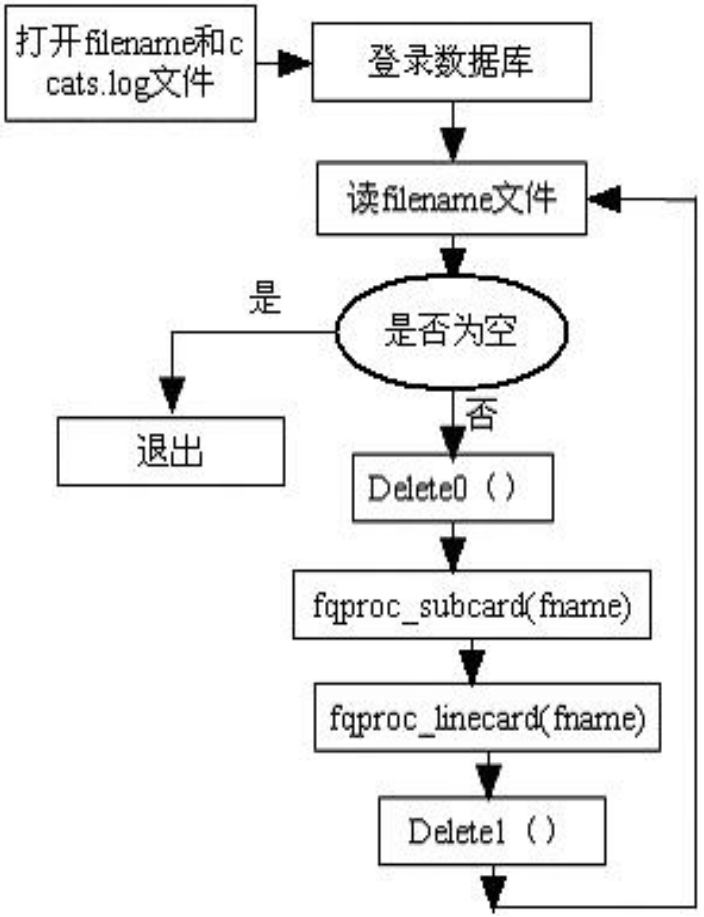


图6-29 随意的用一些图形符号

（4）在流程图中没有出口。如图6-30所示，流程图中就没有出口。

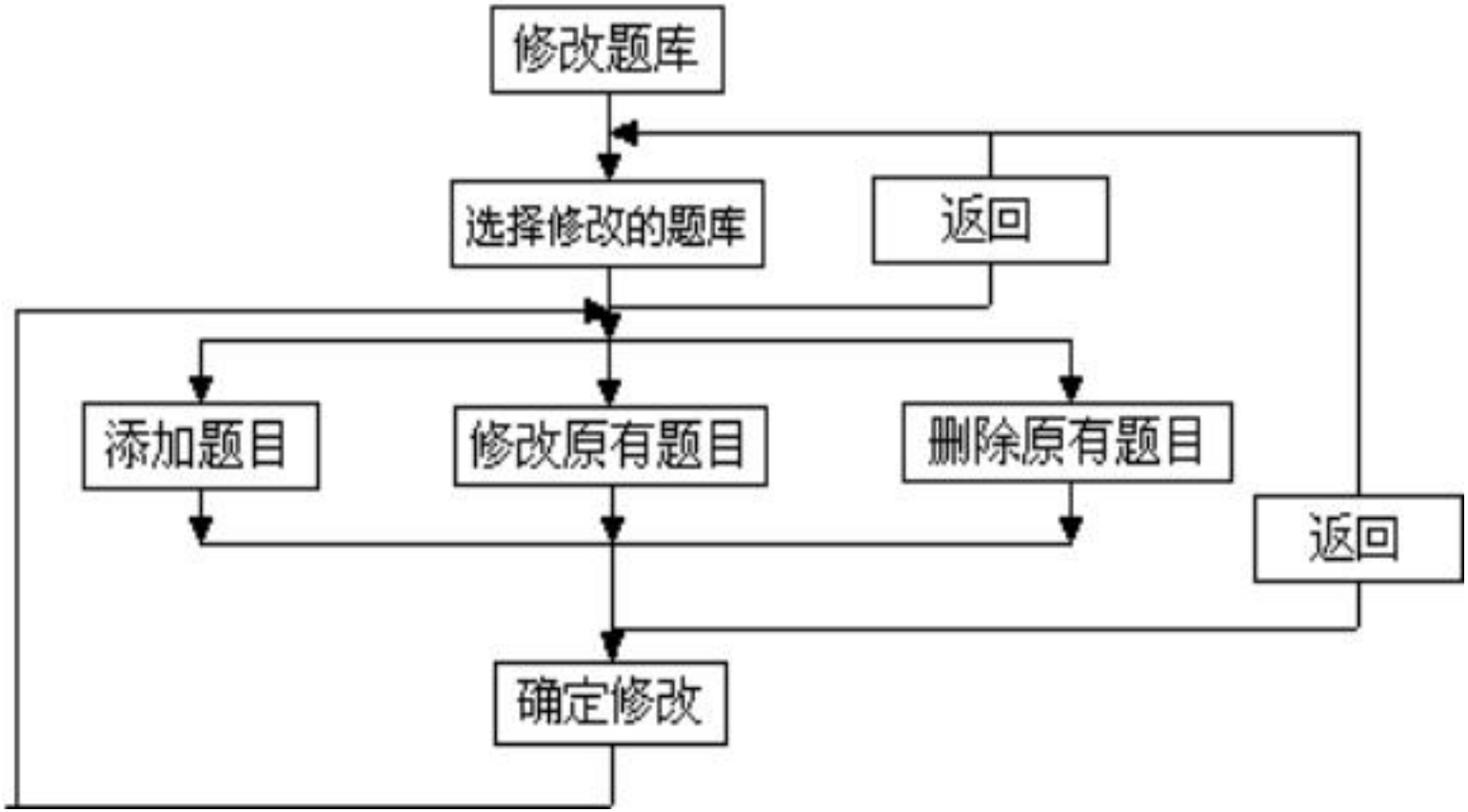


图6-30 没有出口

（5）在流程图中，随意使用图形符号和流向箭头。如图6-31所示中用椭圆圈着的地方，可以看出随意使用符号。类似这样的例子还有，因此在设计中一定要注意。

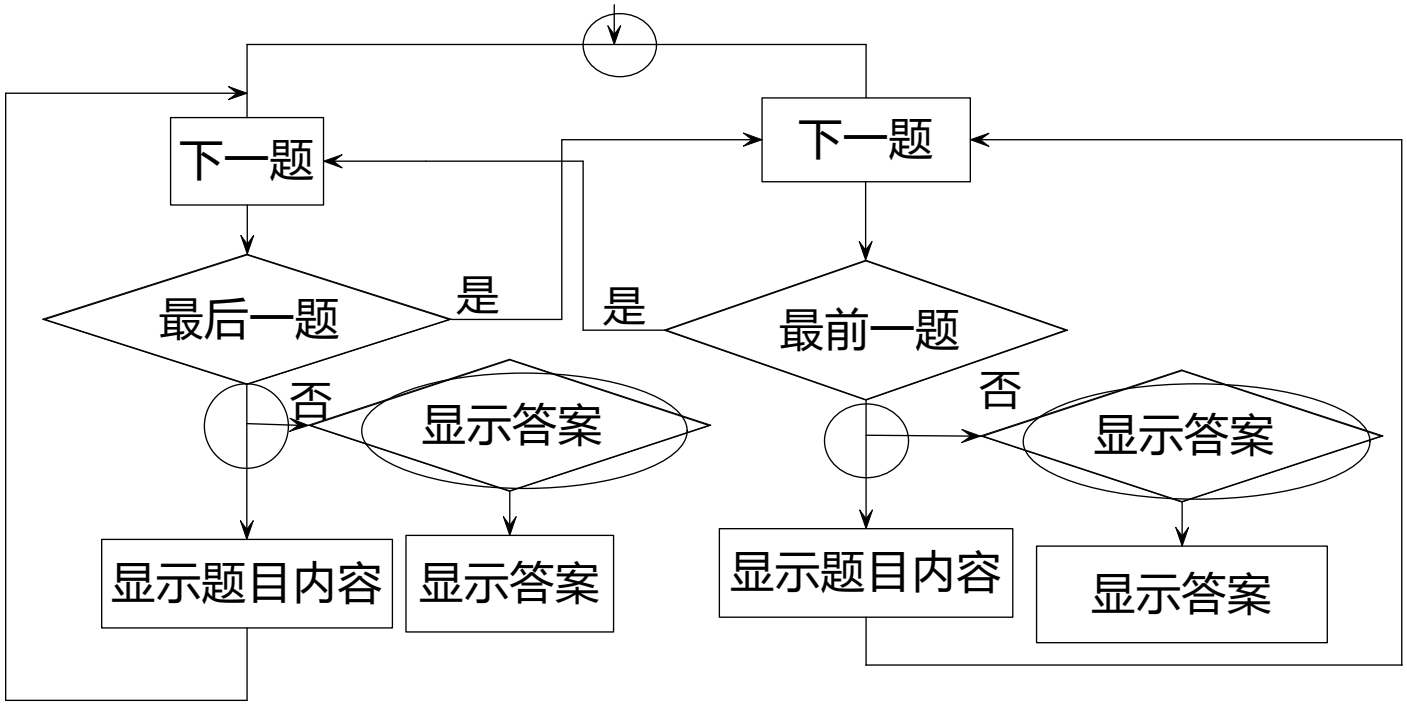
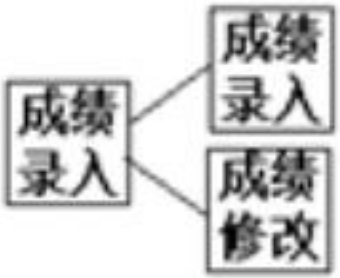


图6-31 随意使用图形符号和流向箭头

6.4 结构化详细设计应用实例

6.4.2 模块结构设计



6.4.3 数据结构设计

学生-课程结构数据设计如下：

名称	字段名称	类型	长度	允许空
学生学号	Sid	varchar	50	no
课程编号	Crid	varchar	50	no
教师工号	Tid	varchar	50	no
学生成绩	Score	float		yes
学年度	Cryear	varchar	50	yes

6.4.4 算法设计

教师的录入成绩（修改）服务程序流程逻辑如图6-16所示：

Insert into S_Score(Sid,Crid,Tid,Score,Cryear)

Values(@xh,@kch,@jgh,@cj,@xn)

//说明：xh,kch,jgh,cj,xn等分别是保存学号、课程编号、教师工号、学生成绩和学年度的变量

小结

本章介绍了详细设计的基本原理和方法。详细设计的任务是确定怎样具体实现所要求的目标系统，就是要设计出系统的程序蓝图。除了保证次序的可靠外，写出来的次序还要可读性好，易于理解，容易修改，容易维护。

在详细设计中，一般都要涉及到问题域中的具体过程，发现和表达问题域中的过程可以用结构化语言、判定表、判定树。在用流程图描述问题域的过程时，在结构化方法的详细设计中，结构程序设计要用到流程图程序、程序的三种基本控制结构、正规程序、基本程序、结构化程序、结构化定理、程序函数这些结构化程序的基本概念。如果发现非结构化程序，就要将其转换为结构化程序。

面向数据结构的设计是详细设计中常用的方法之一。面向数据流的设计和面向数据结构的设计的共同点都是数据信息驱动的，都试图将数据表示转换成软件表示。不同之处在于面向数据结构的设计不利用数据流图，而根据数据结构的表示来设计。有关面向数据结构的设计方法由于现在比较少用，所以这里没有详细介绍。

谢谢