第4章 总体设计

- 软件设计的概念及原则
- 结构化设计
- 体系结构设计
- 接口设计
- 数据设计
- 过程设计
- 软件设计规格说明

4.1 软件设计的概念及原则

- 软件设计的概念

- **设计是一项核心的工程活动。**
- **在20世纪90年代早期，Lotus 1-2-3的发明人Mitch Kapor在Dr. Dobbs杂志上发表了“软件设计宣言”，其中指出：**

  “什么是设计？设计是你站在两个世界——技术世界和人类的目标世界——而你尝试将这两个世界结合在一起……”。

4.1 软件设计的概念及原则

- 软件设计的概念
  - **罗马建筑批评家Vitruvius提出了这样一个观念：**

    “设计良好的建筑应该展示出坚固、适用和令人赏心悦目”。

4.1 软件设计的概念及原则

- 软件设计的原则

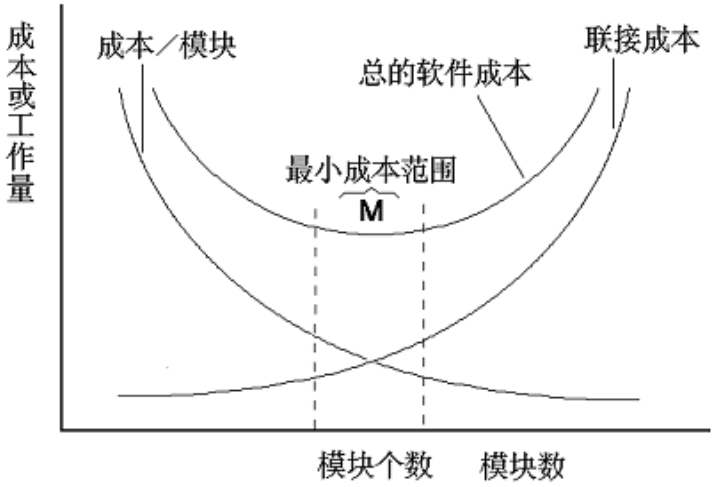
(1) 分而治之

- **分而治之是人们解决大型复杂问题时通常采用的策略。将大型复杂的问题分解为许多容易解决的小问题，原来的问题也就容易解决了。**
- **软件的体系结构设计、模块化设计都是分而治之策略的具体表现。**

4.1 软件设计的概念及原则

(1) 分而治之

- **尽管模块分解可以简化要解决的问题，但模块分解并不是越小越好。**
- **当模块数目增加时，每个模块的规模将减小，开发单个模块的成本确实减少了；但是，随着模块数目增加，模块之间关系的复杂程度也会增加，设计模块间接口所需要的工作量也将增加，如图所示。**



4.1 软件设计的概念及原则

(2) 模块独立性

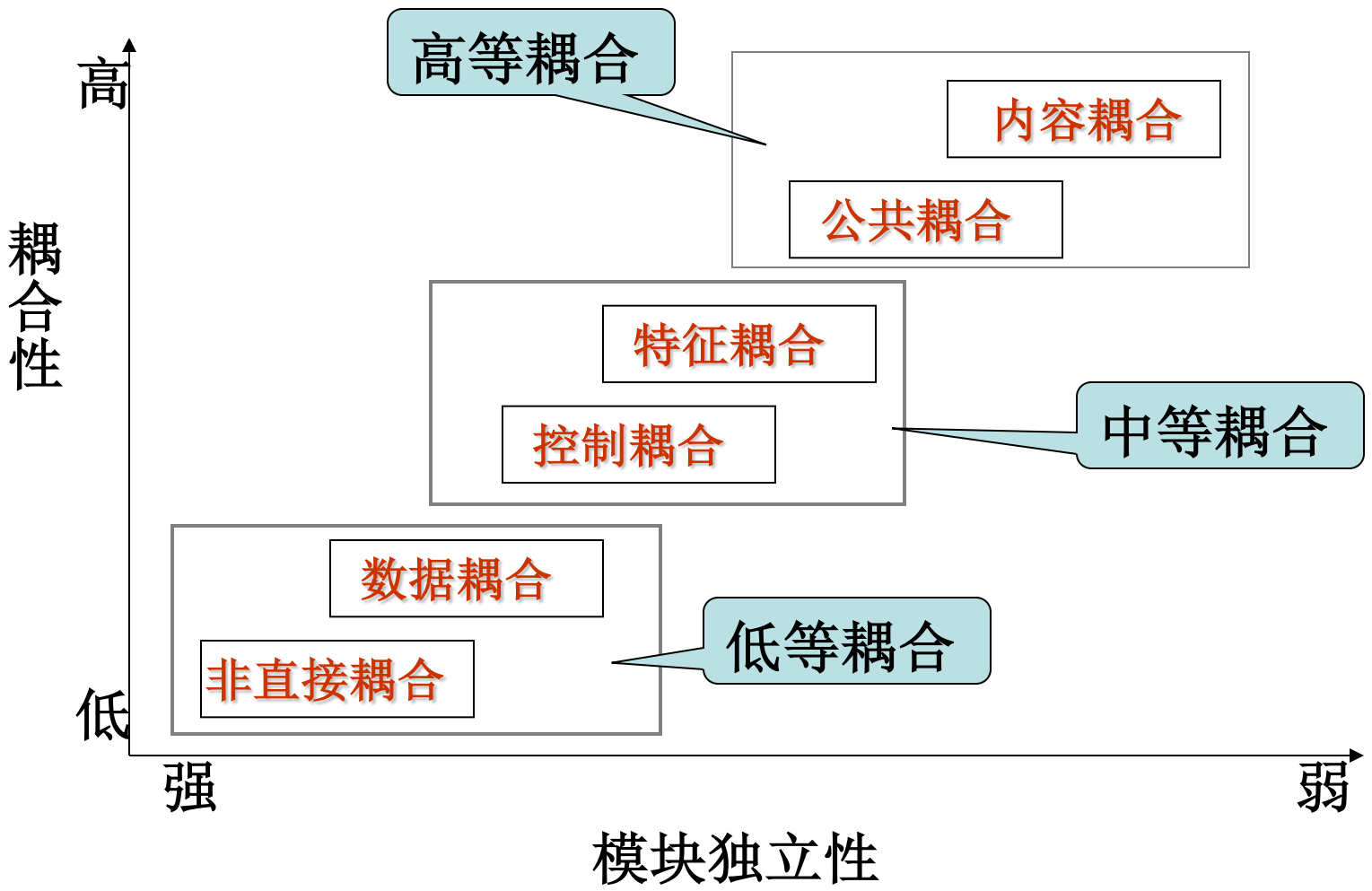
- **定义: 是指软件系统中每个模块只涉及软件要求的具体的子功能, 而和软件系统中其它模块的接口是简单的。**
- **有效的模块化使软件便于分工协作开发。**
- **独立的模块比较容易测试和维护。**

4.1 软件设计的概念及原则

模块独立性的度量准则

- **耦合:是模块之间的互相连接的紧密程度的度量。**
- **内聚:是模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量。**
- **模块独立性比较强的模块应是高内聚低耦合的模块。**

4.1 软件设计的概念及原则

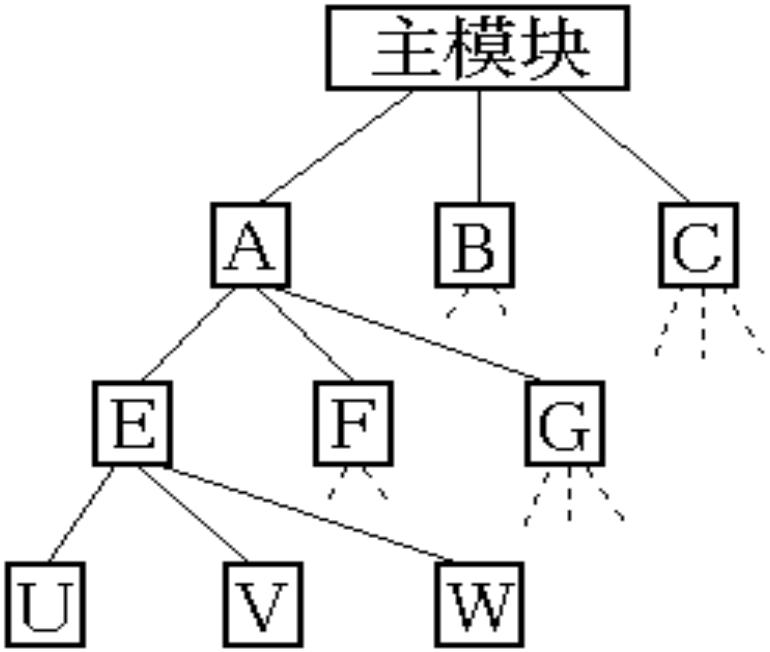


4.1 软件设计的概念及原则

非直接耦合(Nondirect Coupling)

●两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。

●非直接耦合的模块独立性最强。



4.1 软件设计的概念及原则

数据耦合 (Data Coupling)

一个模块访问另一个模块时，彼此之间是通过简单数据参数 (不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的。也是较理想的耦合。

4.1 软件设计的概念及原则

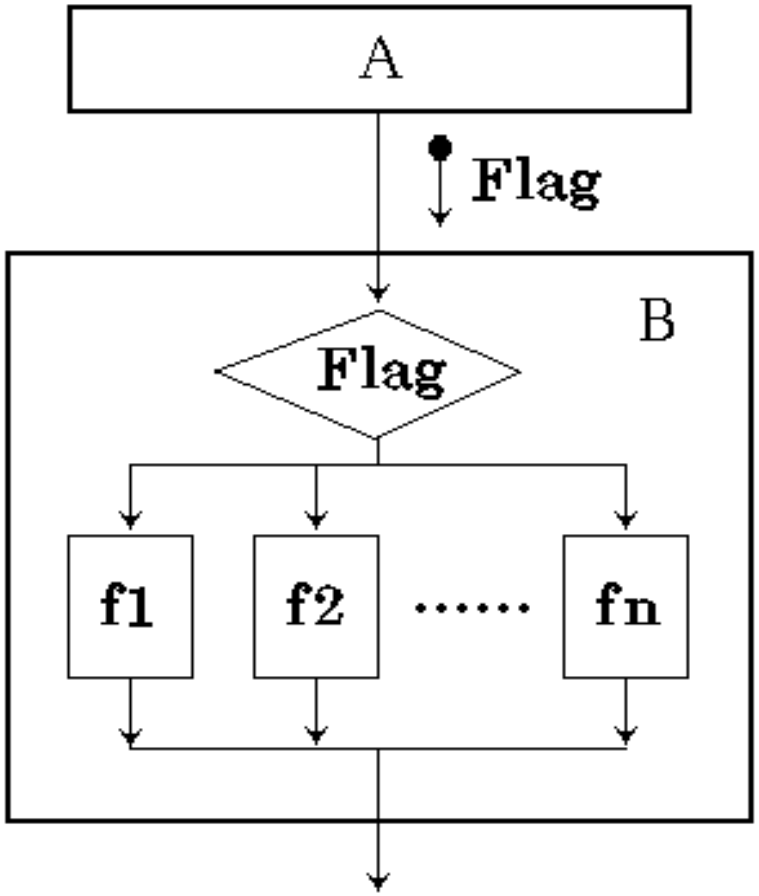
特征耦合 (Stamp Coupling)

一组模块通过参数表传递记录信息，就是特征耦合。这个记录是某一数据结构的子结构，而不是简单变量。

4.1 软件设计的概念及原则

控制耦合 (Control Coupling)

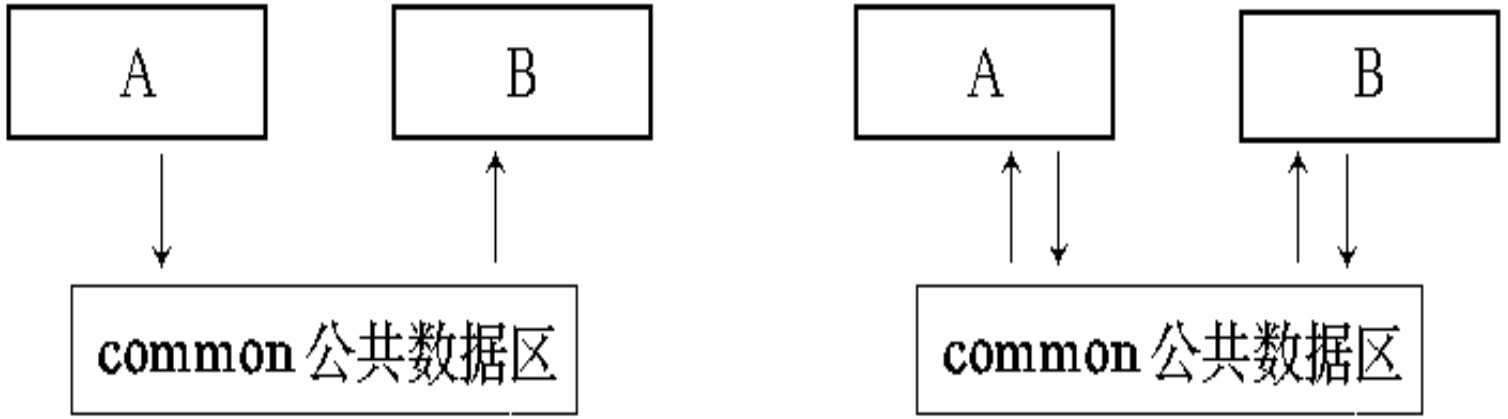
如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。



4.1 软件设计的概念及原则

●公共耦合（Common Coupling）

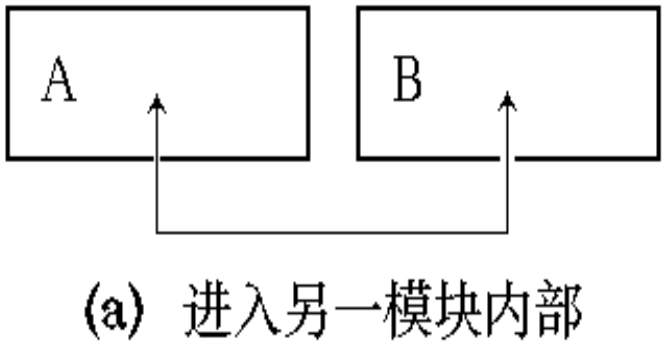
- 若一组模块都访问同一个**公共数据环境**，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。



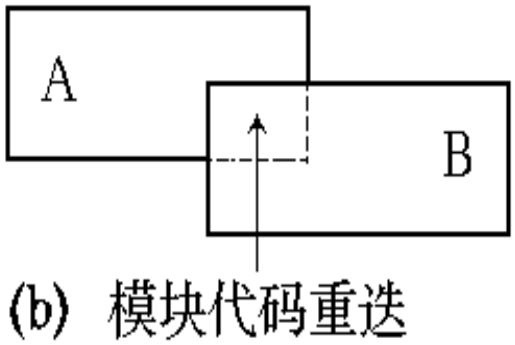
(a) 松散的公共耦合　　(b) 紧密的公共耦合

4.1 软件设计的概念及原则

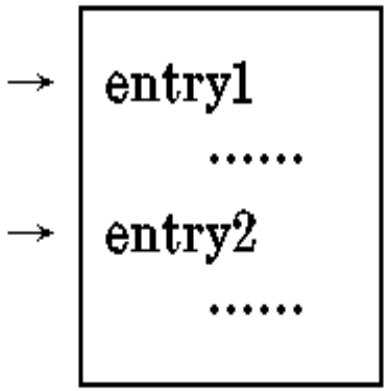
- **内容耦合 (Content Coupling)**
  - 一个模块直接访问另一个模块的内部数据。
  - 一个模块不通过正常入口转到另一模块内部。
  - 两个模块有一部分程序代码重迭。(只可能出现在汇编语言中)。
  - 一个模块有多个入口。



(a) 进入另一模块内部



(b) 模块代码重迭

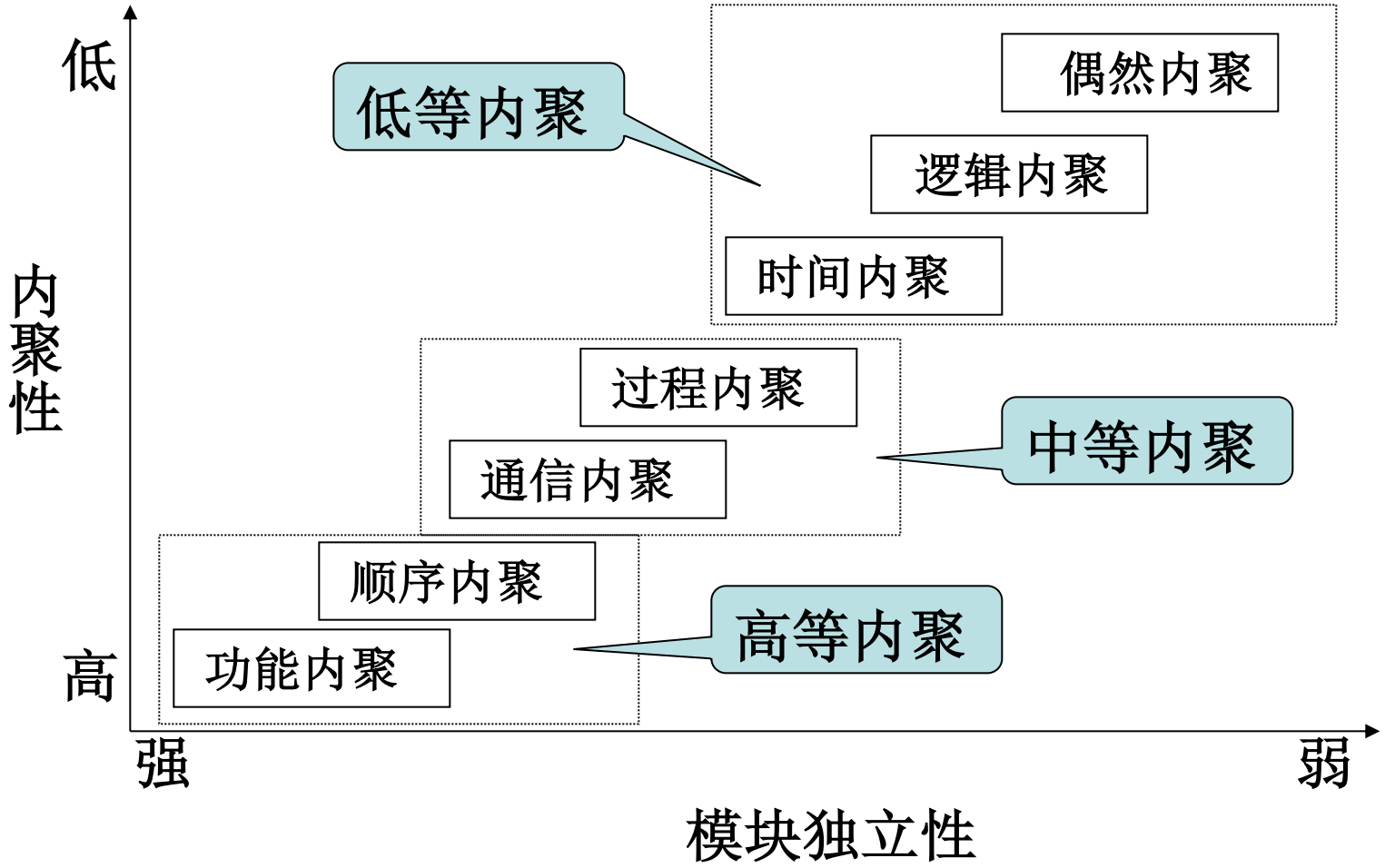


4.1 软件设计的概念及原则

- 设计原则
  - 尽量使用数据耦合
  - 少用控制耦合
  - 限制使用公共耦合（除非传递大量数据）
  - 完全不用内容耦合
- **实际上，两个模块之间的耦合不只是一种类型，而是多种类型的混合。这就要求设计人员进行分析、比较，逐步加以改进，以提高模块的独立性。**

4.1 软件设计的概念及原则

模块内聚



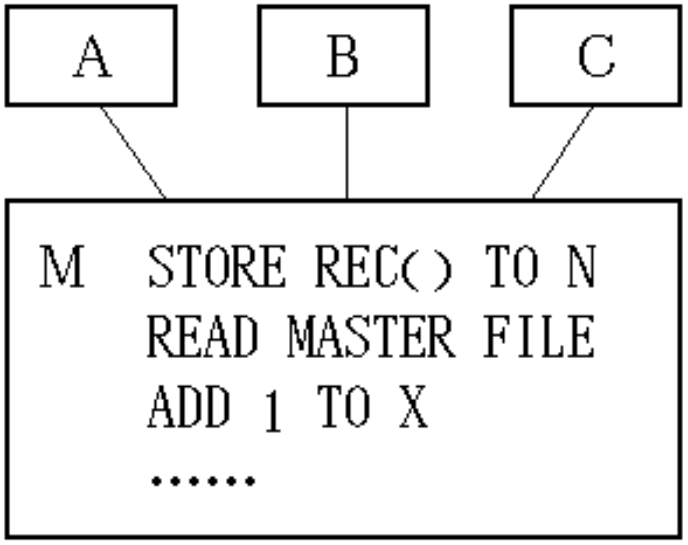
4.1 软件设计的概念及原则

偶然内聚（Coincidental Cohesion）

- 当模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为偶然内聚模块，内聚程度最低。
- **缺点：**

  1）内容不易理解，很难描述其功能。

  2）把完整的程序分割到多个模块中，在程序运行时会频繁地互相调用。



4.1 软件设计的概念及原则

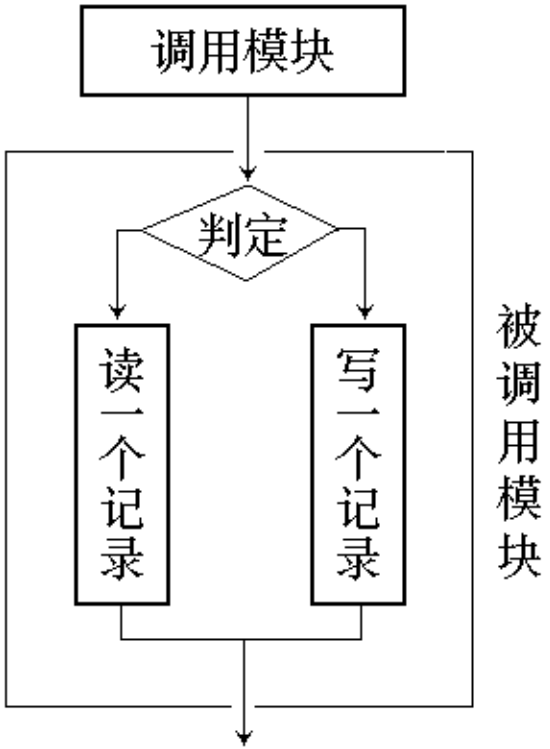
逻辑内聚（Logical Cohesion）

- 把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪个功能。
- 缺点

  1）不易修改，因包含多个功能

  2）需传递控制参数——控制耦合

  3）未用部分调入内存，影响效率



4.1 软件设计的概念及原则

时间内聚（Classical Cohesion）

- 时间内聚模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。

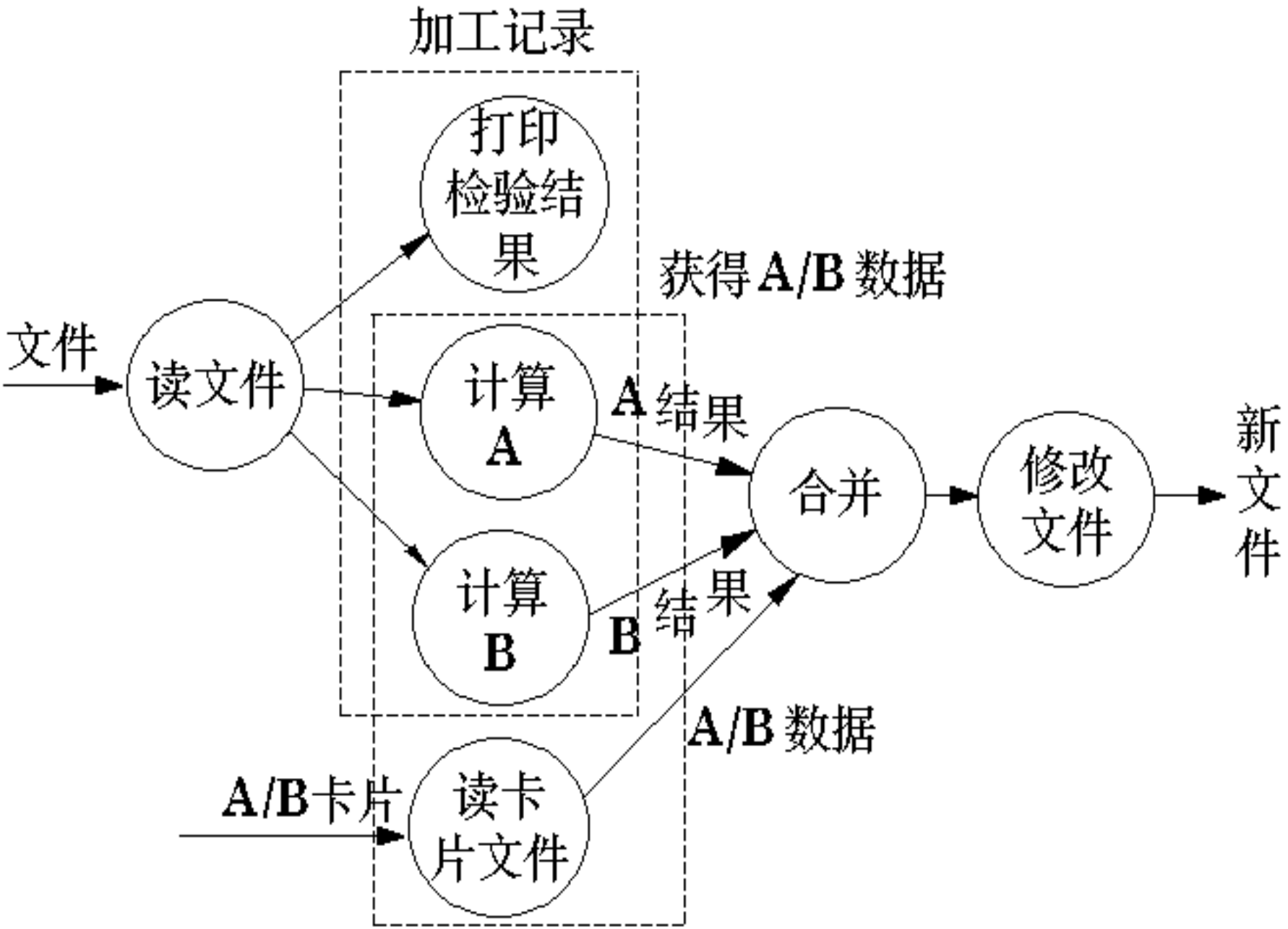
  例如：初始化模块和终止模块。

- 时间内聚模块比逻辑内聚模块的内聚程度又稍高一些。在一般情形下，各部分可以以任意的顺序执行，所以它的内部逻辑更简单。

4.1 软件设计的概念及原则

过程内聚（Procedural Cohesion）

- 如果一个模块内的处理是相关的，而且必须以特定次序执行，则是过程内聚。
- 使用流程图做为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。
  例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。
- 过程内聚仅包含完整功能的一部分。


加工记录
打印检验结果
获得A/B数据
文件
读文件
计算A
A结果
合并
修改文件
新文件
计算B
B结果
A/B数据
A/B卡片
读卡片文件

4.1 软件设计的概念及原则

顺序内聚

- 一个模块中的处理元素和同一功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。
- 根据数据流图划分模块时，通常得到顺序内聚的模块。

4.1 软件设计的概念及原则

功能内聚 (Functional Cohesion)

- 一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的。则称该模块为功能内聚模块。
- 优点：容易修改和维护

4.1 软件设计的概念及原则

(3) 提高抽象层次

- **抽象是指忽视一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。**
- **当我们进行软件设计时，设计开始时应尽量提高软件的抽象层次，按抽象级别从高到低进行软件设计。**

4.1 软件设计的概念及原则

(4) 复用性设计

- **复用是指同一事物不做修改或稍加修改就可以多次重复使用。将复用的思想用于软件开发，称为软件复用。**
- **我们将软件的重用部分称为软构件。**
- **也就是说，在构造新的软件系统时不必从零做起，可以直接使用已有的软构件即可组装（或加以合理修改）成新的系统。**

4.1 软件设计的概念及原则

(5) 灵活性设计

- **保证软件灵活性设计的关键是抽象。**
- **面向对象系统中的类结构类似一座金字塔，越接近金字塔的顶端，抽象程度就越高。**
- **“抽象”的反义词是“具体”。理想情况下，一个系统的任何代码、逻缉、概念在这个系统中都应该是唯一的，也就是说不存在重复的代码。**

4.1 软件设计的概念及原则

- 在设计中引入灵活性的方法有：
  - **降低耦合并提高内聚（易于提高替换能力）；**
  - **建立抽象（创建有多态操作的接口和父类）；**
  - **不要将代码写死（消除代码中的常数）；**
  - **抛出异常（由操作的调用者处理异常）；**
  - **使用并创建可复用的代码。**

4.2 结构化设计

- 结构化设计的任务
- 结构化设计与结构化分析的关系
- 模块结构及表示
- 数据结构及表示

4.2.1 软件设计的任务

- 软件设计的主要任务是要解决如何做的问题，要在需求分析的基础上，建立各种设计模型，并通过对设计模型的分析和评估，来确定这些模型是否能够满足需求。
- 软件设计是将用户需求准确地转化成为最终的软件产品的唯一途径，在需求到构造之间起到了桥梁作用。
- 在软件设计阶段，往往存在多种设计方案，通常需要在多种设计方案之中进行决策和折中，并使用选定的方案进行后续的开发活动。

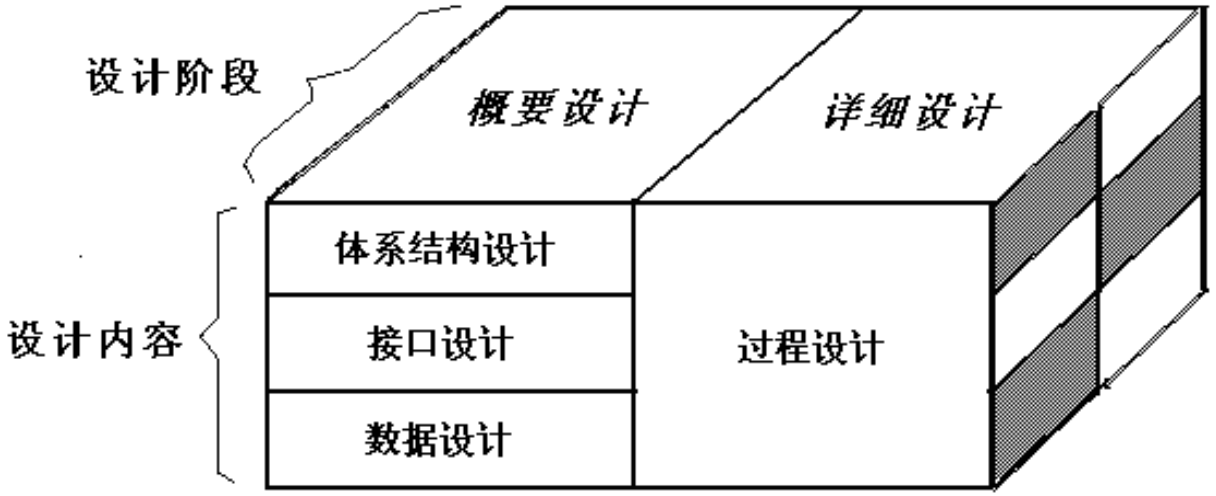
4.2.1 软件设计的任务

- 软件设计的阶段与任务

- **从工程管理的角度，可以将软件设计分为概要设计阶段和详细设计阶段。**
- **从技术的角度，传统的结构化方法将软件设计划分为体系结构设计、数据设计、接口设计和过程设计4部分；**
- **面向对象方法则将软件设计划分为体系结构设计、类设计/数据设计、接口设计和构件级设计4部分。**

4.2.1 软件设计的任务

- 软件设计的阶段与任务
  - **从管理和技术两个不同的角度对设计的认识。**



4.2.1 软件设计的任务

- 软件设计的阶段与任务

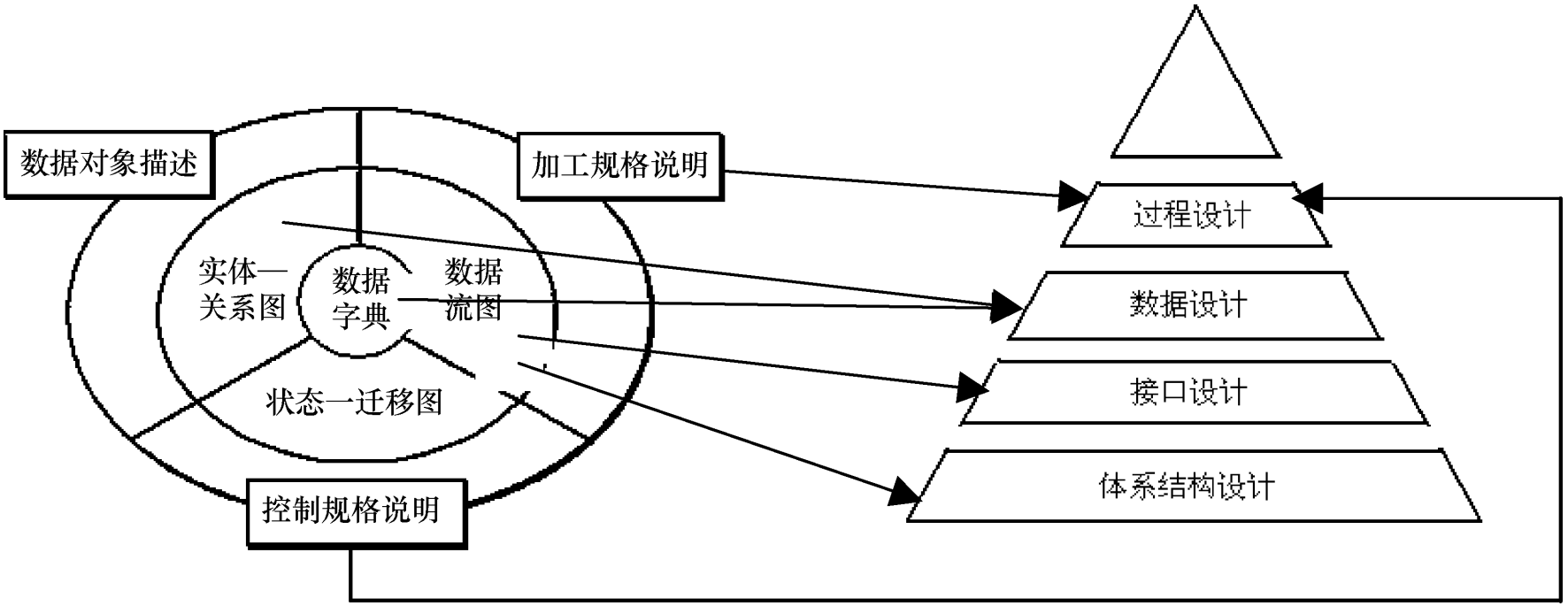
- **体系结构设计**：体系结构设计定义软件的主要结构元素及其之间的关系。
- **接口设计**：接口设计描述用户界面，软件和其他硬件设备、其他软件系统及使用人员的外部接口，以及各种构件之间的内部接口。
- **数据设计**：传统方法主要根据需求阶段所建立的实体—关系图（ER图）来确定软件涉及的文件系统的结构及数据库的表结构。

4.2.1 软件设计的任务

- 软件设计的阶段与任务
  - **过程设计**：**过程设计的主要工作是确定软件各个组成部分内的算法及内部数据结构，并选定某种过程的表达形式来描述各种算法。**

4.2.2 结构化设计与结构化分析的关系

- **结构化分析的结果为结构化设计提供了最基本的输入信息。两者的关系如图所示。**



4.2.2 结构化设计与结构化分析的关系

- 结构化设计方法的实施要点

(1) 研究、分析和审查数据流图。

(2) 根据数据流图决定问题的类型：变换型和事务型。针对两种不同的类型分别进行分析处理。

(3) 由数据流图推导出系统的初始结构图。

(4) 利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。

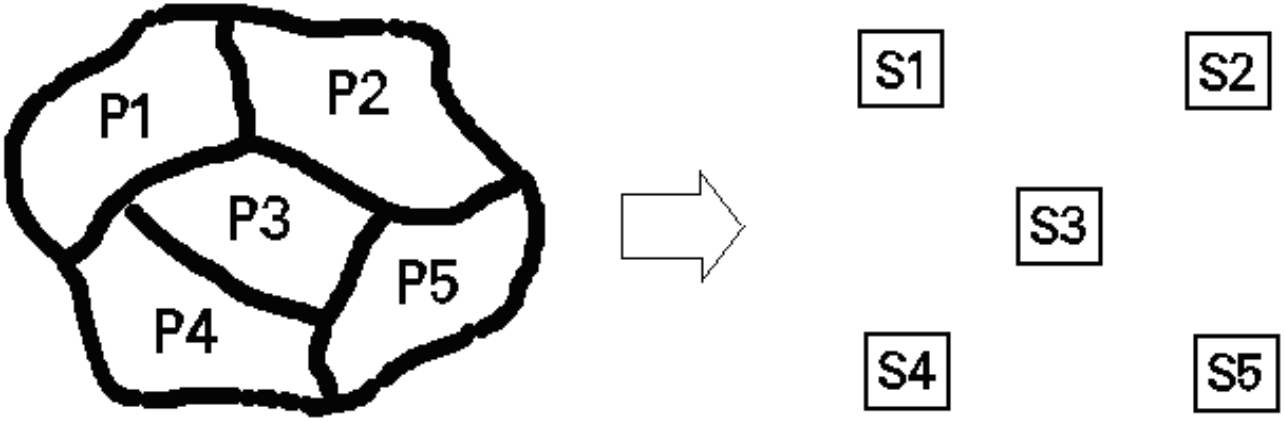
(5) 根据分析模型中的实体关系图和数据字典进行数据设计，包括数据库设计或数据文件的设计。

(6) 在上面设计的基础上，并依据分析模型中的加工规格说明、状态转换图进行过程设计。

(7) 制定测试计划。

4.2.3 模块结构及表示

- **一般通过功能划分过程来完成软件结构设计。功能划分过程从需求分析确立的目标系统的模型出发，对整个问题进行分割，使其每一部分用一个或几个软件模块加以解决，整个问题就解决了。**



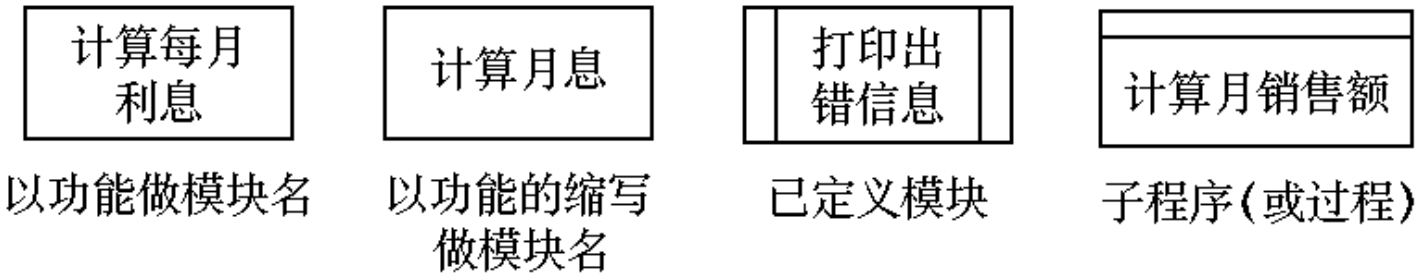
需要通过软件解决的问题　　通过功能划分得到的软件模块

4.2.3 模块结构及表示

- 模块

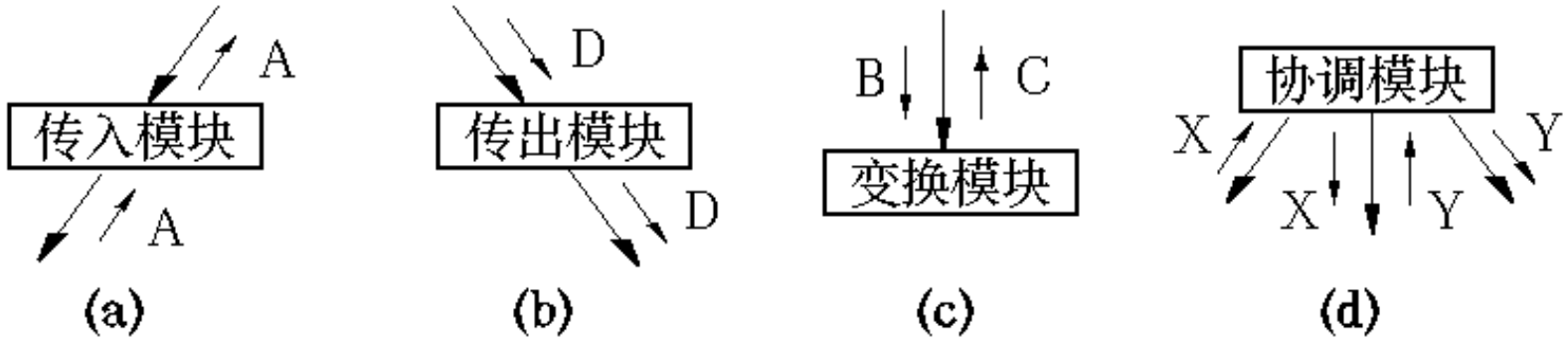
➢ **一个软件系统通常由很多模块组成，结构化程序设计中的函数和子程序都可称为模块，它是程序语句按逻辑关系建立起来的组合体。**

➢ **模块用矩形框表示，并用模块的名字标记它。**



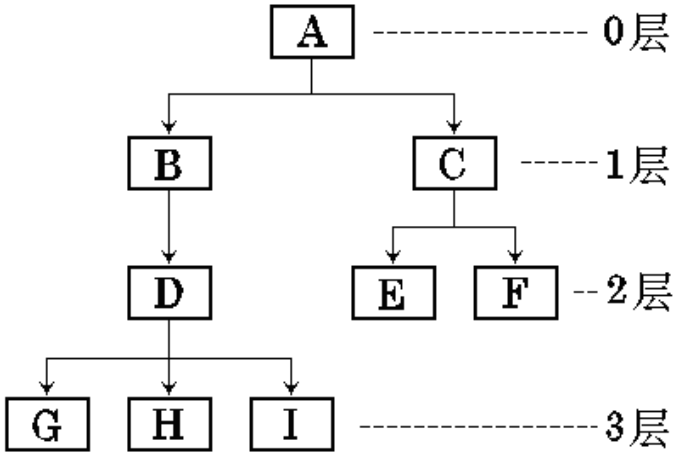
4.2.3 模块结构及表示

- 模块的分类

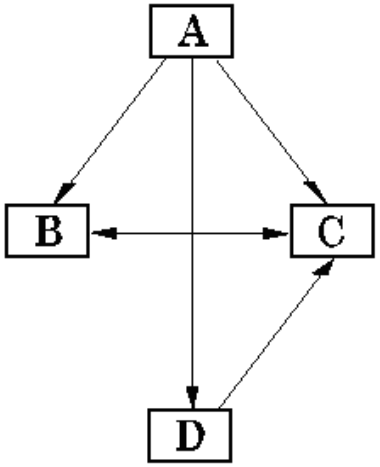


4.2.3 模块结构及表示

- 模块的结构
  - **模块结构最普通的形式就是树状结构和网状结构，如图所示。**



(a) 树状结构



(b) 网状结构

4.2.3 模块结构及表示

- 结构图

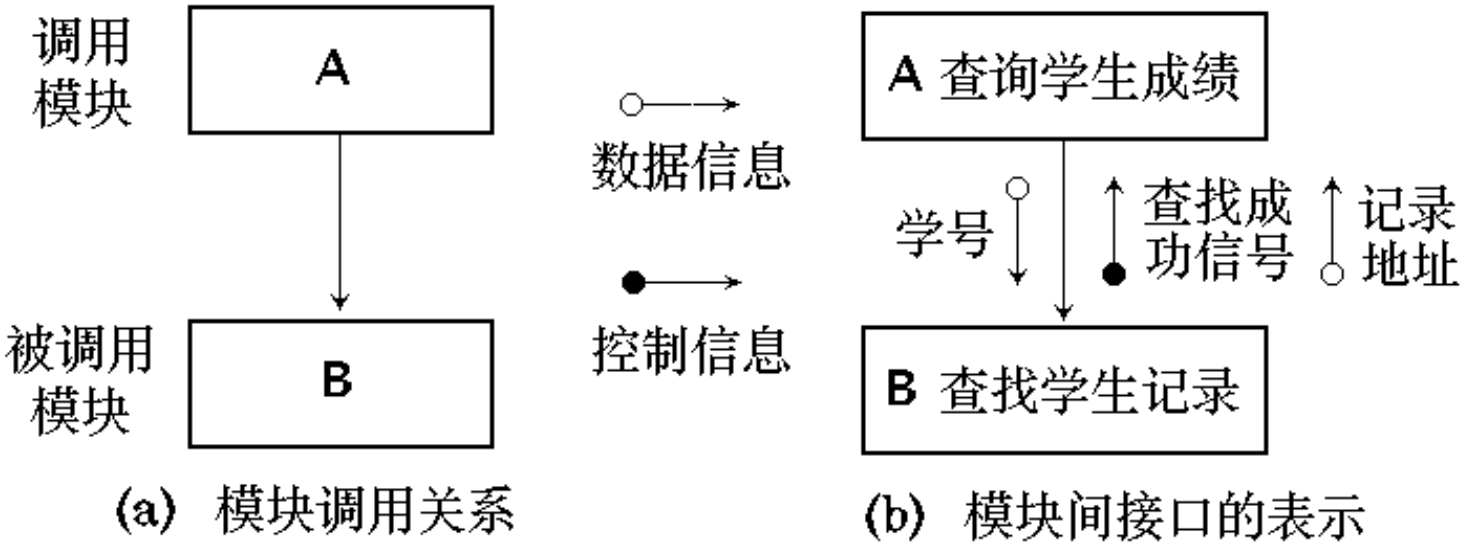
➢ **结构图（structure chart，SC）是精确表达模块结构的图形表示工具。**

(1) 模块的调用关系和接口：在结构图中，两个模块之间用单向箭头连接。

(2) 模块间的信息传递：当一个模块调用另一个模块时，调用模块把数据或控制信息传送给被调用模块，以使被调用模块能够运行。

4.2.3 模块结构及表示

- 结构图
  - **模块间的调用关系和接口表示**

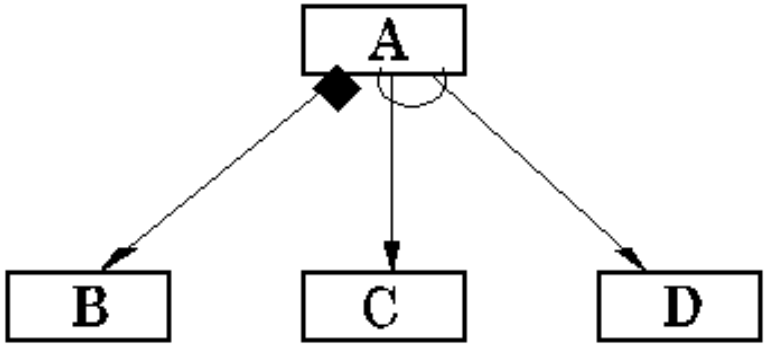


(a) 模块调用关系　　(b) 模块间接口的表示

4.2.3 模块结构及表示

- 结构图

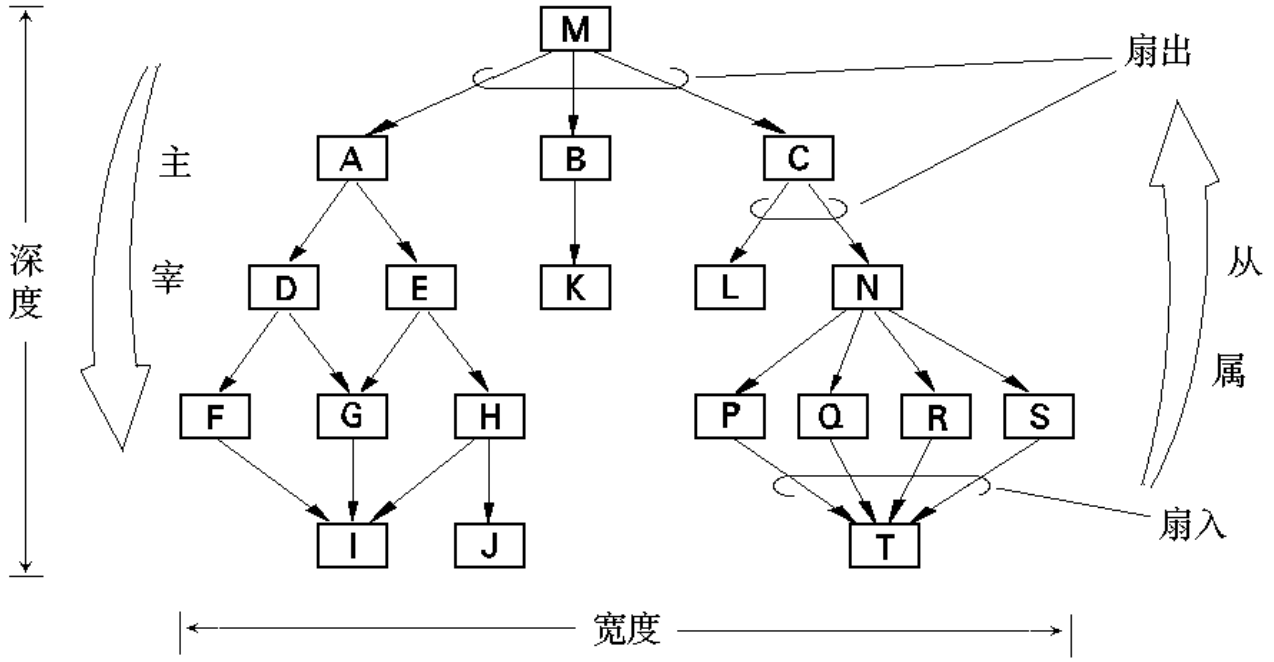
(3) 条件调用和循环调用 ：当模块A有条件地调用另一个模块B时，在模块A的箭头尾部标以一个菱形符号；当一个模块A反复地调用模块C和模块D时，在调用箭头尾部则标以一个弧形符号。



4.2.3 模块结构及表示

- 结构图

(4) 结构图的形态特征。在图中，上级模块调用下级模块，它们之间存在主从关系。



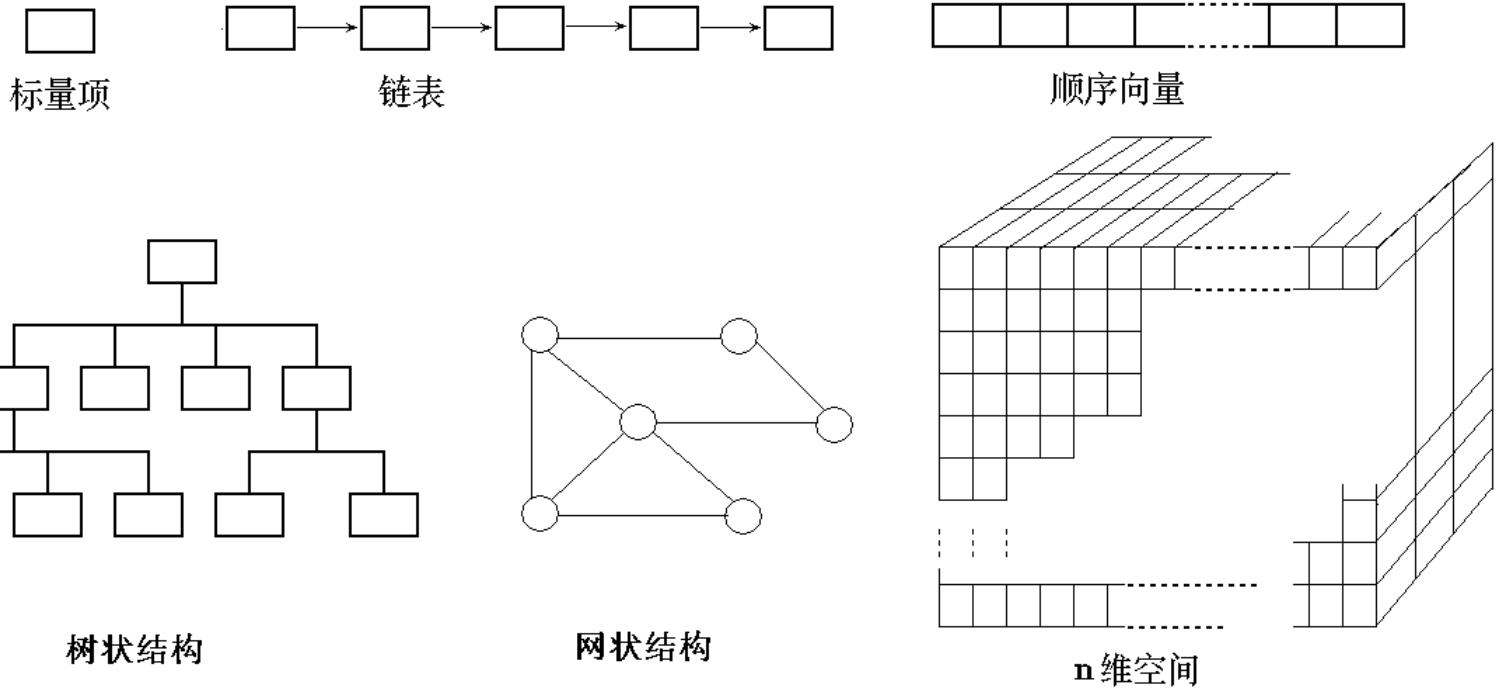
相关概念：宽度、深度、扇入、扇出。

4.2.4 数据结构及表示

- 数据结构是数据的各个元素之间逻辑关系的一种表示。
- 数据结构设计应确定数据的组织、存取方式、相关程度，以及信息的不同处理方法。
- 数据结构的组织方法和复杂程度可以灵活多样，但典型的数据结构种类是有限的，它们是构成一些更复杂结构的基本构件块。

4.2.4 数据结构及表示

- 典型的数据结构



4.3 体系结构设计

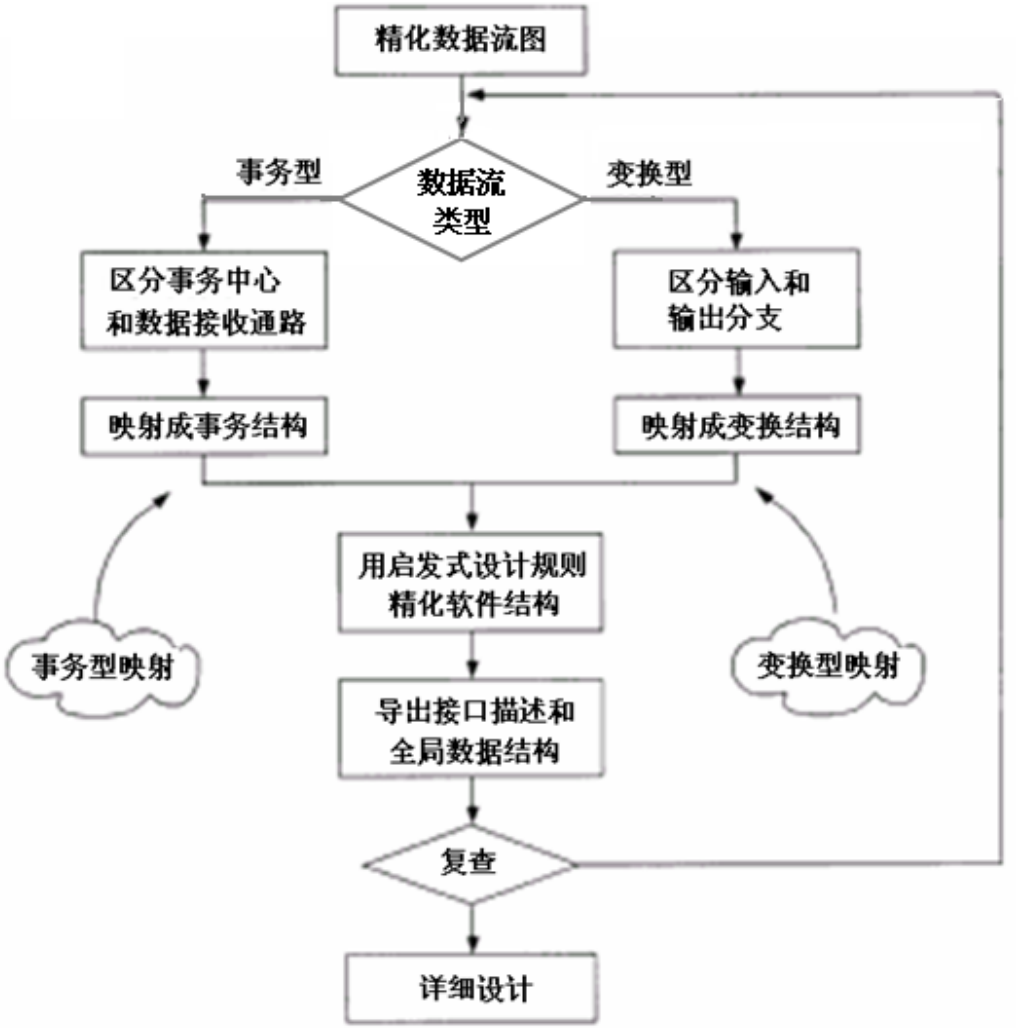
- 基于数据流方法的设计过程
- 典型的数据流类型和系统结构
- 变换型映射方法
- 事务型映射方法
- 软件模块结构的改进方法

4.3.1 基于数据流方法的设计过程

- **基于数据流的设计方法也称为过程驱动的设计方法；**
- **这种方法与软件需求分析阶段的结构化分析方法相衔接，可以很方便地将用数据流图表示的信息转换成程序结构的设计描述；**
- **这种方法还能和编码阶段的“结构化程序设计方法”相适应，成为常用的结构化设计方法。**

4.3.1 基于数据流方法的设计过程

- 设计过程
  - **基于数据流方法的设计过程**

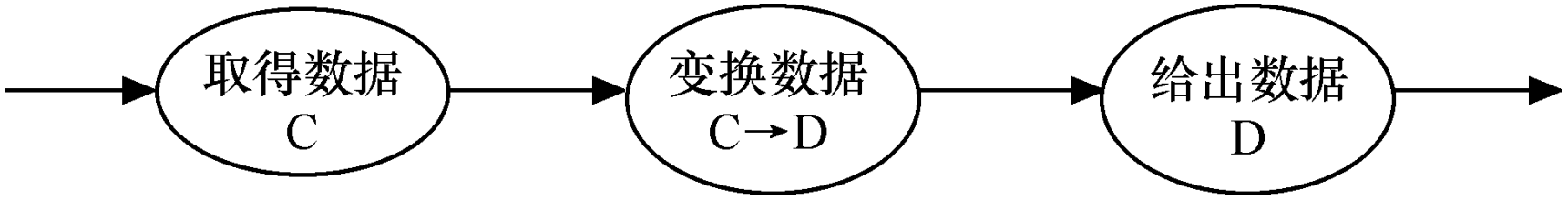


4.3.2 典型的数据流类型和系统结构

- 典型的数据流类型有变换型数据流和事务型数据流，数据流的类型不同，得到的系统结构也不同。
- 通常，一个系统中的所有数据流都可以认为是变换流，但是，当遇到有明显事务特性的数据流时，建议采用事务型映射方法进行设计。

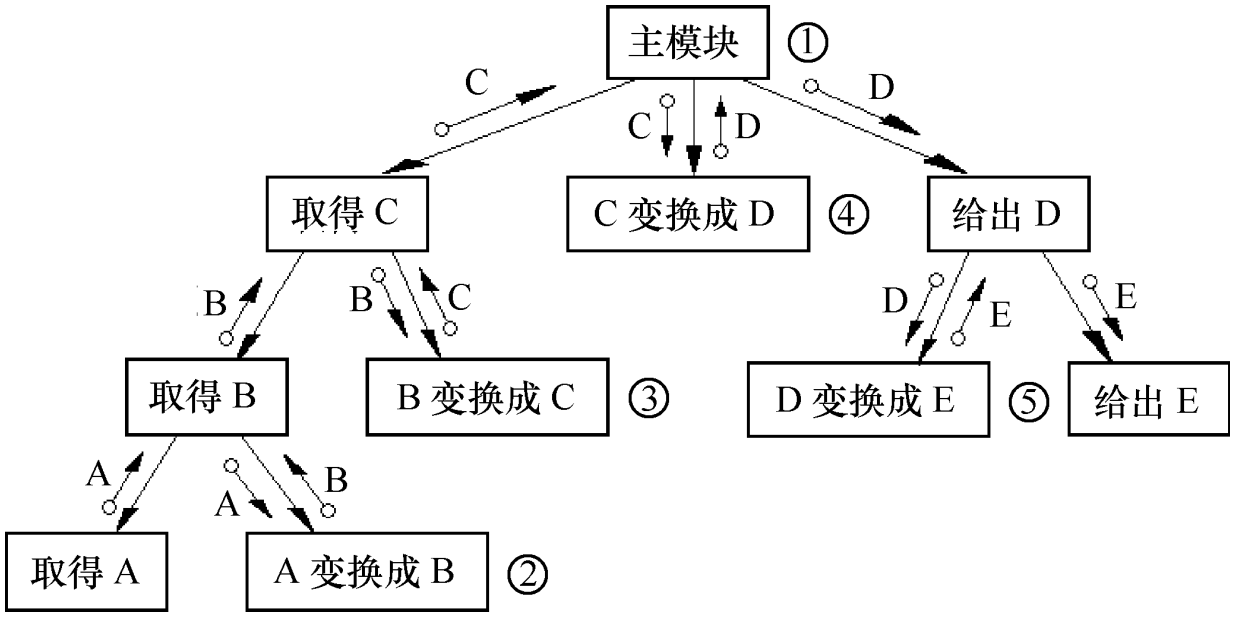
4.3.2 典型的数据流类型和系统结构

- 变换型数据流
  - **变换型数据处理问题的工作过程大致分为3步，即取得数据、变换数据和给出数据，如图所示。**



4.3.2 典型的数据流类型和系统结构

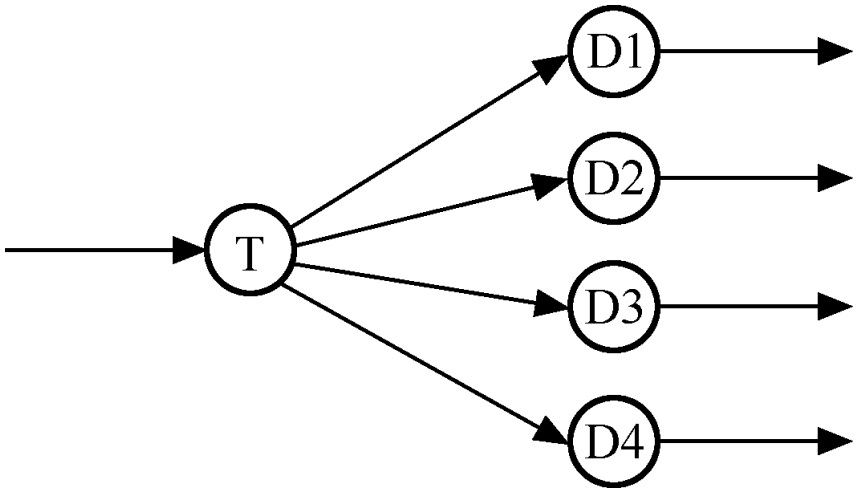
- 变换型系统结构图
  - **变换型系统的结构图由输入、中心变换和输出3部分组成。**



4.3.2 典型的数据流类型和系统结构

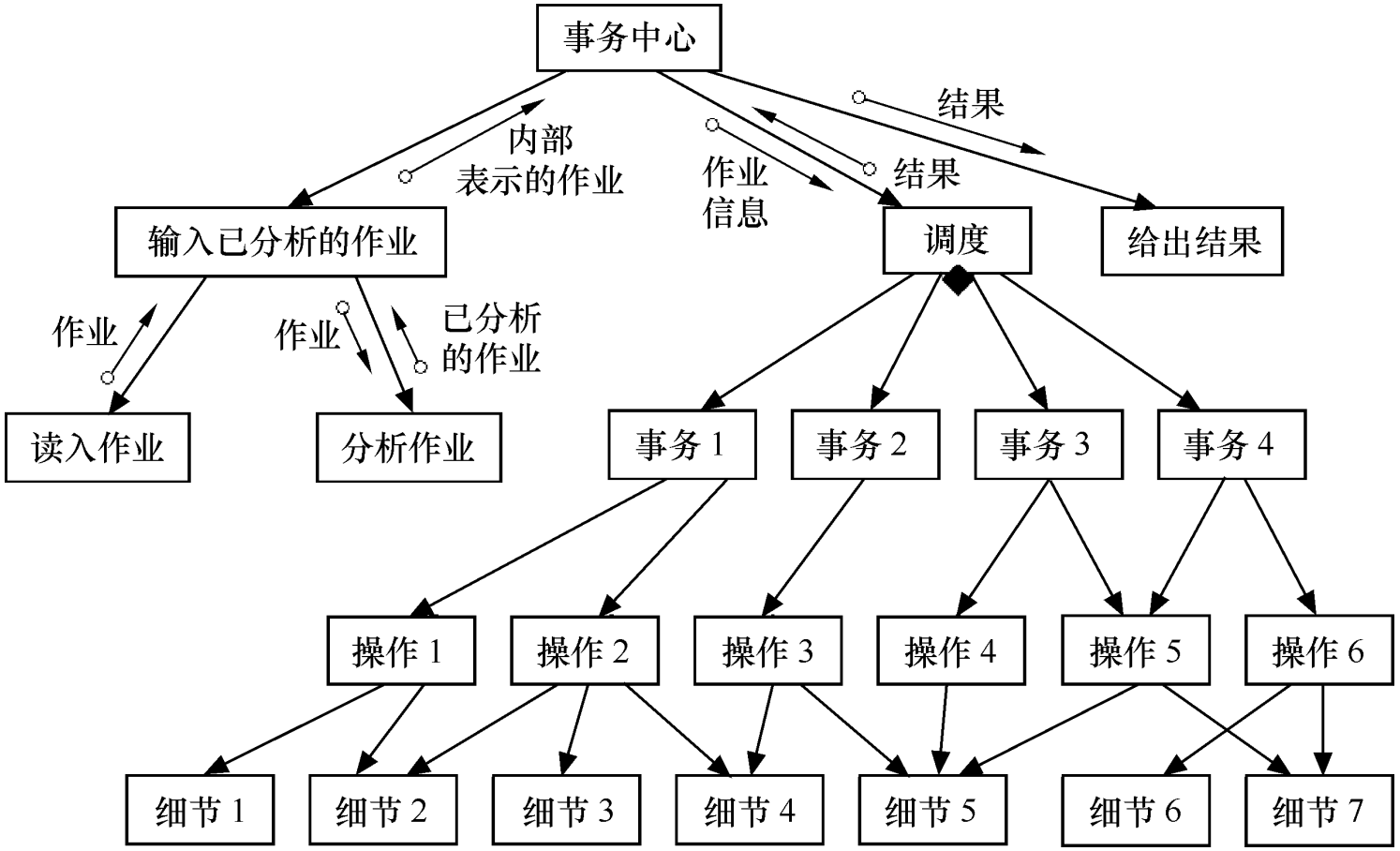
- 事务型数据流

- **通常接受一项事务，根据事务处理的特点和性质，选择分派一个适当的处理单元，然后给出结果。**
- **完成选择分派任务的部分称为事务处理中心，或分派部件。**



4.3.2 典型的数据流类型和系统结构

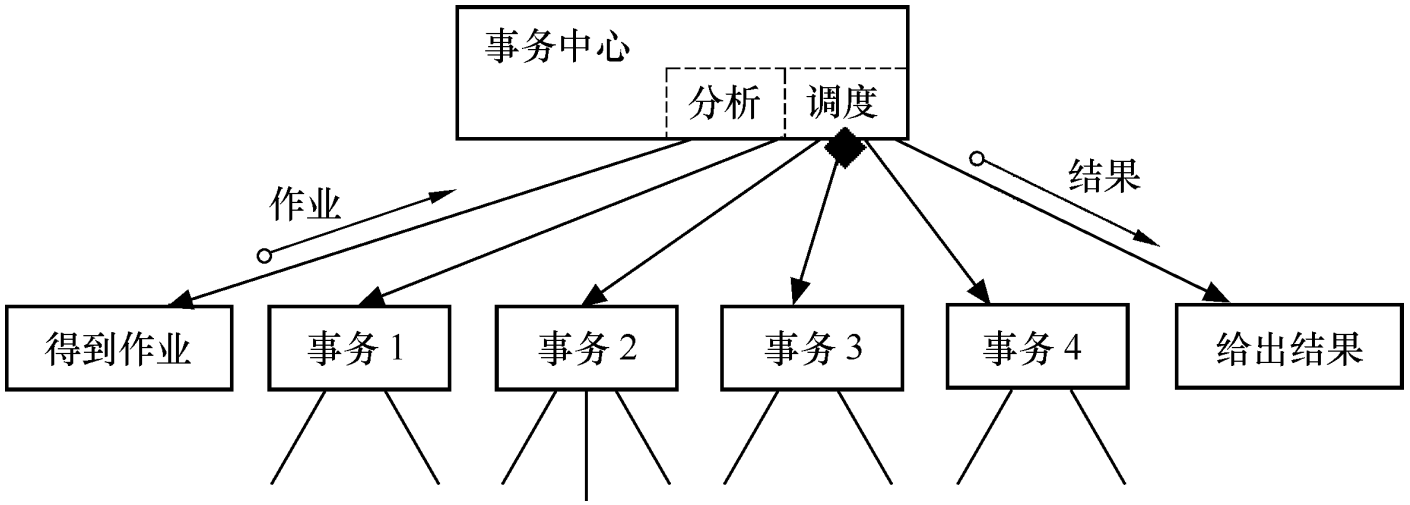
- 事务型系统结构图



4.3.2 典型的数据流类型和系统结构

- 简化的事务型系统结构图

- **事务型系统的结构图可以有多种不同的形式，如有多层操作层或没有操作层。**
- **如果调度模块并不复杂，可将其归入事务中心模块。**



4.3.3 变换型映射方法

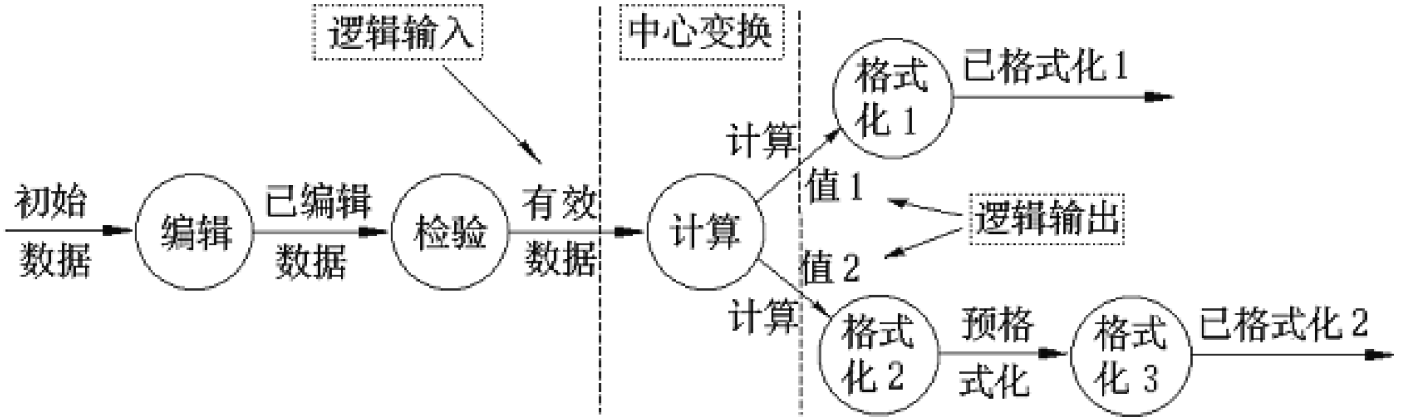
- **系统数据处理问题的处理流程总能表示为变换型数据流图，进一步可采用变换型映射方法建立系统的结构图。**
- **也可能遇到明显的事务数据处理问题，这时可采用事务型映射方法。**

4.3.3 变换型映射方法

- 变换分析方法的步骤

(1) 重画数据流图。在需求分析阶段得到的数据流图侧重于描述系统如何加工数据，而重画数据流图的出发点是描述系统中的数据是如何流动的。

(2)在数据流图上区分系统的逻辑输入、逻辑输出和中心变换部分。



4.3.3 变换型映射方法

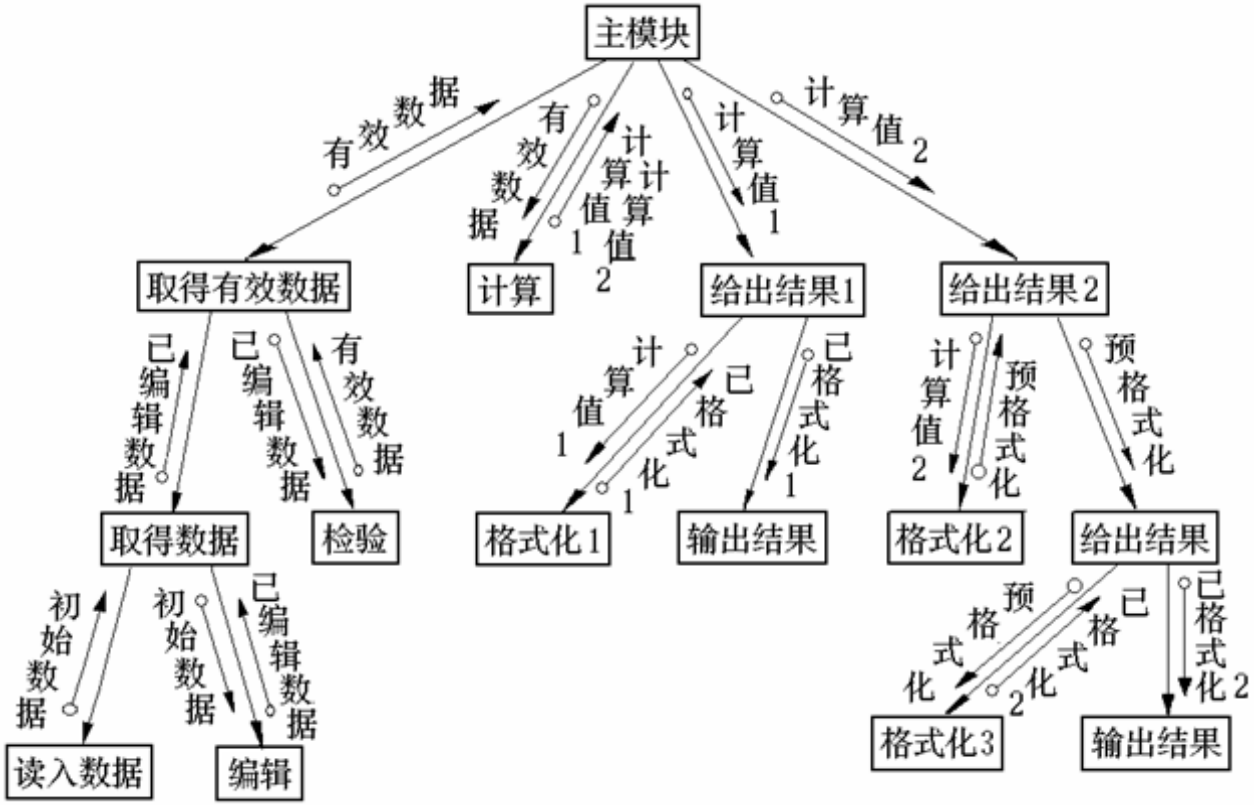
- 变换分析方法的步骤

(3) 进行一级分解，设计系统模块结构的顶层和第一层。自顶向下设计的关键是找出系统树形结构图的根或顶层模块。

- **首先设计一个主模块，并用程序的名字为它命名，然后将它画在与中心变换相对应的位置上。**
- **第1层设计：为每个逻辑输入设计一个输入模块，它的功能是为主模块提供数据；为每个逻辑输出设计一个输出模块，它的功能是将主模块提供的数据输出；为中心变换设计一个变换模块，它的功能是将逻辑输入转换成逻辑输出。**

4.3.3 变换型映射方法

- 变换分析方法的步骤
  - **第一层模块与主模块之间传送的数据应与数据流图相对应，如图所示。**



4.3.3 变换型映射方法

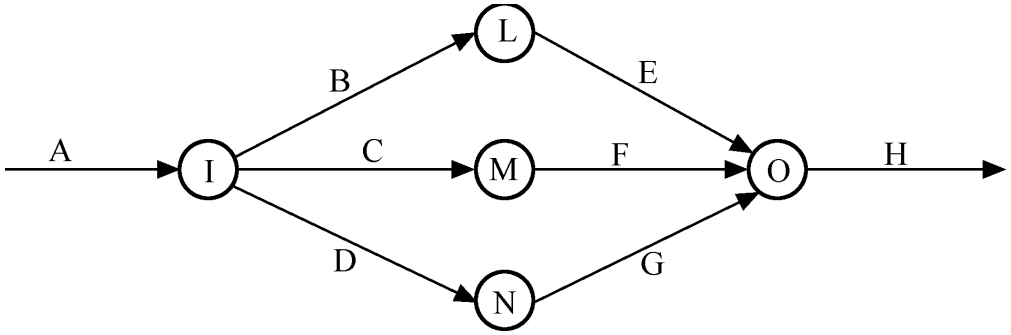
- 变换分析方法的步骤

(4) 进行二级分解，设计中、下层模块。

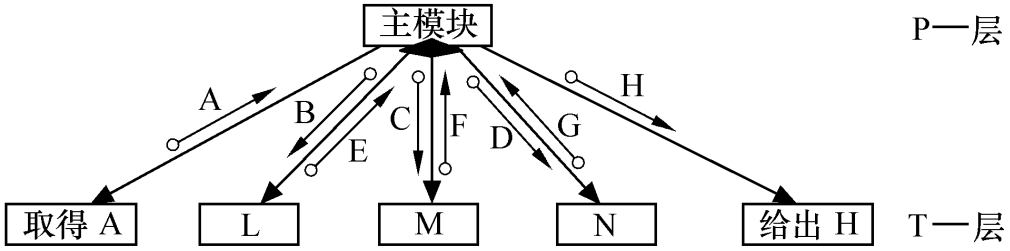
- **这一步工作是自顶向下，逐层细化，为每一个输入模块、输出模块、变换模块设计它们的从属模块。**
- **设计下层模块的顺序是任意的。但一般是先设计输入模块的下层模块。**

4.3.4 事务型映射方法

- **事务分析也是从分析数据流图开始，自顶向下，逐步分解，建立系统的结构图。**



（a）



（b）

4.3.4 事务型映射方法

- 事务分析方法的步骤

(1) **识别事务源**。利用数据流图和数据词典，从问题定义和需求分析的结果中，找出各种需要处理的事务。

(2) **规定适当的事务型结构**。在确定了该数据流图具有事务型特征之后，根据模块划分理论，建立适当的事务型结构。

(3) **识别各种事务和它们定义的操作**。

(4) **注意利用公用模块**。

4.3.4 事务型映射方法

- 事务分析方法的步骤

(5) 建立事务处理模块。对每一事务，或对联系密切的一组事务，建立一个事务处理模块。

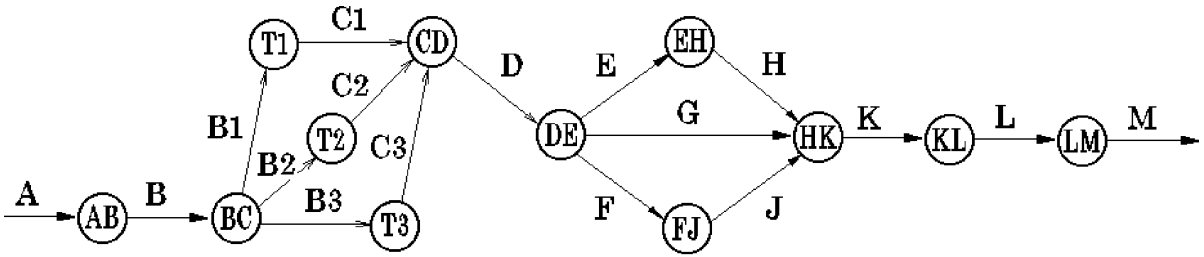
(6) 对事务处理模块规定它们全部的下层操作模块。

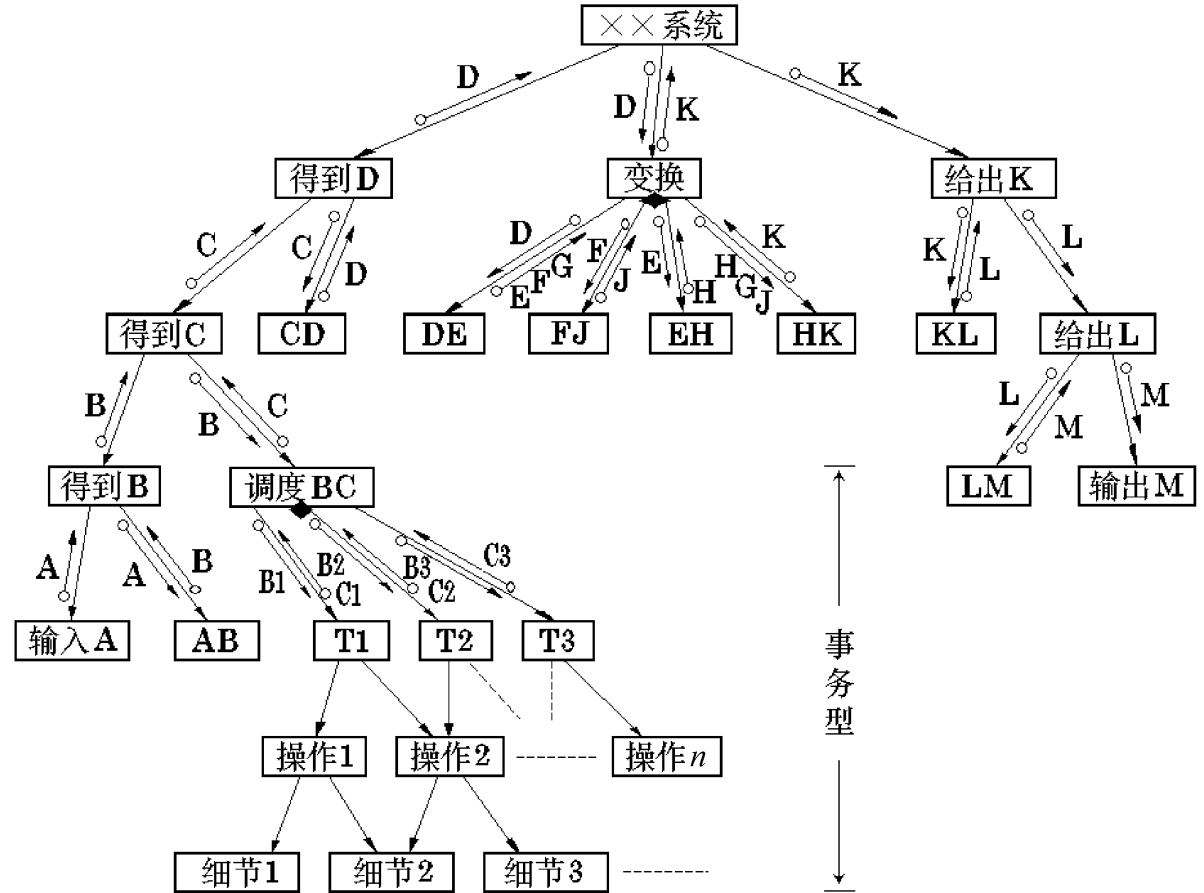
(7) 对操作模块规定它们的全部细节模块。

大型的软件系统通常是变换型结构和事务型结构的混合结构，所以，我们通常利用以变换分析为主，事务分析为辅的方式进行软件结构设计。

4.3.4 事务型映射方法

- 混合结构的例子





4.3.5 软件模块结构的改进方法

(1) **模块功能的完善化**。一个完整的功能模块，不仅能够完成指定的功能，而且还应当能够告诉使用者完成任务的状态，以及不能完成的原因。也就是说，一个完整的模块应当有以下几部分。

① 执行规定的功能的部分。

② 出错处理的部分。 当模块不能完成规定的功能时，必须回送出错标志，向它的调用者报告出现这种例外情况的原因。

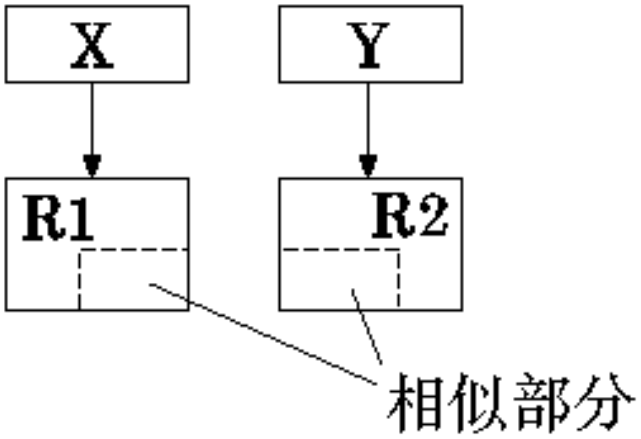
③ 如果需要返回一系列数据给它的调用者，在完成数据加工或结束时，应当给它的调用者返回一个“结束标志”。

4.3.5 软件模块结构的改进方法

(2) 消除重复功能，改善软件结构。

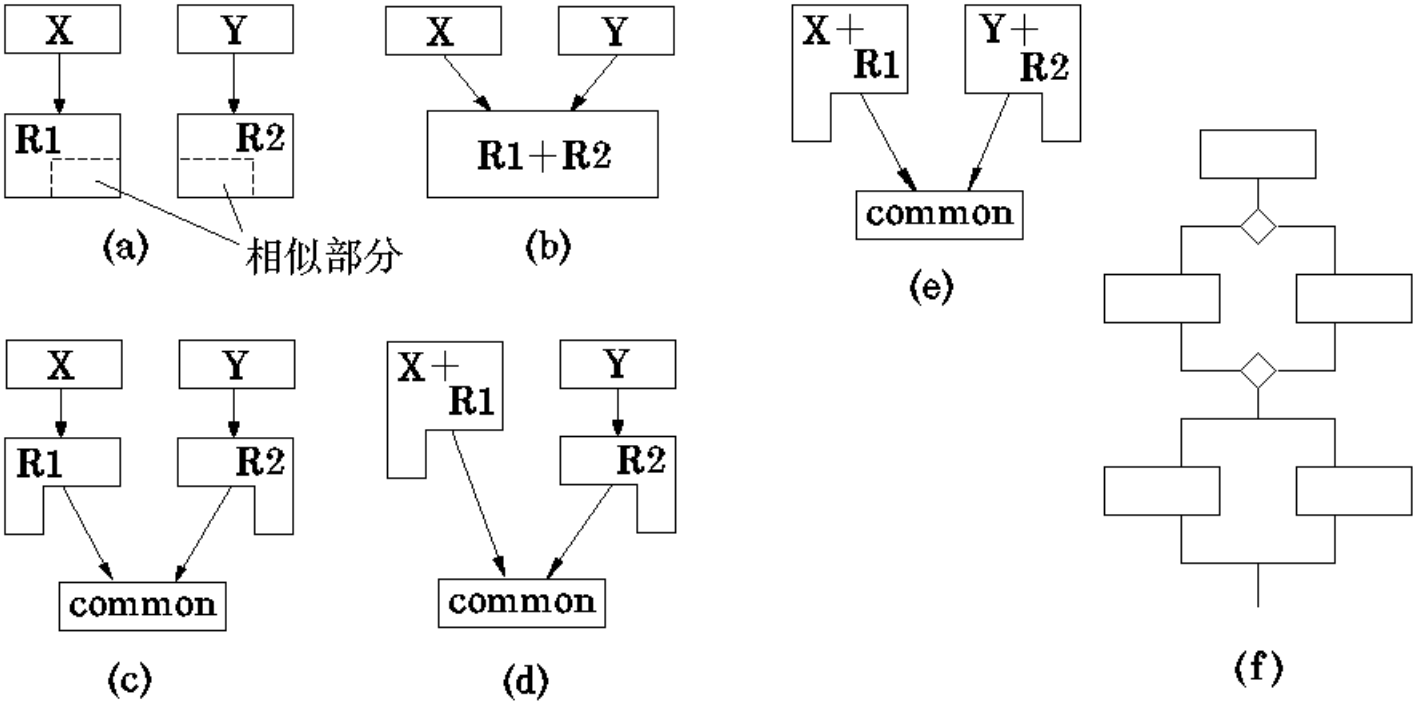
① 完全相似。在结构上完全相似，可能只是在数据类型上不一致。此时可以采取完全合并的方法。

② 局部相似。



4.3.5 软件模块结构的改进方法

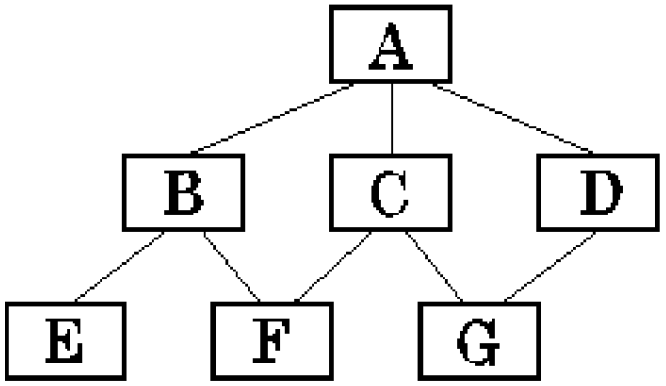
② 局部相似：此时，不可以把两者合并为一，如图(b)所示，因为这样在合并后的模块内部必须设置许多查询开关，如图(f)所示。



4.3.5 软件模块结构的改进方法

(3) 模块的作用范围应在控制范围之内。

- **模块的控制范围包括它本身及其所有的从属模块。**
- **模块的作用范围是指模块内一个判定的作用范围，凡是受这个判定影响的所有模块都属于这个判定的作用范围。**
- **如果一个判定的作用范围包含在这个判定所在模块的控制范围之内，则这种结构是简单的。**

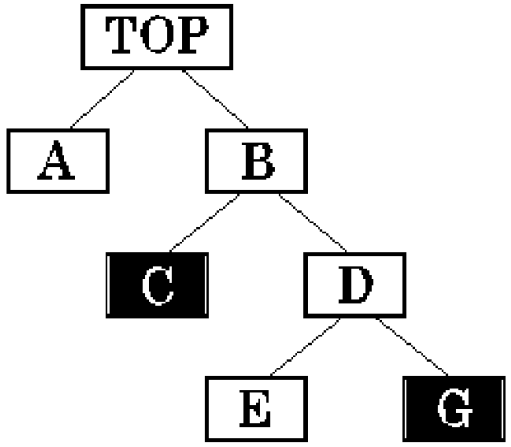


(a)

4.3.5 软件模块结构的改进方法

(3) 模块的作用范围应在控制范围之内。

- 图(b)表明作用范围不在控制范围之内。模块G做出一个判定之后，若需要模块 C工作，则必须把信号回送给模块D，再由D把信号回送给模块B。图中加黑框表示判定的作用范围。

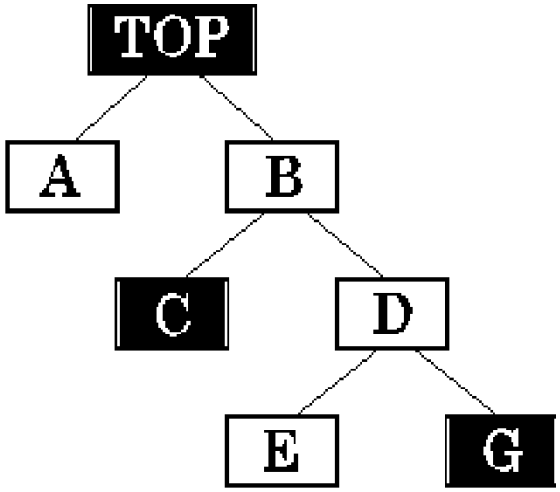


(b)

4.3.5 软件模块结构的改进方法

(3) 模块的作用范围应在控制范围之内。

- **图(c)虽然表明模块的作用范围是在控制范围之内，可是判定所在模块TOP所处层次太高，这样也需要经过不必要的信号传送，增加了数据的传送量。**

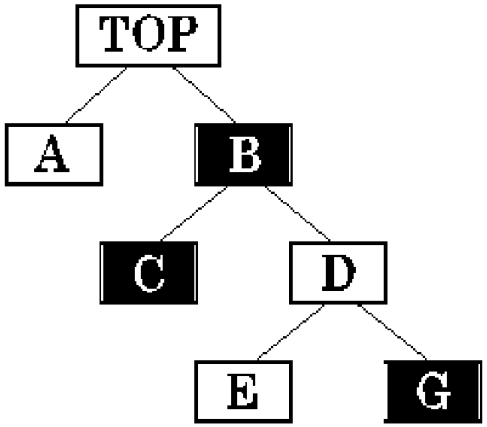


(c)

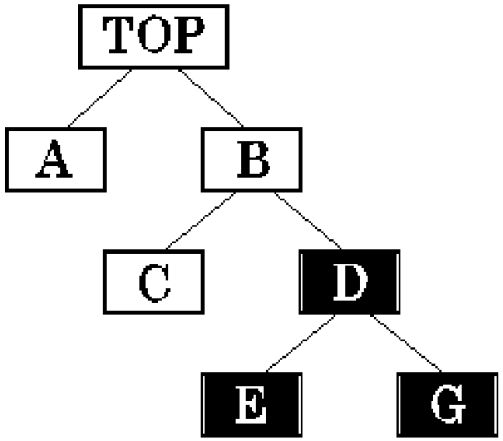
4.3.5 软件模块结构的改进方法

(3) 模块的作用范围应在控制范围之内。

- **图(d) 表明作用范围在控制范围之内，只有一个判定分支有一个不必要的穿越，是一个较好的结构；**
- **图(e)所示为一个比较理想的结构。**



(d)



(e)

4.3.5 软件模块结构的改进方法

(3) 模块的作用范围应在控制范围之内。

- **如果在设计过程中，发现作用范围不在控制范围内，可采用如下办法把作用范围移到控制范围之内。**

① 将判定所在模块合并到父模块中，使判定处于较高层次。

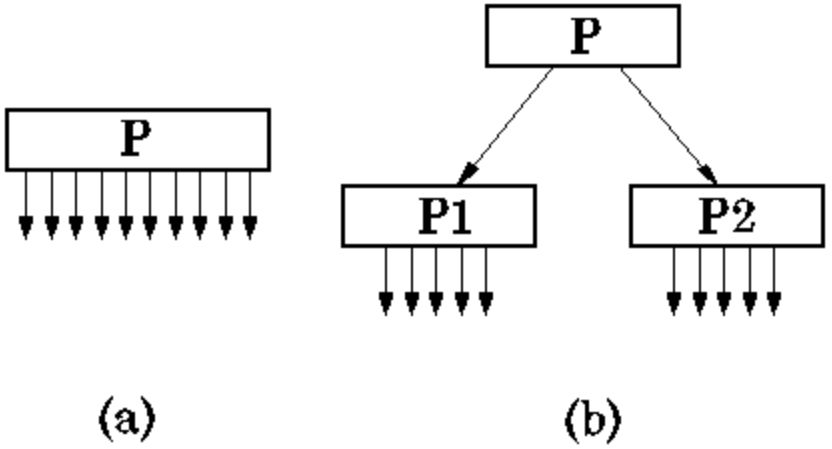
② 将受判定影响的模块下移到控制范围内。

③ 将判定上移到层次中较高的位置。

4.3.5 软件模块结构的改进方法

(4) 尽可能减少高扇出结构，随着深度增大扇入。

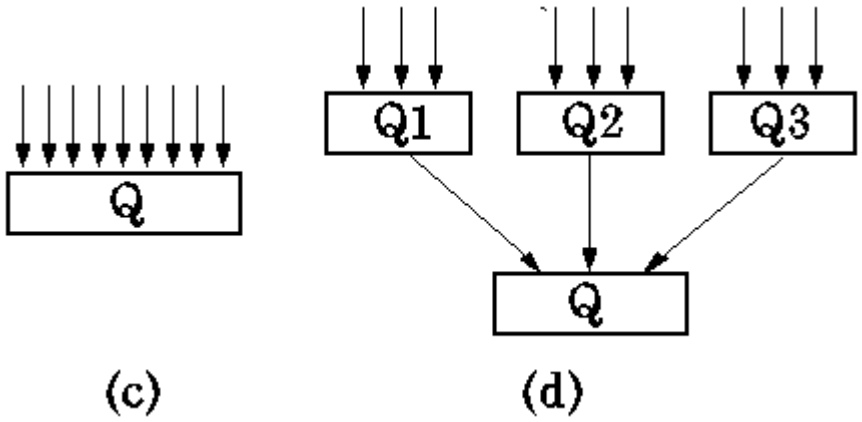
- **模块的扇出数是指模块调用子模块的个数。如果一个模块的扇出数过大，就意味着该模块过分复杂，需要协调和控制过多的下属模块。**
- **出现这种情况是由于缺乏中间层次，所以应当适当增加中间层次的控制模块。如图所示。**



4.3.5 软件模块结构的改进方法

(4) 尽可能减少高扇出结构，随着深度增大扇入。

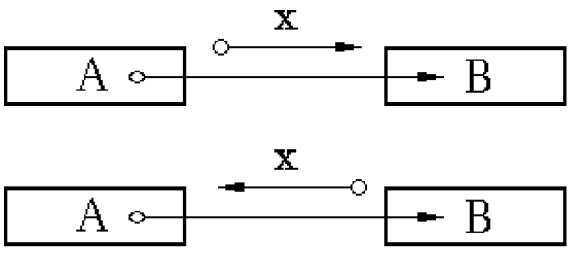
- **一个模块的扇入数越大，则共享该模块的上级模块数目越多。扇入大，是有好处的。**
- **但如果一个模块的扇入数太大，如超过8，而它又不是公用模块，说明该模块可能具有多个功能。**
- **在这种情况下应当对它进一步分析并将其功能分解。**



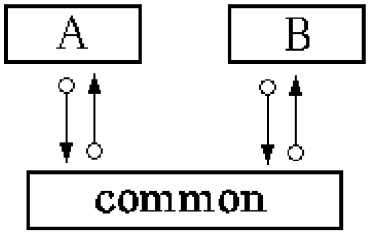
4.3.5 软件模块结构的改进方法

(5) 避免或减少使用病态连接。

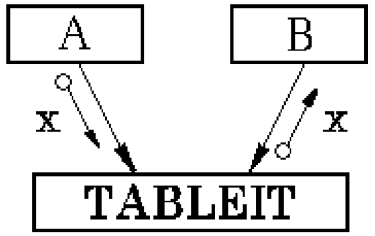
- **应限制使用如下3种病态连接。**



(a) 直接病态连接



(b) 公共数据域病态连接



(c) 通信模块病态连接

4.3.5 软件模块结构的改进方法

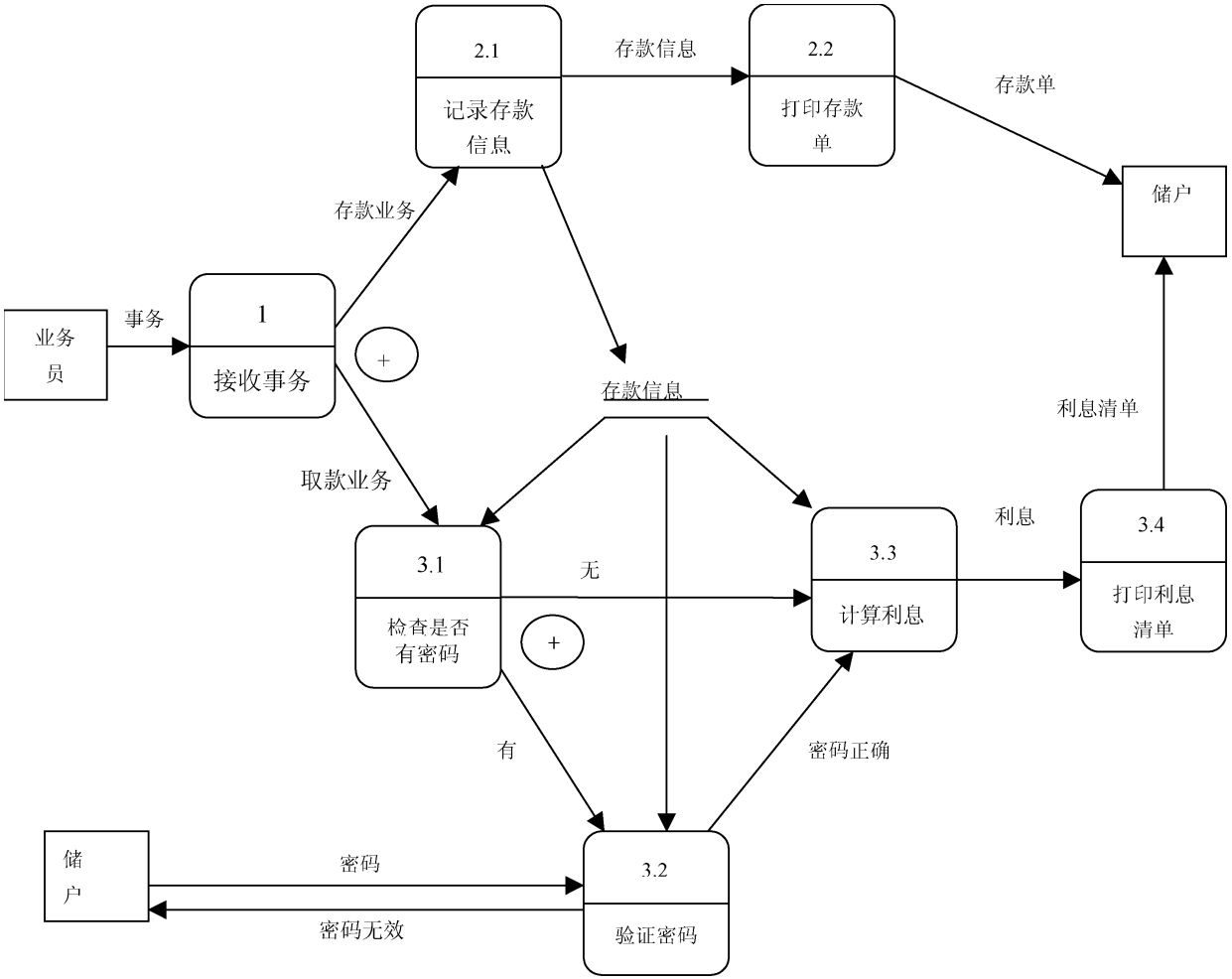
(6) 模块的大小要适中。

- **模块的大小，可以用模块中所含语句的数量的多少来衡量。**
- **通常规定其语句行数为50～100，保持在一页纸之内，最多不超过500行。**

4.3.5 软件模块结构的改进方法

- 实例研究
  - **针对第3章例3.1的银行储蓄系统，开发软件的结构图。**

第1步：对银行储蓄系统的数据流图进行复查并精化，得到如图所示的数据流图。



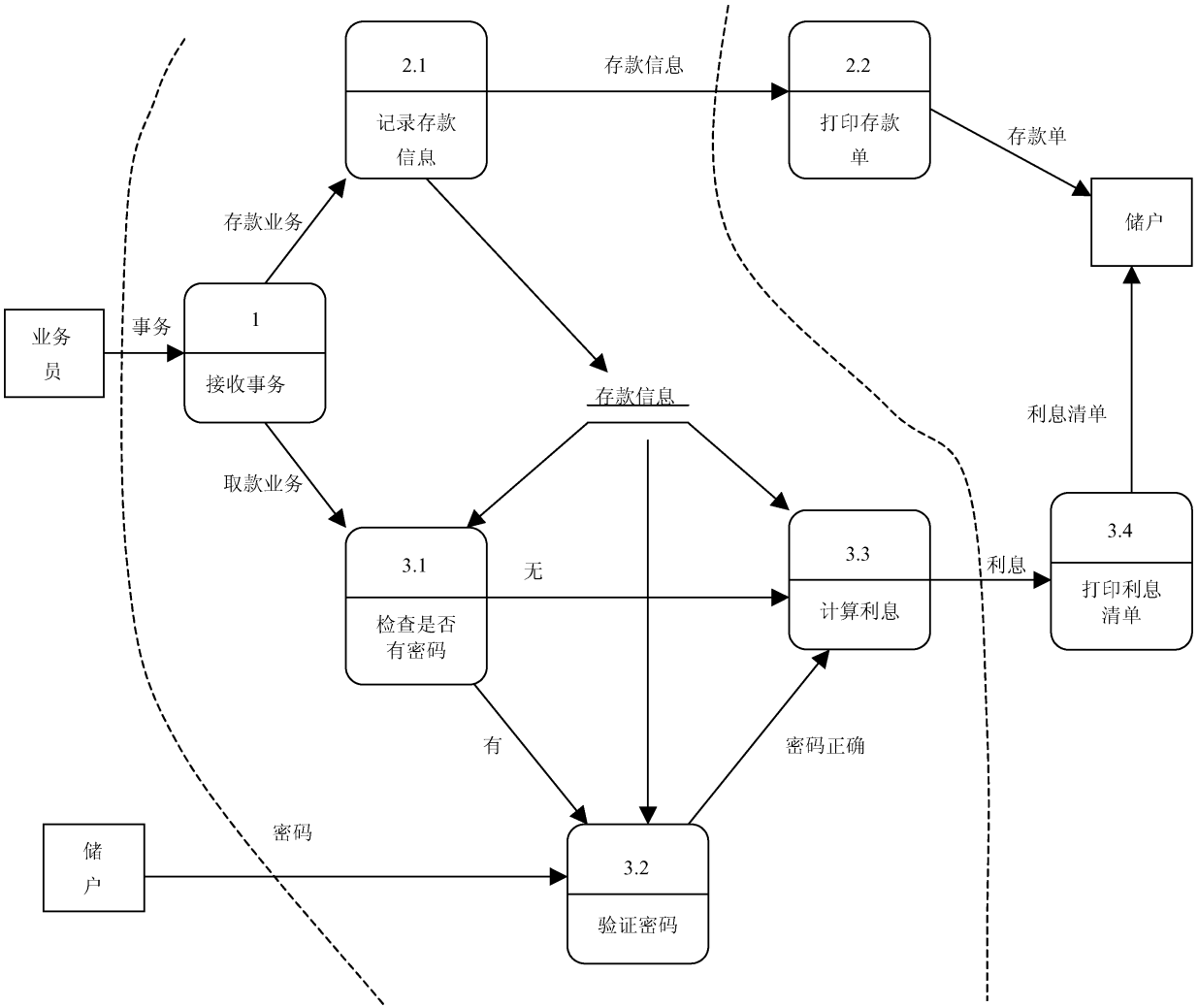
4.3.5 软件模块结构的改进方法

第2步：确定数据流图具有变换特性还是事务特性。

通过对精化后的数据流图进行分析，可以看到整个系统是对存款及取款两种不同的事务进行处理，因此具有事务特性。

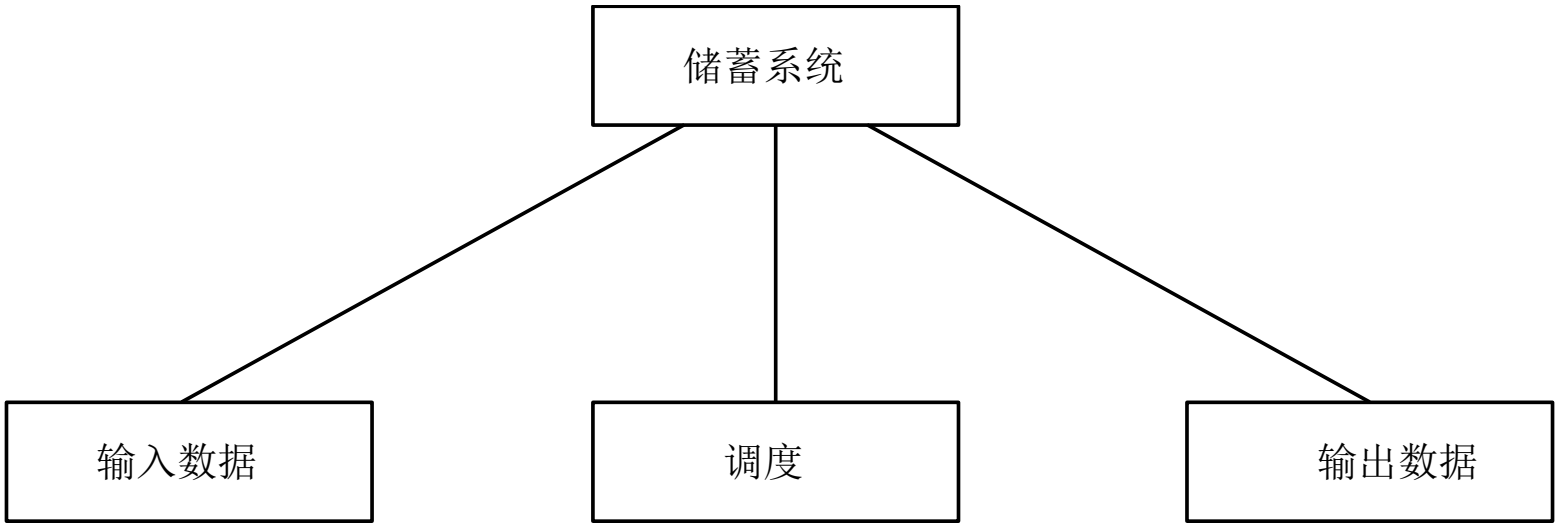
4.3.5 软件模块结构的改进方法

第3步：确定输入流和输出流的边界，如图所示。



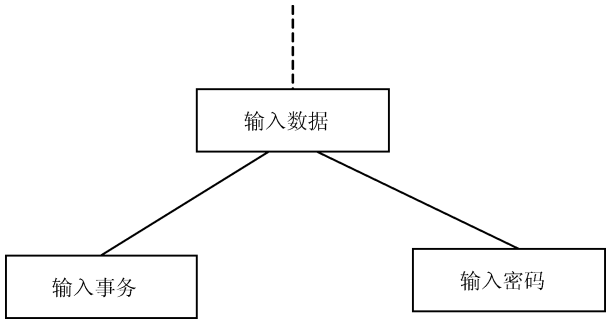
4.3.5 软件模块结构的改进方法

第4步：完成第一级分解。分解后的结构图如图所示。

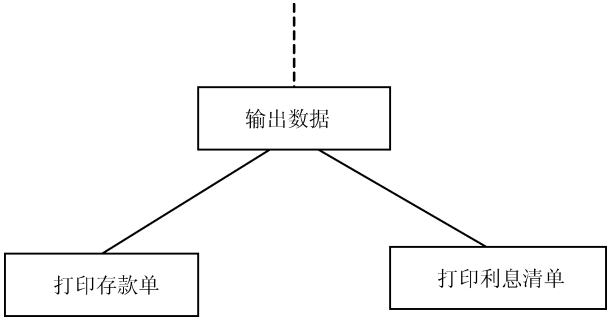


4.3.5 软件模块结构的改进方法

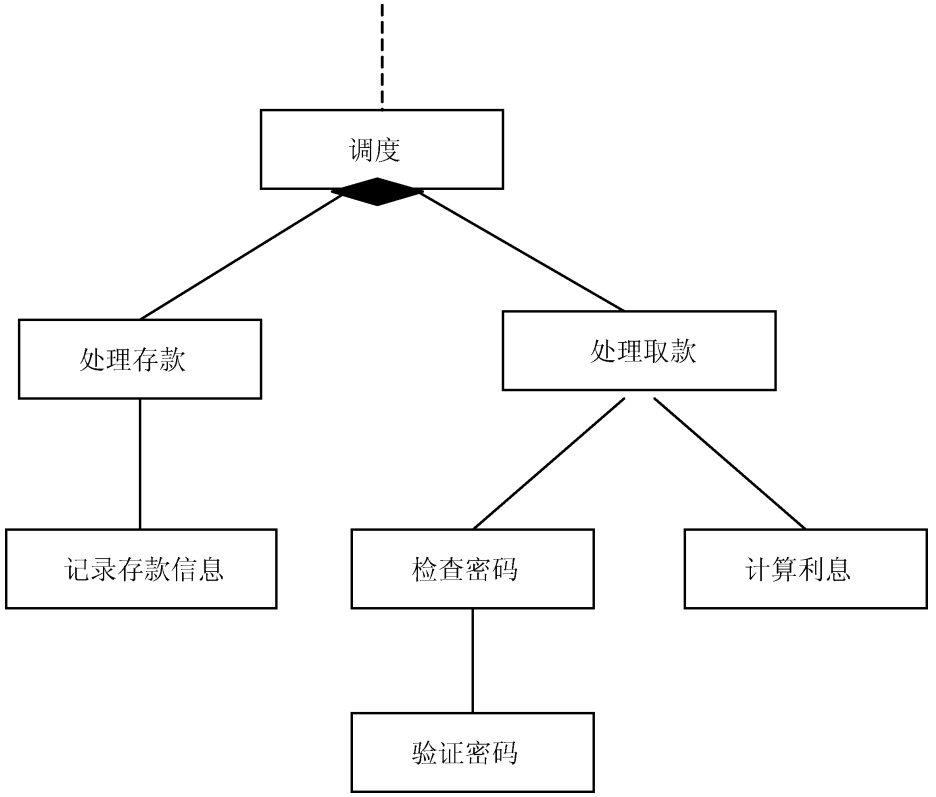
第5步：完成第二级分解。对上图中的“输入数据”、“输出数据”和“调度”模块进行分解，得到未经精化的输入结构、输出结构和事务结构，分别如图(a)、(b)和(c)所示。



(a) 未经精化的输入结构



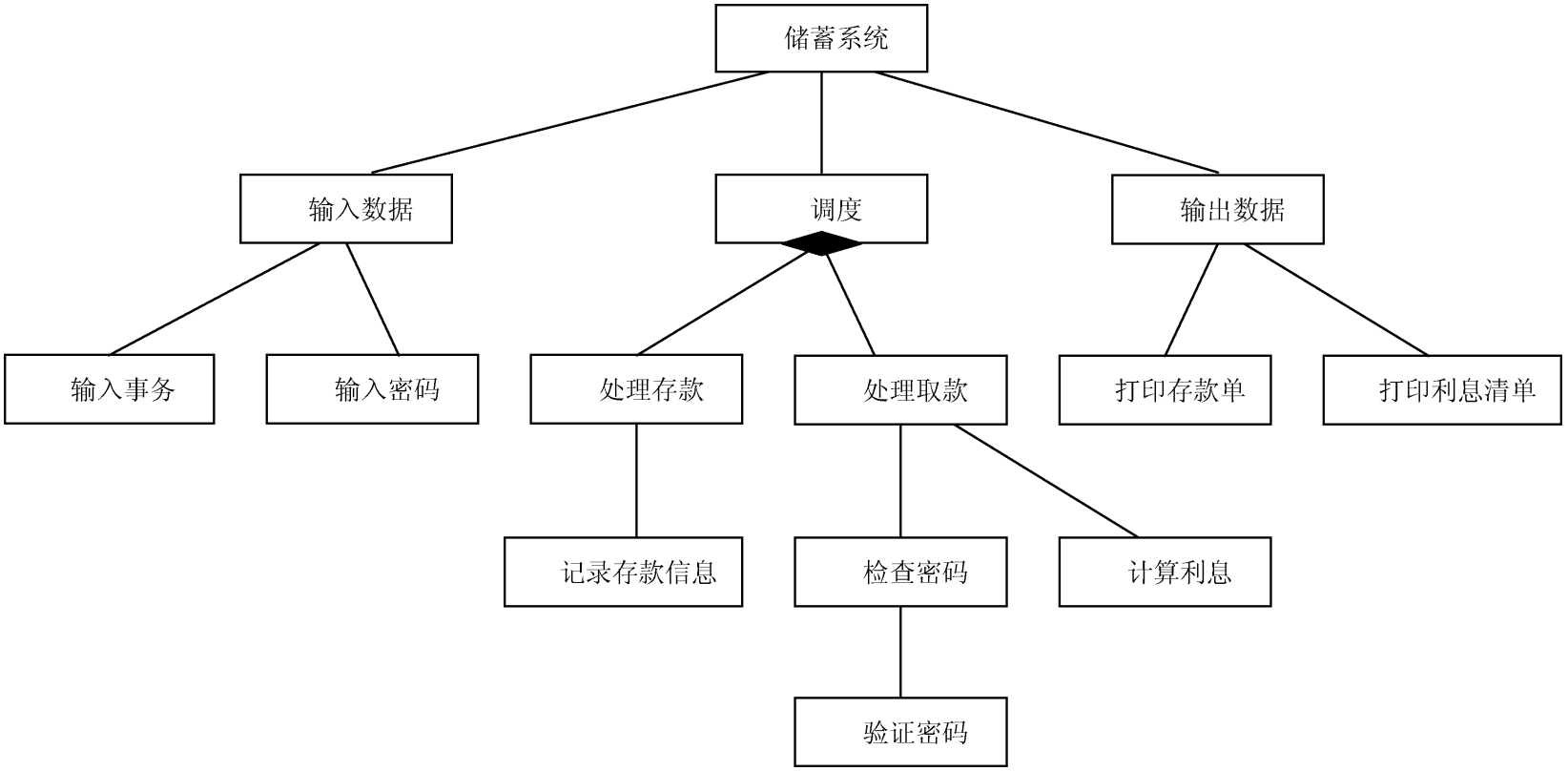
(b) 未经精化的输出结构



(c) 未经精化的事务结构

4.3.5 软件模块结构的改进方法

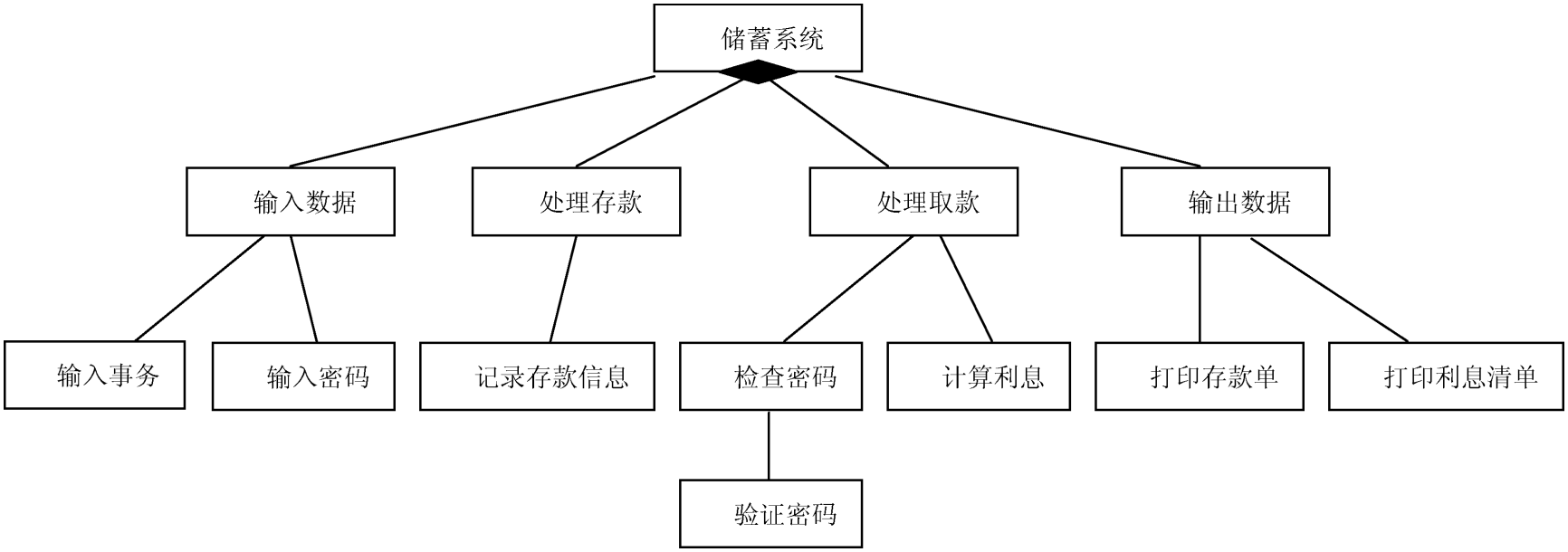
第5步：完成第二级分解。将上面的3部分合在一起，得到初始的软件结构，如图所示。



4.3.5 软件模块结构的改进方法

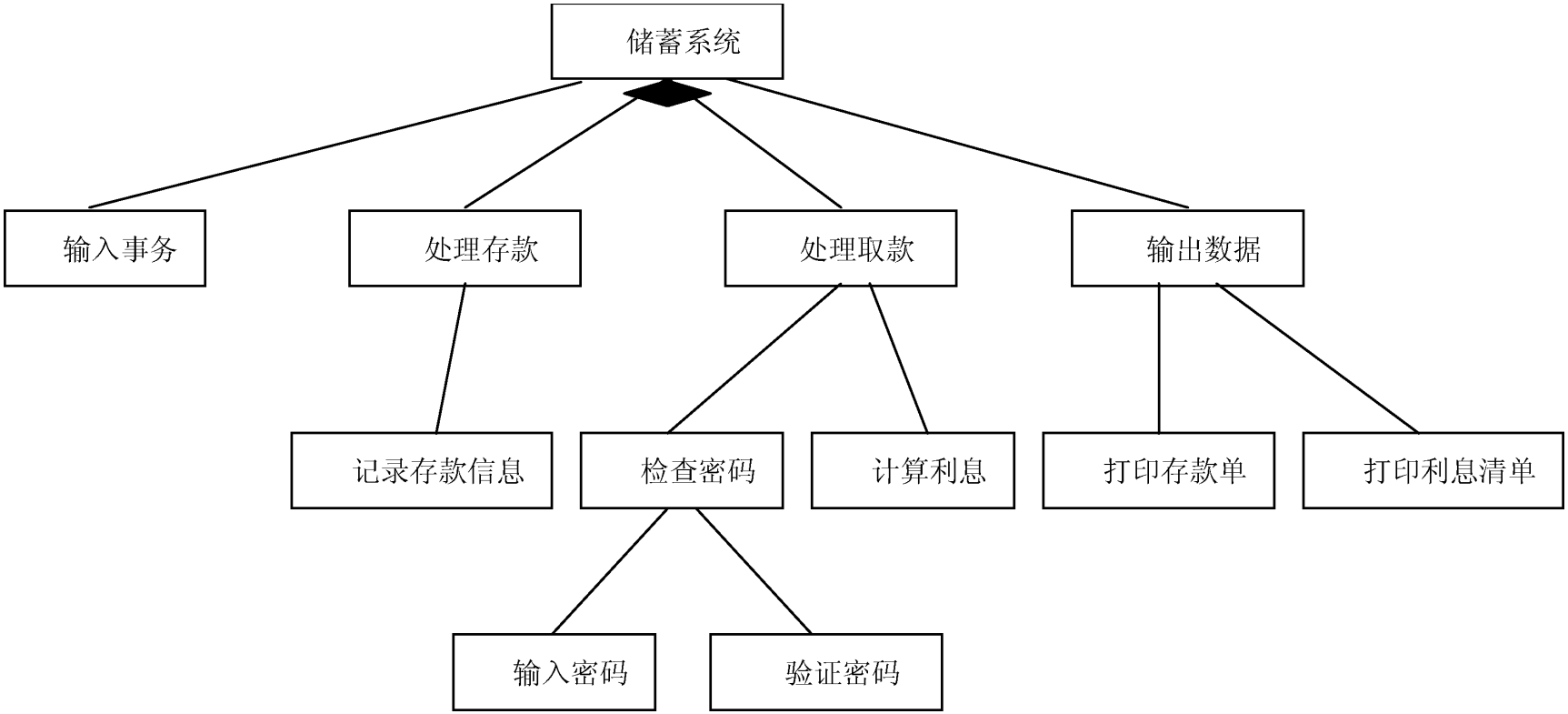
第6步：对软件结构进行精化。

(1) 由于调度模块下只有两种事务，因此，可以将调度模块合并到上级模块中，如图所示。



4.3.5 软件模块结构的改进方法

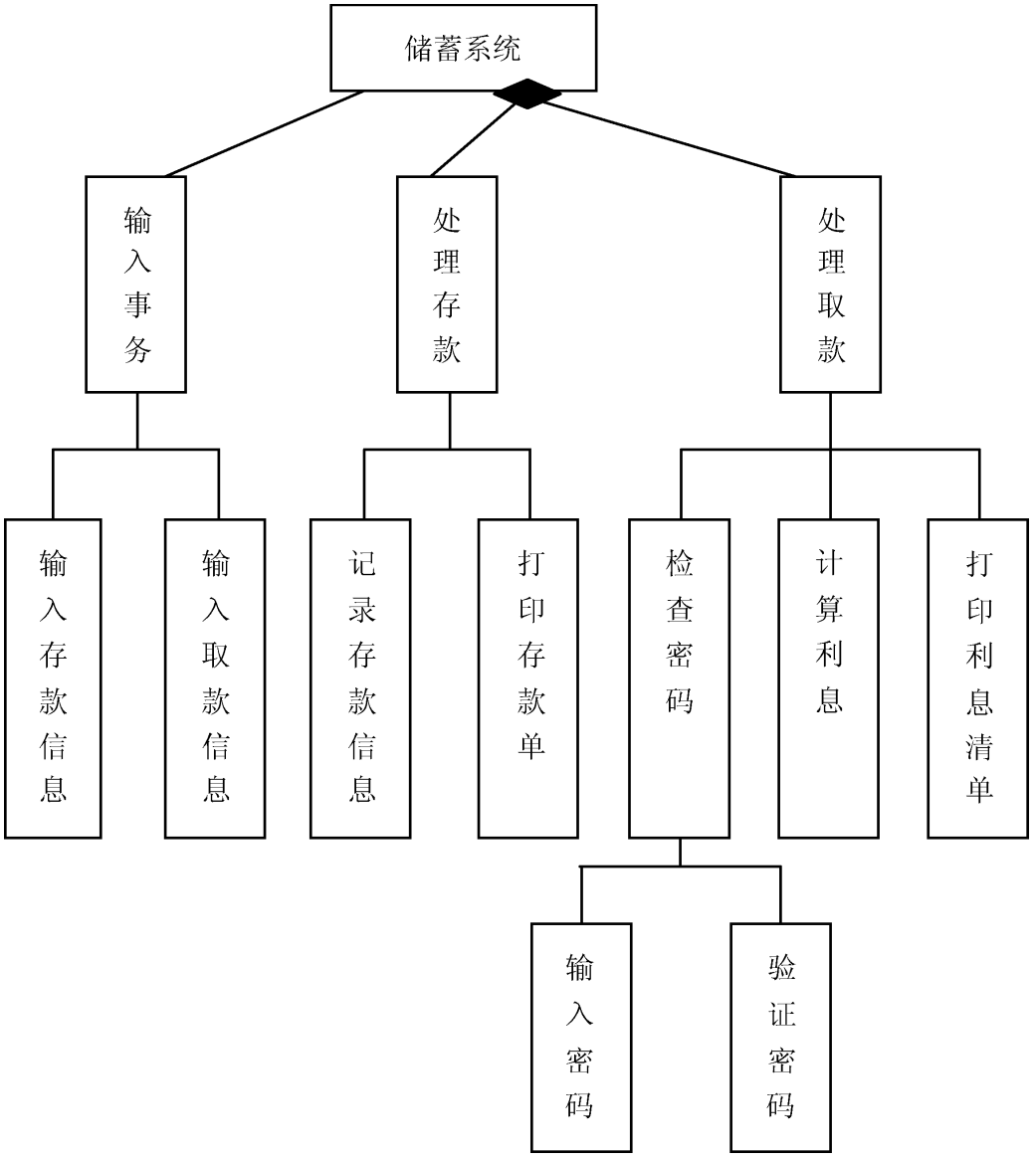
(2) “检查密码”模块的作用范围不在其控制范围之内（即“输入密码”模块不在“检查密码”模块的控制范围之内），需对其进行调整，如图所示。



4.3.5 软件模块结构的改进方法

(3) 提高模块的独立性，并对“输入事务”模块进行细化。

也可以将“检查密码”功能合并到其上级模块中。



4.4 接口设计

- 接口设计概述

- **接口设计的依据是数据流图中的自动化系统边界。**
- **接口设计主要包括3个方面：模块或软件构件间的接口设计；软件与其他软硬件系统之间的接口设计；软件与人（用户）之间的交互设计。**
- **人机交互（用户）界面是人机交互的主要方式**

4.4 接口设计

- 人机交互界面
  - **为了设计好人机交互界面，设计者需要了解用户界面应具有的特性；**
  - **还应该认真研究使用软件的用户，包括用户是什么人？用户怎样学习与新的计算机系统进行交互？用户需要完成哪些工作？等等。**

4.4 接口设计

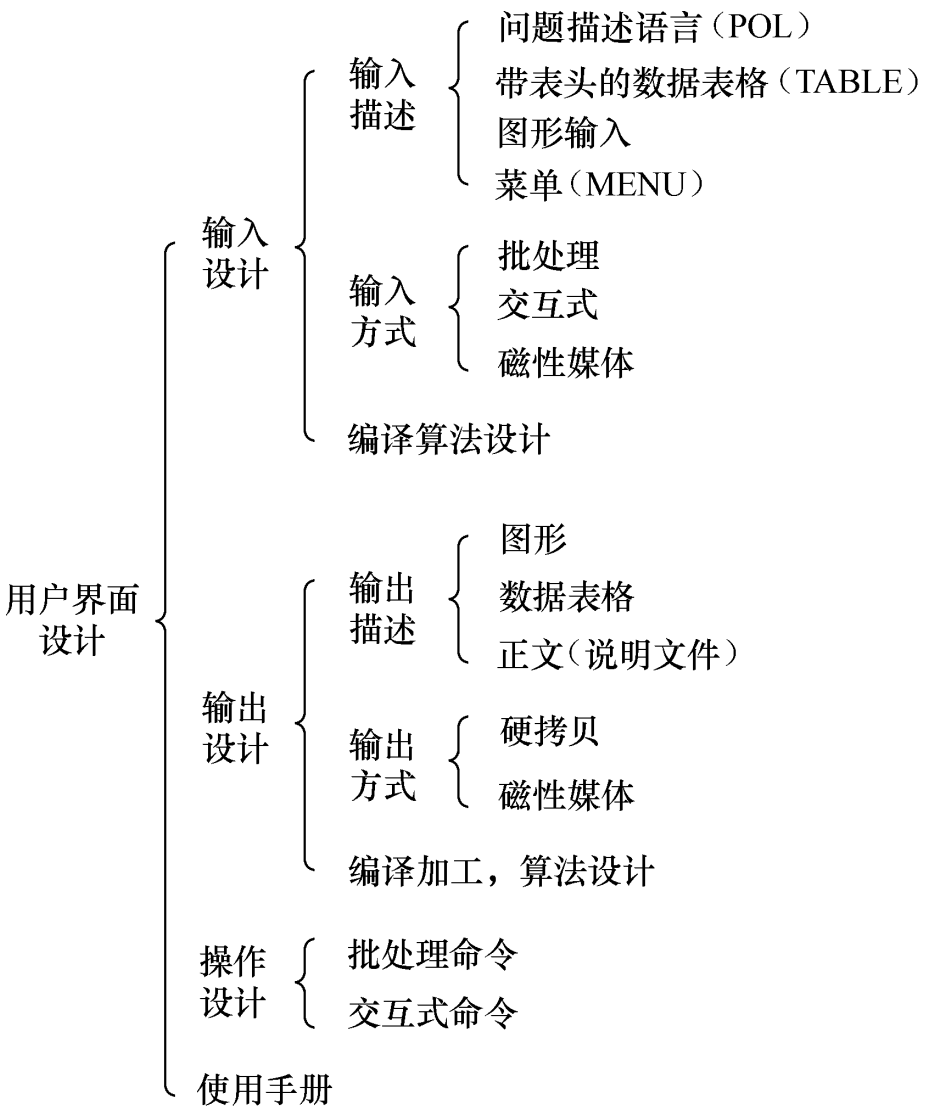
- 用户界面应具备的特性
  - **可使用性**：**包括使用简单、界面一致、拥有HELP帮助功能、快速的系统响应和低的系统成本、具有容错能力等。**
  - **灵活性**：**考虑到用户的特点、能力和知识水平，应当使用户接口满足不同用户的要求。**
  - **可靠性**：**用户界面的可靠性是指无故障使用的间隔时间。用户界面应能保证用户正确、可靠地使用系统，保证有关程序和数据的安全性。**

4.4 接口设计

- 用户类型
  - **外行型**：以前从未使用过计算机系统的用户。
  - **初学型**：尽管对新的系统不熟悉，但对计算机还有一些使用经验的用户。
  - **熟练型**：对一个系统有相当多的经验，能够熟练操作的用户。
  - **专家型**：这一类用户了解系统内部的构造，有关于系统工作机制的专业知识，具有维护和修改基本系统的能力。专家型需要为他们提供能够修改和扩充系统能力的复杂界面。

4.4 接口设计

- 界面设计类型



4.4 接口设计

- 界面设计类型

➢ **在选用界面形式的时候，应当考虑每种类型的优点和限制，可以从以下几个方面来考察：**

(1) 使用的难易程度：对于没有经验的用户，该界面使用的难度有多大。

(2) 学习的难易程度：学习该界面的命令和功能的难度有多大。

(3) 操作速度：在完成一个指定操作时，该界面在操作步骤、击键和反应时间等方面效率有多高。

4.4 接口设计

- 界面设计类型

(4) **复杂程度**：该界面提供了什么功能、□能否用新的方式组合这些功能以增强界面的功能。

(5) **控制**：人机交互时，是由计算机还是由人发起和控制对话。

(6) **开发的难易程度**：该界面设计是否有难度、开发工作量有多大。

4.4 接口设计

- 设计详细的交互

➢ **人机交互的设计有若干准则，包括以下内容：**

(1) 一致性。采用一致的术语、一致的步骤和一致的活动。

(2) 操作步骤少。使击键或点击鼠标的次数减到最少，甚至要减少做某些事所需的下拉菜单的距离。

(3) 不要“哑播放”。

(4) 提供Undo功能。

(5) 减少人脑的记忆负担。不应该要求人从一个窗口中记住某些信息，然后在另一个窗口中使用。

(6) 提高学习效率。为高级特性提供联机帮助，以便用户在需要时容易找到。

4.5 数据设计

- 文件设计

➢ **以下几种情况适合于选择文件存储。**

(1) 数据量较大的非结构化数据，如多媒体信息。

(2) 数据量大，信息松散，如历史记录、档案文件等。

(3) 非关系层次化数据，如系统配置文件。

(4) 对数据的存取速度要求极高的情况。

(5) 临时存放的数据。

4.5 数据设计

- 文件设计

➢ 一般要根据文件的特性，来确定文件的组织方式。

(1) 顺序文件：这类文件分两种，一种是连续文件，另一种是串联文件。

(2) 直接存取文件：可根据记录关键字的值，通过计算直接得到记录的存放地址。

(3) 索引顺序文件：其基本数据记录按顺序文件组织，记录排列顺序必须按关键字值升序或降序安排，且具有索引部分，索引部分也按同一关键字进行索引。

4.5 数据设计

- 文件设计

(4) **分区文件**：这类文件主要用于存放程序。它由若干称为成员的顺序组织的记录组和索引组成。

每一个成员就是一个程序，由于各个程序的长度不同，所以各个成员的大小也不同，需要利用索引给出各个成员的程序名、开始存放位置和长度。

(5) **虚拟存储文件**：这是基于操作系统的请求页式存储管理功能而建立的索引顺序文件。

4.5 数据设计

- 数据库设计

- 根据数据库的组织，可以将数据库分为网状数据库、层次数据库、关系数据库、面向对象数据库、文档数据库、多维数据库等。
- 关系数据库最成熟，应用也最广泛，一般情况下，大多数设计者都会选择关系数据库。
- 在结构化设计方法中，很容易将结构化分析阶段建立的实体—关系模型映射到关系数据库中。

4.5 数据设计

- **数据对象实体的映射**
  - **一个数据对象（实体）可以映射为一个表或多个表，当分解为多个表时，可以采用横切和竖切的方法。**
  - **竖切常用于实例较少而属性很多的对象。通常将经常使用的属性放在主表中，而将其他一些次要的属性放到其他表中。**
  - **横切常常用于记录与时间相关的对象。往往在主表中只记录最近的对象，而将以前的记录转到对应的历史表中。**

4.5 数据设计

- **关系的映射**
  - **一对一关系的映射**：可以在两个表中都引入外键，进行双向导航。也可以将两个数据对象组合成一张单独的表。
  - **一对多关系的映射**：可以将关联中的“一”端毫无变化地映射到一张表，将关联中表示“多”的端上的数据对象映射到带有外键的另一张表，使外键满足关系引用的完整性。
  - **多对多关系的映射**：为了表示多对多关系，关系模型必须引入一个关联表，将两个数据实体之间的多对多关系转换成两个一对多关系。

4.6 过程设计

- 概要设计的任务完成后，就进入详细设计阶段，也就是过程设计阶段。
- 在这个阶段，要决定各个模块的实现算法，并使用过程描述工具精确地描述这些算法。

4.6 过程设计

- 表达过程规格说明的工具称为过程描述工具，可以将过程描述工具分为以下3类。

(1) 图形工具：把过程的细节用图形方式描述出来，如程序流程图、N-S图、PAD图、决策树等。

(2) 表格工具：用一张表来表达过程的细节。这张表列出了各种可能的操作及其相应的条件，即描述了输入、处理和输出信息，如决策表。

(3) 语言工具：用某种类高级语言（称为伪代码）来描述过程的细节，如很多数据结构教材中使用类Pascal、类C语言来描述算法。

4.6.1 结构化程序设计

- 由于软件开发和维护中存在的一系列严重问题，于20世纪60年代爆发了软件危机。
- 为了克服软件危机，引发了程序设计的一场革命，诞生了结构化程序设计方法。

4.6.1 结构化程序设计

- 结构化程序设计的概念与原则
  - **最早由E. W. Dijkstra提出;建议从高级语言中取消GOTO语句；**
  - **1966年，Bohm和Jacopini证明：只用三种基本的控制结构“顺序”、“选择”和“循环”就能实现任何单入口和单出口的没有“死循环”的程序。**

4.6.1 结构化程序设计

- 结构程序设计的概念

  - **如果一个程序的代码块仅仅通过顺序、选择和循环这三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。**

4.6.1 结构化程序设计

- 结构程序设计的主要原则

(1)使用语言中的顺序、选择、重复等有限的基本控制结构表示程序逻辑。

(2)选用的控制结构只准许有一个入口和一个出口。

(3)程序语句组成容易识别的块（Block），每块只有一个入口和一个出口。

(4)复杂结构应该用基本控制结构进行组合嵌套来实现。

(5)语言中没有的控制结构，可用一段等价的程序段模拟，但要求该程序段在整个系统中应前后一致。

4.6.1 结构化程序设计

- 结构程序设计的主要原则

(6) 严格控制GOTO语句，仅在下列情形才可使用：

- **用非结构化的程序设计语言去实现结构化的构造。**
- **若不使用GOTO语句就会使程序功能模糊。**
- **在某种可以改善而不是损害程序可读性的情况下。例如，在查找结束时，文件访问结束时，出现错误情况要从循环中转出时，使用布尔变量和条件结构来实现就不如用GOTO语句来得简洁易懂。**

4.6.1 结构化程序设计

- 结构程序设计的主要原则

(7) 在程序设计过程中，尽量采用自顶向下(Top - Down)、逐步细化(Stepwise Refinement)的原则，由粗到细，一步步展开。

4.6.2 程序流程图

- 程序流程图也称为程序框图，是软件开发者最熟悉的算法表达工具。
- 早期的流程图也存在一些缺点。特别是表示程序控制流程的箭头，使用的灵活性极大，程序员可以不受任何约束，随意转移控制，这将不符合结构化程序设计的思想。
- 为使用流程图描述结构化程序，必须对流程图加以限制。

4.6.2 程序流程图

- 程序流程图的基本控制结构

(1) **顺序型**：几个连续的加工步骤依次排列构成。

(2) **选择型**：由某个逻辑判断式的取值决定选择两个加工中的一个。

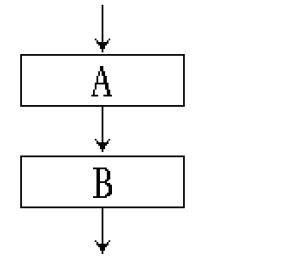
(3) **先判定（while）型循环**：在循环控制条件成立时，重复执行特定的加工。

(4) **后判定（until）型循环**：重复执行某些特定的加工，直至控制条件成立。

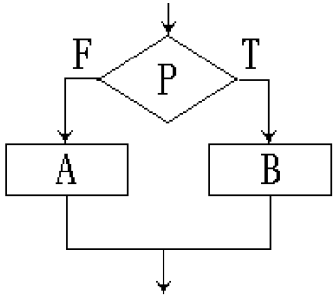
(5) **多情况（case）型选择**：列举多种加工情况，根据控制变量的取值，选择执行其一。

4.6.2 程序流程图

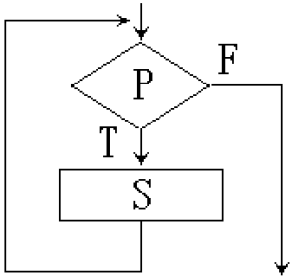
- 程序流程图的基本控制结构



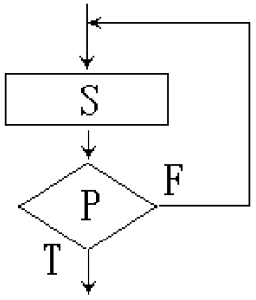
(a) 顺序型



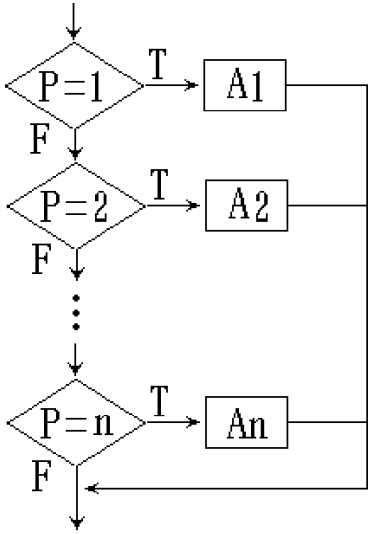
(b) 选择型



(c) 先判定型循环
(DO-WHILE)



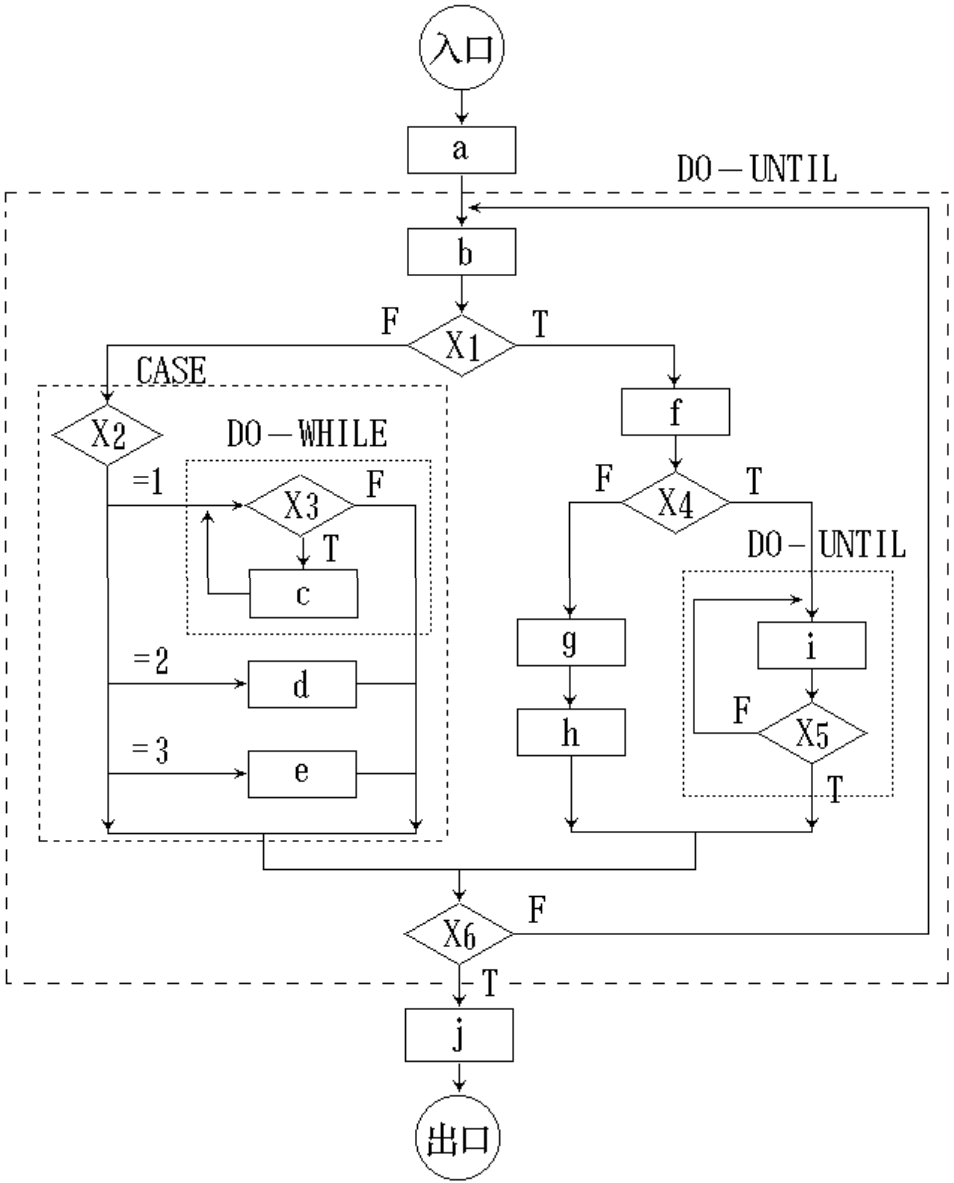
(d) 后判定型循环
(DO-UNTIL)



(e) 多情况选择型
(CASE型)

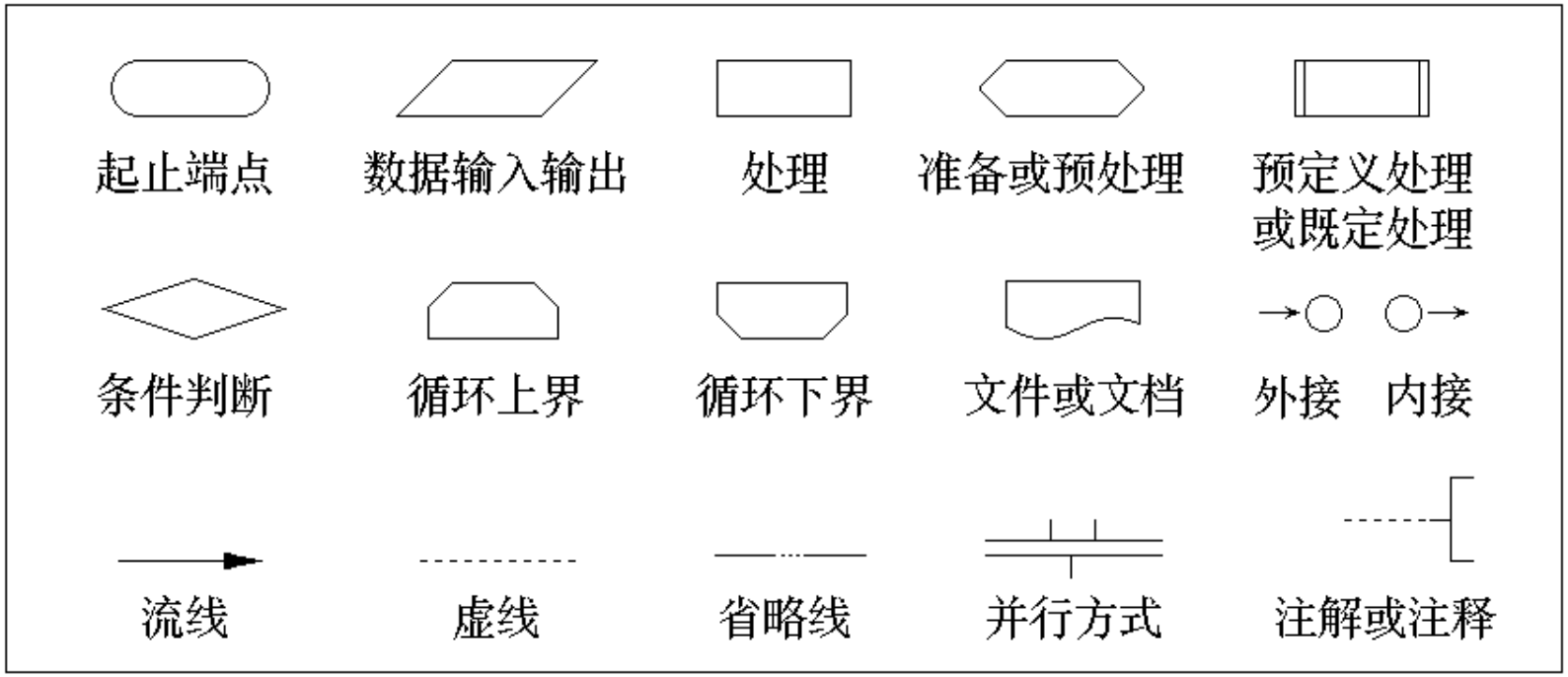
4.6.2 程序流程图

- 程序流程图实例



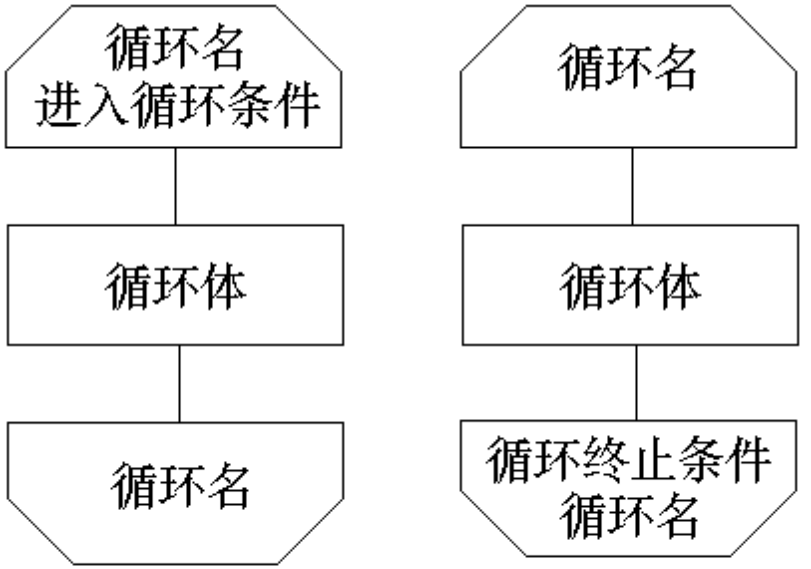
4.6.2 程序流程图

- 程序流程图的标准符号（国家标准）



4.6.2 程序流程图

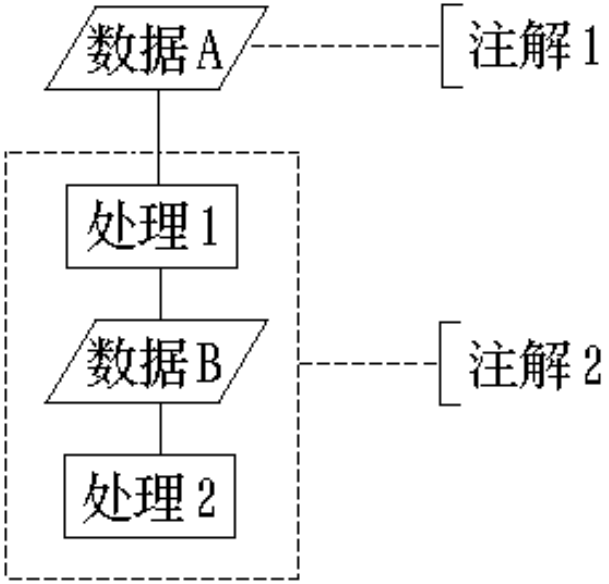
- 循环的标准符号



(a) while型循环　(b) until型循环

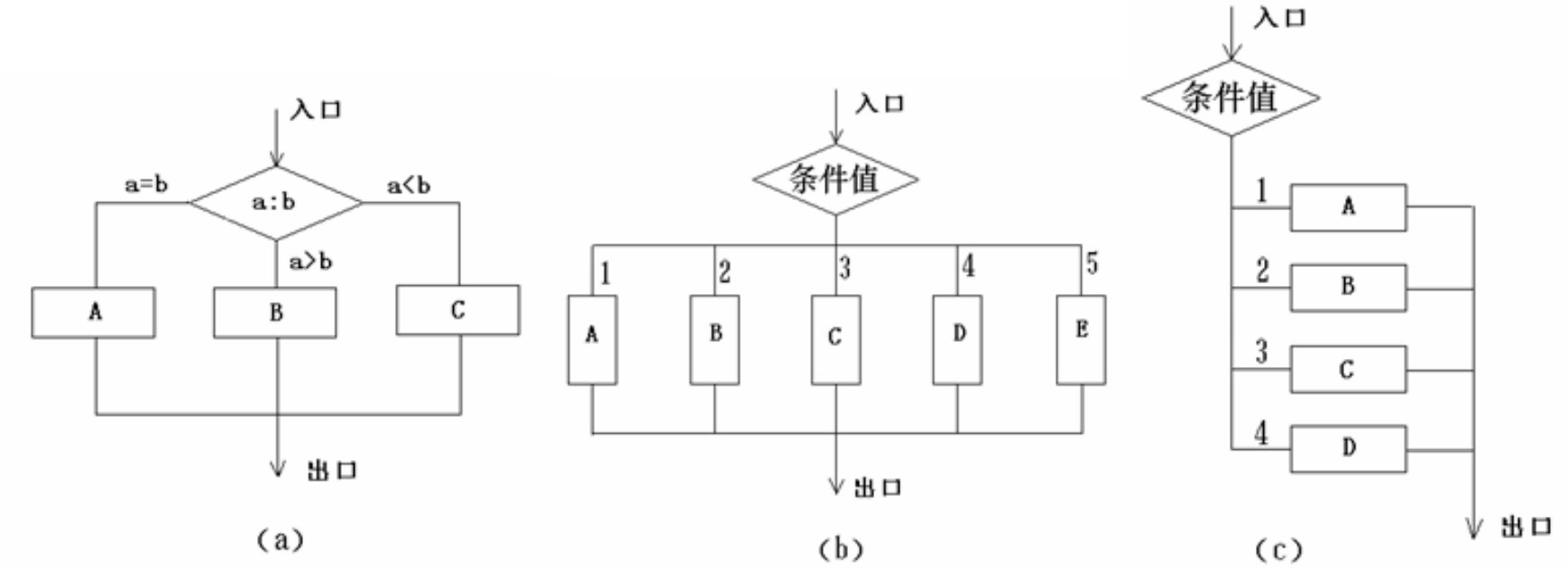
4.6.2 程序流程图

- 注解符的使用



4.6.2 程序流程图

- 多选择判断

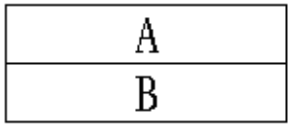


4.6.3 N-S图

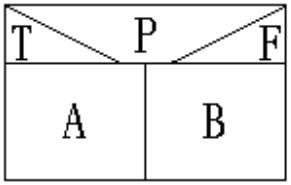
- **Nassi和Shneiderman 提出了一种符合结构化程序设计原则的图形描述工具，叫做盒图（box-diagram），也叫做N-S图。**
- **在N-S图中，为了表示5种基本控制结构，规定了5种图形构件。**

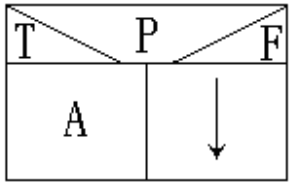
4.6.3 N-S图

- **N-S图的基本控制结构**

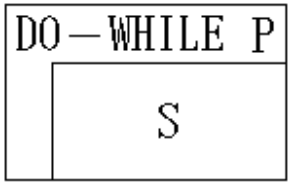


(a) 顺序型

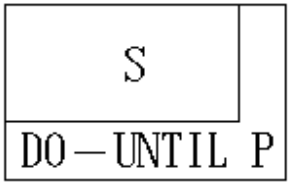




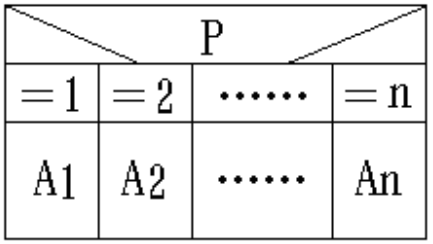
(b) 选择型



(c) WHILE重复型



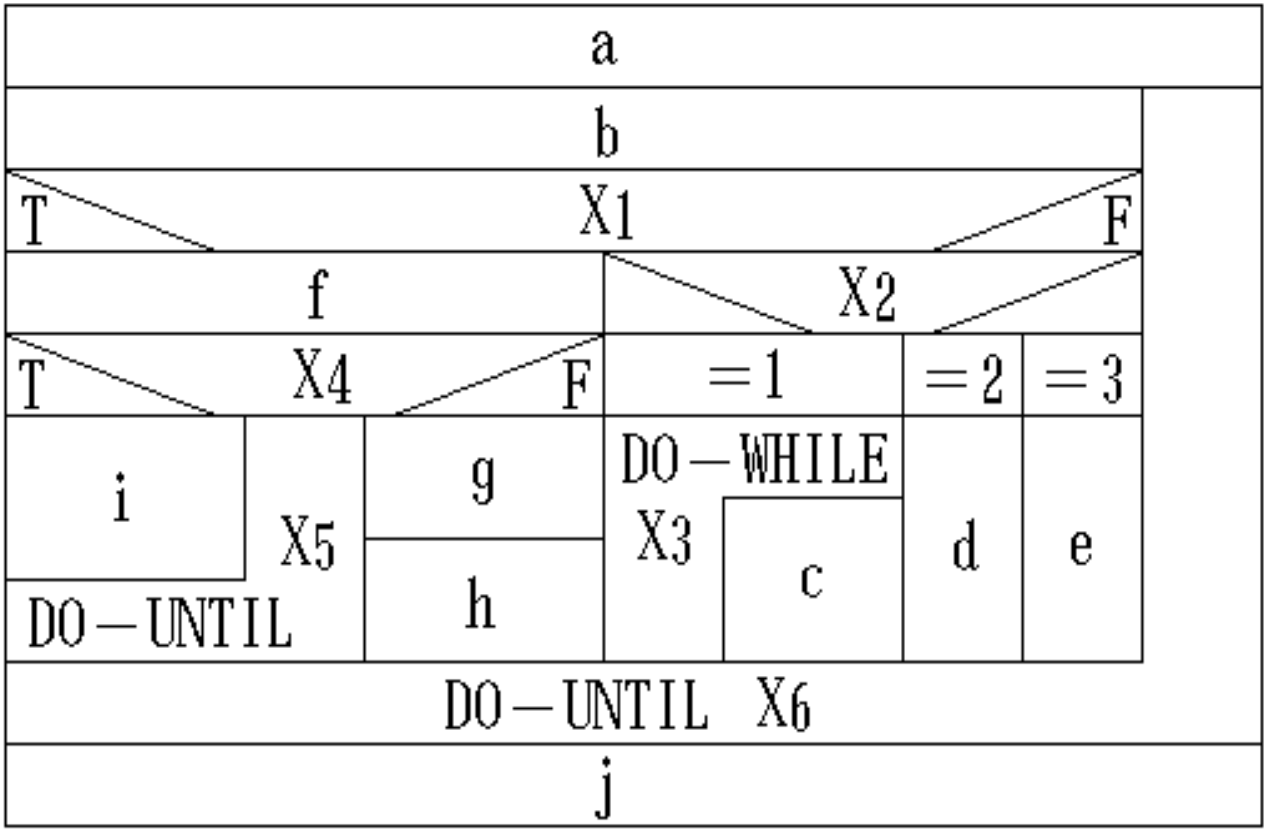
(d) UNTIL重复型



(e) 多分支选择型
(CASE型)

4.6.3 N-S图

- **N-S图的实例**



4.6.3 N-S图

- **N-S图的特点**

(1) 图中每个矩形框（除CASE构造中表示条件取值的矩形框外）都是明确定义了的功能域（即一个特定控制结构的作用域），以图形表示，清晰可见。

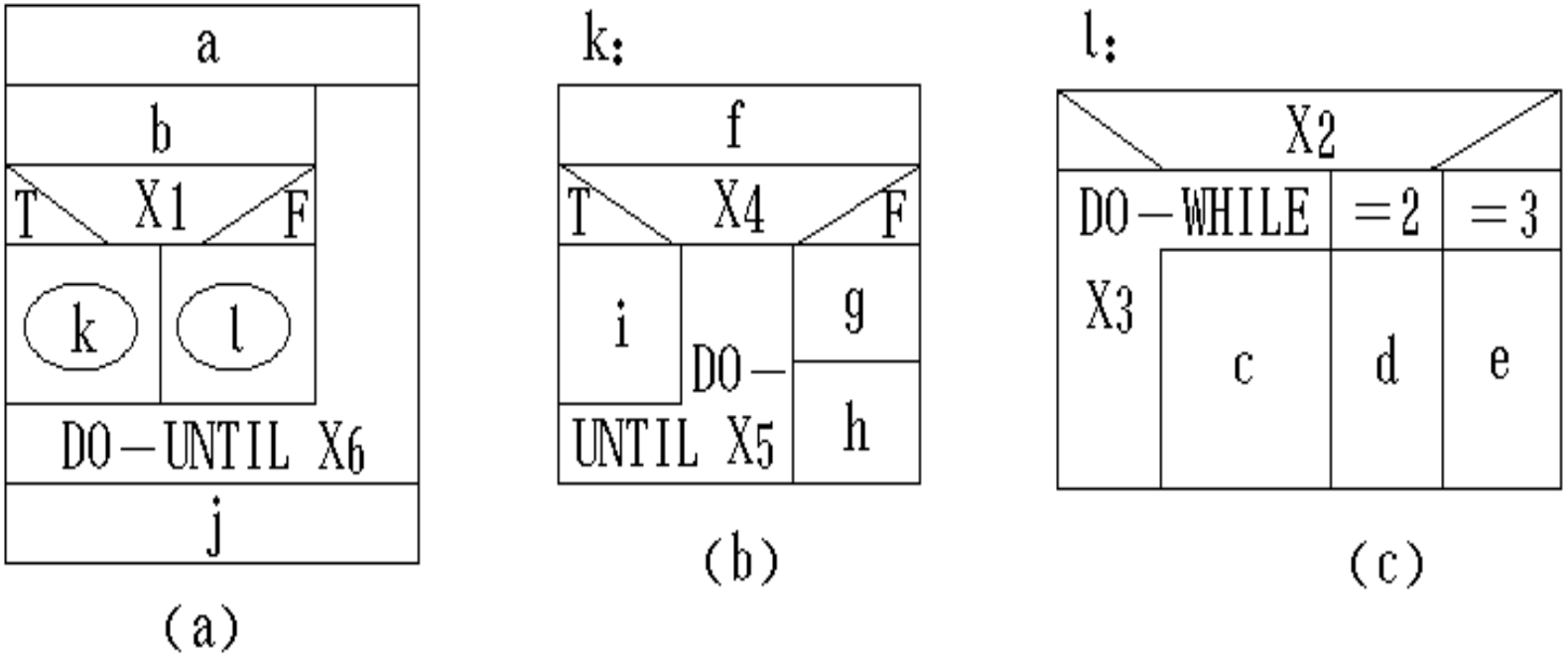
(2) 它的控制转移不能任意规定，必须遵守结构化程序设计的要求。

(3) 很容易确定局部数据和（或）全局数据的作用域。

(4) 很容易表现嵌套关系，也可以表示模块的层次结构。

4.6.3 N-S图

- **N-S图的扩展表示**

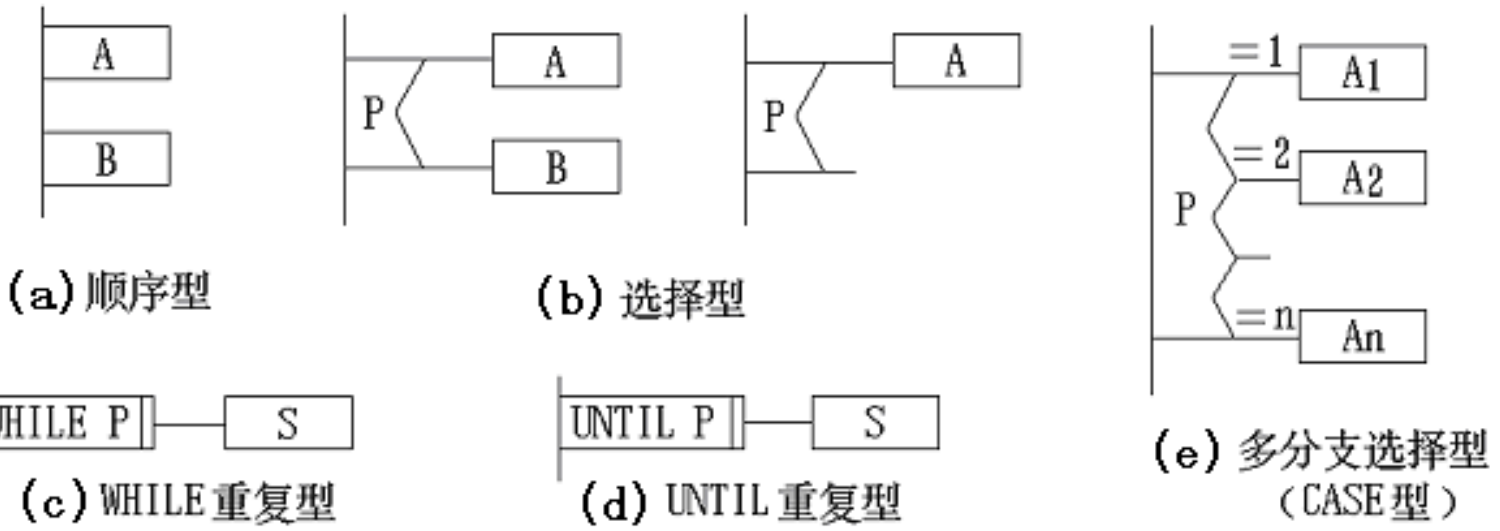


4.6.4 PAD图

- PAD（problem analysis diagram）是日本日立公司提出，由程序流程图演化来的，用结构化程序设计思想表现程序逻辑结构的图形工具。
- PAD也设置了5种基本控制结构的图式，并允许递归使用。

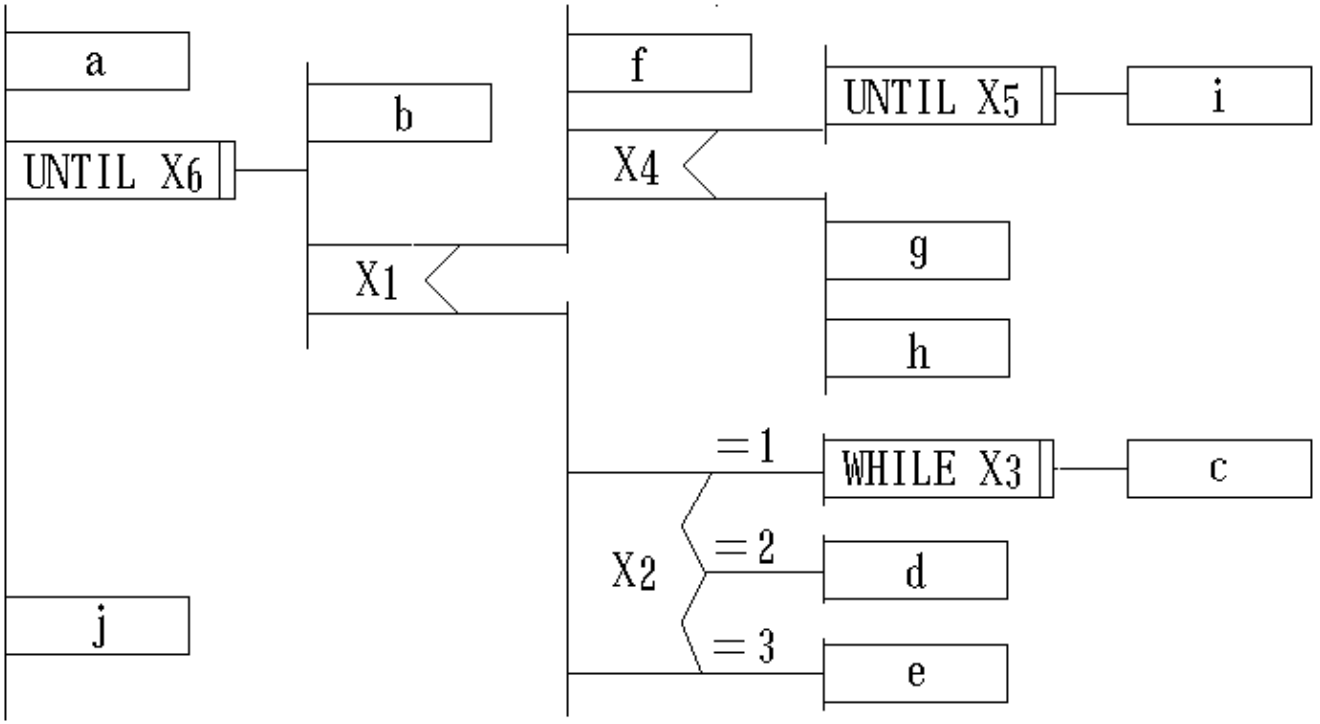
4.6.4 PAD图

- **PAD图的基本控制结构**



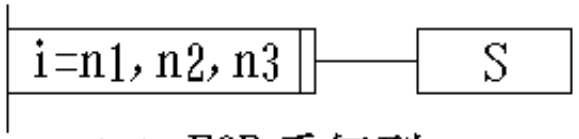
4.6.4 PAD图

- **PAD图的实例**



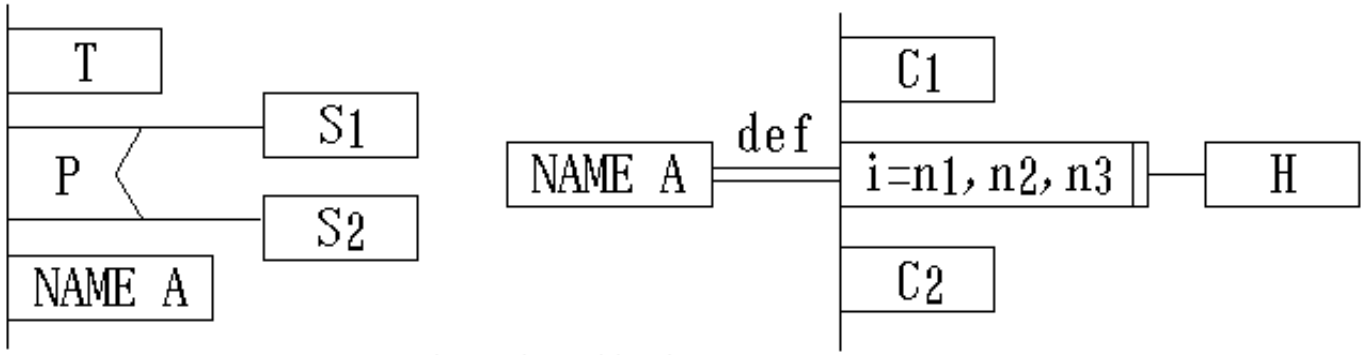
4.6.4 PAD图

- **PAD的扩充控制结构**



其中，i=n1,n2,n3 表示
i:=n1 to n2 step n3

（a）FOR重复型



（b）def格式

4.6.4 PAD图

- PAD的优点
  - 使用PAD符号所设计出来的程序必然是结构化程序。
  - PAD图描绘程序结构清晰，图中竖线的总条数就是程序的层次数。
  - 用PAD图表现程序逻辑易读、易懂、易记。
  - 容易将PAD图自动转换为高级语言源程序。
  - PAD图既可以表示程序逻辑，也可用于描绘数据结构。
  - PAD图的符号支持自顶向下、逐步求精方法的使用。

4.6.5 伪代码

- 伪代码是一种介于自然语言和形式化语言之间的半形式化语言，是一种用于描述功能模块的算法设计和加工细节的语言，也称为**程序设计语言**（Program Design Language，PDL）。
- 伪码的语法规则分为“外语法”和“内语法”。
- **外语法**应当符合一般程序设计语言常用语句的语法规则；
- **内语法**可以用英语中一些简单的句子、短语和通用的数学符号来描述程序应执行的功能。

4.6.5 伪代码

- 伪代码的基本控制结构
  - **简单陈述句结构**：**避免复合语句。**
  - **判定结构**：**IF_THEN_ELSE或CASE_OF结构。**
  - **重复结构**：**WHILE_DO或REPEAT_UNTIL结构。**

4.6.5 伪代码

- 伪代码实例：“检查订货单”例子

```
IF 客户订货金额超过 5000 元 THEN
    IF 客户拖延未还赊欠钱款超过 60 天 THEN
        在偿还欠款前不予批准
    ELSE （拖延未还赊欠钱款不超过 60 天）
        发批准书，发货单
    ENDIF
ELSE （客户订货金额未超过 5000 元）
    IF 客户拖延未还赊欠钱款超过 60 天 THEN
        发批准书、发货单，并发催款通知书
    ELSE （拖延未还赊欠钱款不超过 60 天）
        发批准书，发货单
    ENDIF
ENDIF
```

4.6.5 伪代码

- 伪代码的特点

(1) 有固定的关键字外语法，提供全部结构化控制结构、数据说明和模块特征。外语法的关键字是有限的词汇集，它们能对伪代码正文进行结构分割，使之变得易于理解。

(2) 内语法使用自然语言来描述处理特性，为开发者提供方便，提高可读性。

(3) 有数据说明机制，包括简单的（如标量和数组）与复杂的（如链表和层次结构）的数据结构。

(4) 有子程序定义与调用机制，用以表达各种方式的接口说明。

4.6.6 自顶向下、逐步细化的设计过程

主要包括两个方面：

- **一是将复杂问题的解法分解和细化成由若干个模块组成的层次结构；**
- **二是将每个模块的功能逐步分解细化为一系列的处理。**

4.6.6 自顶向下、逐步细化的设计过程

- **在处理较大的复杂任务时，常采取“模块化”的方法，即在程序设计时不是将全部内容都放在同一个模块中，而是分成若干个模块，每个模块实现一个功能。**
- **模块分解完成后，下一步的任务就是将每个模块的功能逐步分解细化为一系列的处理。**

4.6.6 自顶向下、逐步细化的设计过程

- **在概要设计阶段，我们已经采用自顶向下、逐步细化的方法，把复杂问题的解法分解和细化成了由许多功能模块组成的层次结构的软件系统。**
- **在详细设计和编码阶段，我们还应当采取自顶向下、逐步求精的方法，把模块的功能逐步分解，细化为一系列具体的步骤，进而翻译成一系列用某种程序设计语言写成的程序。**

4.6.6 自顶向下、逐步细化的设计过程

- 自顶向下、逐步细化方法举例

➢ **用筛选法求100以内的素数。**

所谓的筛选法，就是从2到100中去掉2,3,5,7的倍数，剩下的就是100以内的素数。

4.6.6 自顶向下、逐步细化的设计过程

- 首先按程序功能写出一个框架

```
main（ ）{
  建立2到100的数组A[ ]，其中A[i]＝i；- - - - - 1
  建立2到10的素数表B[ ]，存放2到10以内的素数；- - 2
  若A[i]＝i是B[ ]中任一数的倍数，则剔除A[i]； - - 3
  输出A[ ]中所有没有被剔除的数； - - - - - - - 4
 }
```

4.6.6 自顶向下、逐步细化的设计过程

- 上述框架中每一个加工语句都可进一步细化成一个循环语句

```
main （ ） {
  /*建立2到100的数组A[ ]，其中A[i]＝i*/   - - 1
  for （i = 2 ; i <= 100 ; i++ ) A[i] = i ;
  /* 建立2到10的素数表B[ ]，存放2到10以内的素数*/ - 2
  B[1] =2 ;  B[2] = 3 ;  B[3] = 5 ;  B[4] = 7 ;
  /*若A[i]＝i是B[ ]中任一数的倍数，则剔除A[i]*/ - - 3
  for （j = 1 ; j <= 4 ; j++ )
    检查A[ ]所有的数能否被B[j]整除并将能被整除的数从A[]中剔除；
  /*输出A[ ]中所有没有被剔除的数*/  - - - - - - - 4
  for （i = 2 ; i <= 100 ; i++ )
    若A[i]没有被剔除，则输出之
}
```

4.6.6 自顶向下、逐步细化的设计过程

- 自顶向下、逐步求精的方法的优点

(1) 自顶向下、逐步求精方法符合人们解决复杂问题的普遍规律。可提高软件开发的成功率和生产率。

(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的过程开发出来的程序具有清晰的层次结构，因此程序容易阅读和理解。

(3) 程序自顶向下、逐步细化，分解成树形结构。 在同一层的结点上做细化工作，相互之间没有关系，因此它们之间的细化工作相互独立。在任何一步发生错误，一般只影响它下层的结点，同一层的其他结点不受影响。

4.6.6 自顶向下、逐步细化的设计过程

- 自顶向下、逐步求精的方法的优点

(4) 程序清晰和模块化，使得在修改和重新设计一个软件时，可复用的代码量最大。

(5) 程序的逻辑结构清晰，有利于程序正确性证明。

(6) 每一步工作仅在上层结点的基础上做不多的设计扩展，便于检查。

(7) 有利于设计的分工和组织工作。

4.7 软件设计规格说明

- **国家标准GB/T 8567—2006《计算机软件文档编制规范》中有关软件总体设计的文档是《系统/子系统设计（结构设计）说明（SSDD）》，描述了系统或子系统的系统级或子系统级设计与体系结构设计。**

That's All!