软件开发技术

第5章 结构化软件开发方法

浙江大学 机械工程学院 航空制造工程研究所

王 青 李江雄

本章内容

浙江大学 ZHEJIANG UNIVERSITY

1 结构化设计的任务和内容

1.1 结构化设计的任务和内容

1.2 概要设计基本概念

1.3 详细设计基本概念

2 程序结构设计

2.1 两个基本概念：模块和程序结构

2.2 模块化设计

2.3 程序结构图

2.4 从数据流图导出程序结构图

2.5 程序结构图的评价与改进

2.6 模块说明

3 结构化程序设计

3.1 结构化程序设计简介

3.2 程序过程

3.3 程序过程的描述方法

01 结构化设计的任务和内容

01 结构化设计的任务和内容

浙江大学 ZHEJIANG UNIVERSITY

软件需求分析阶段完成了对问题的分析建模工作，明确了软件要“做什么”，并用数据流图和实体联系图进行了表示。

◆软件设计阶段的任务

软件设计阶段要根据需求分析阶段的数据流图对软件“**怎么做**”进行建模工作，即对软件解决问题的方案和具体方法进行建模，并使用程序结构图和程序流程图等模型描述程序的执行过程。

◆结构化设计的工作内容

●对于解决问题的方案

对于程序结构设计—得到程序结构图

对于数据库结构设计—得到关系数据模式

结果为：概要设计说明书和数据库结构说明书

●对于具体的解决方法设计

对程序的详细设计过程进行设计

-------—得到程序流程图\问题分析图\盒图等

结果为：详细设计说明书

若用结构化方法，该步称为结构化设计
(Structured Design， SD)

●概要设计

概要设计又称总体设计，它的基本任务是:

- ●将系统划分成模块
- ●决定每个模块的功能
- ●决定模块的调用关系
- ●决定模块的界面.即模块间传递的数据

概要设计所包含的文档包括:

- ●数据结构的描述
- ●模块说明
- ●模块结构图
- ●每个模块的功能说明—用数据词典方式描述

概要设计的主要工作是完成模块分解，确定系统的模块的层次关系，即确定系统的解决方案。

●详细设计

详细设计确定每个模块的内部特征，即每个模块内部的执行过程(怎么做)。

每个模块的详细内部执行过程包括:

- 局部数据组织
- 控制流
- 每一步的具体加工要求及种种实现细节

通过概要设计和详细设计为编程制定了一个周密的计划。

02 程序结构设计

浙江大学 ZHEJIANG UNIVERSITY

2.1两个基本概念

(1)模块

工程上许多大系统都是由一些较小的单元或组件等组成，如建筑工程中的构件、机械设备中的各种成品部件等。这样做的优点是便于加工制造，便于维修，由于一些部件可以公用，成本也较节省.同样，一个几万行的程序系统也不应该是铁板一块，它应由许多较小的单元组成，这种单元可以称为模块。

模块(Module)通常是指用一个名字可以调用的一段程序语句，可以理解为子程序、函数、过程等。

一个模块有如下四个参数组成:

- 输入和输出：分别是模块需要的和产生的信息
- 功能：指模块所做的工作
- 内部数据：仅供该模块本身引用的数据
- 程序代码

程序结构设计即完成程序的模块化设计，获取系统由多少个模块组成，以及每个模块的具体参数。

(2)程序结构

程序结构或程序物理结构是对要解决的问题或要设计的系统的一种分层的表示方法，它指出了组成程序（系统）的各个元素（即各个模块）以及它们之间的关系。

- 程序结构是从需求分析阶段定义的分析模型中导出；
- 程序结构隐含着控制层次的关系， 但不表示程序的具体算法或过程关系，即不表示诸如处理的顺序，选择的出现和次序、操作的重复循环等。
- 程序结构通常用程序结构图的形式表示。

2.2 模块化设计

程序结构设计需要完成系统的模块划分和设计，即完成系统的模块组成及每个模块的相关参数，其设计原则为：

- 模块完成独立的功能
- 符合信息隐蔽和信息局部化原则
- 模块间关连和依赖程度尽量小

其中耦合性和内聚性是描述模块设计中模块内部及模块之间的联系的两个定性指标，Myers量化了它们的概念：

- 内聚性：一个模块内部的交互程度；
- 耦合性：模块之间的交互程度。

我们希望设计出的软件是强内聚低耦合的！

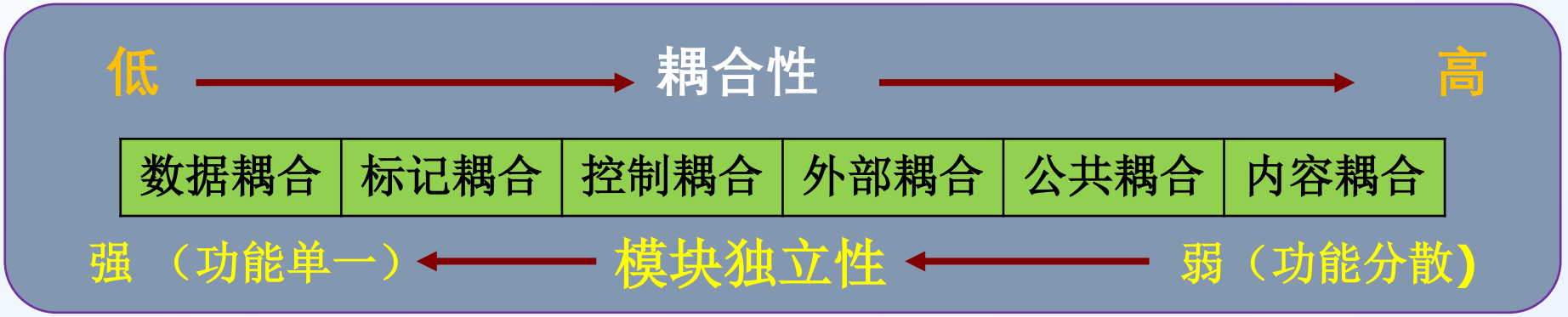
好的程序结构表现：

块间联系小（耦合性弱）　块内联系大（内聚性强）

2.2.1 模块的耦合性

耦合：(coupling)表示软件结构内不同模块之间的相互依赖（连接）的紧密程度；---块间联系

- 耦合的强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。
- 耦合性越高，模块独立性越弱，因而追求松散耦合避免强耦合。
- 两个模块之间的耦合有6种形式，按照耦合程度由低到高排列为：数据耦合、标记耦合、控制耦合、外部耦合、公共耦合、内容耦合。



模块耦合可以分为6种类型，耦合程度从低到高排列为：

耦合类型	特点	备注
数据耦合	模块之间通过参数交换数据信息	必须的
标记耦合	两个模块接口的参数包含相同的内部结构	
控制耦合	模块之间传递的参数含有控制信息	
外部耦合	模块受到程序外部环境的约束，如受通讯协议、设备等。应控制在少数模块上。	必须的，但应尽量少
公共耦合	模块共同使用全局数据	要尽量控制
内容耦合	一个模块直接修改另一个模块的内容	最好不使用

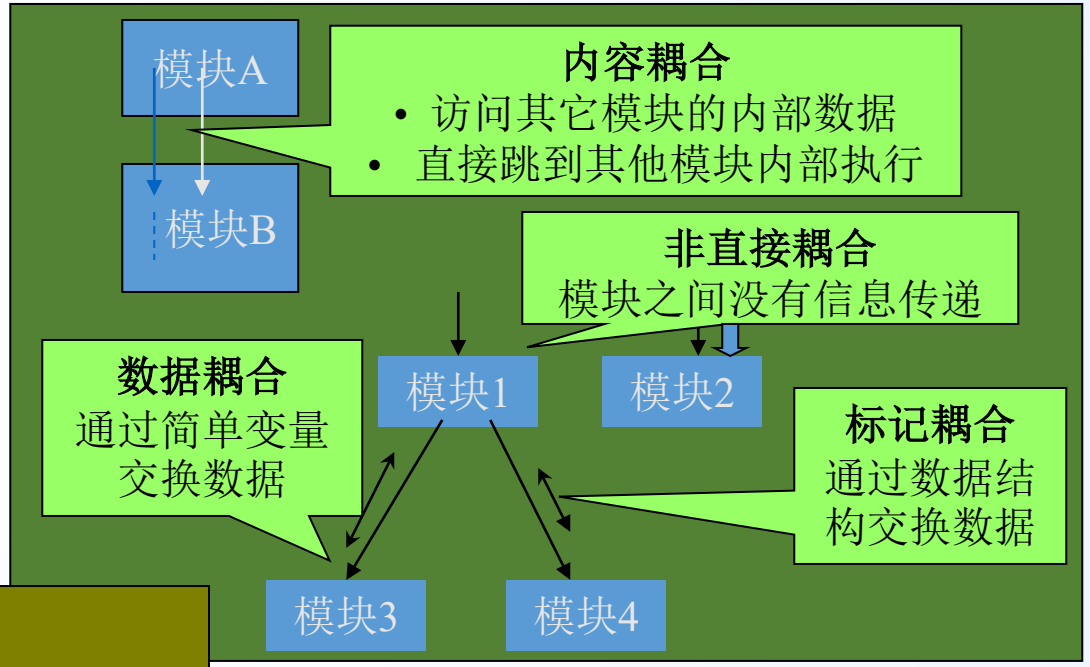
1. 数据耦合
2. 标记耦合
3. 控制耦合
4. 外部耦合
5. 公共耦合
6. 内容耦合

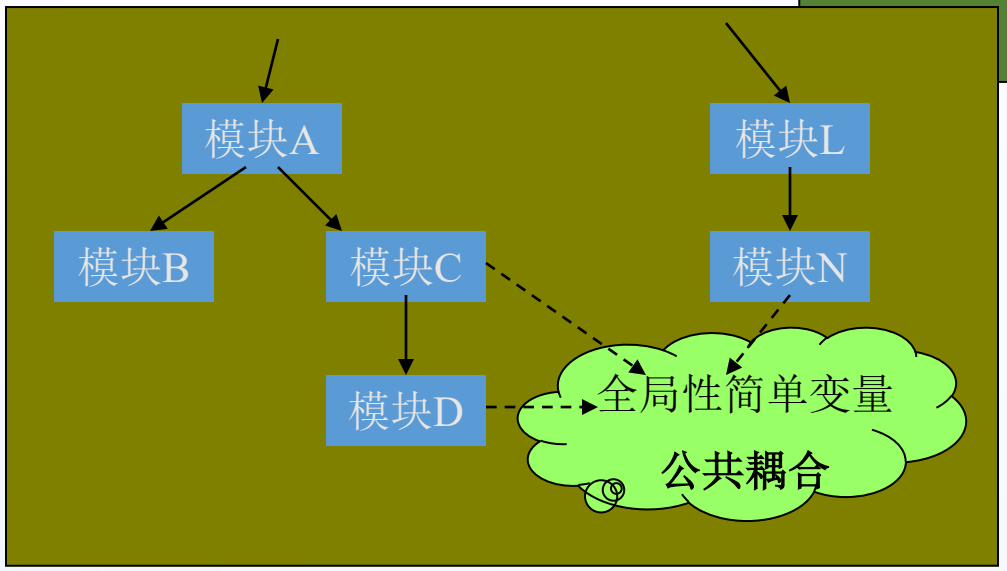
弱耦合

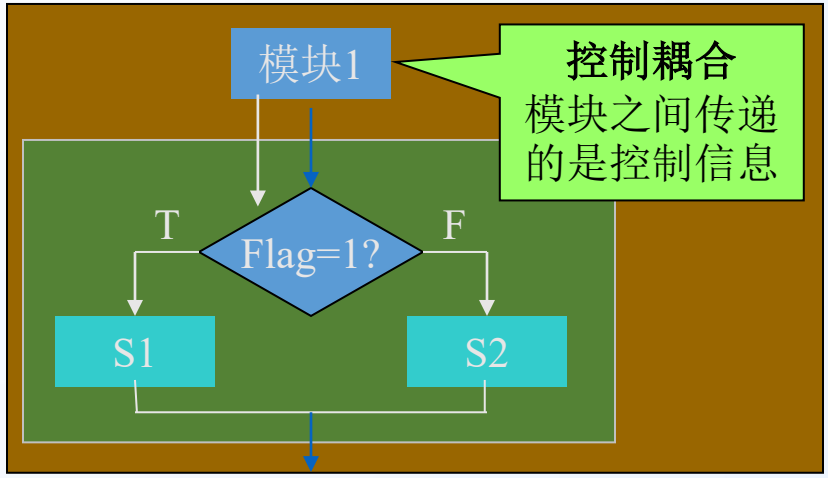
中耦合

较强耦合

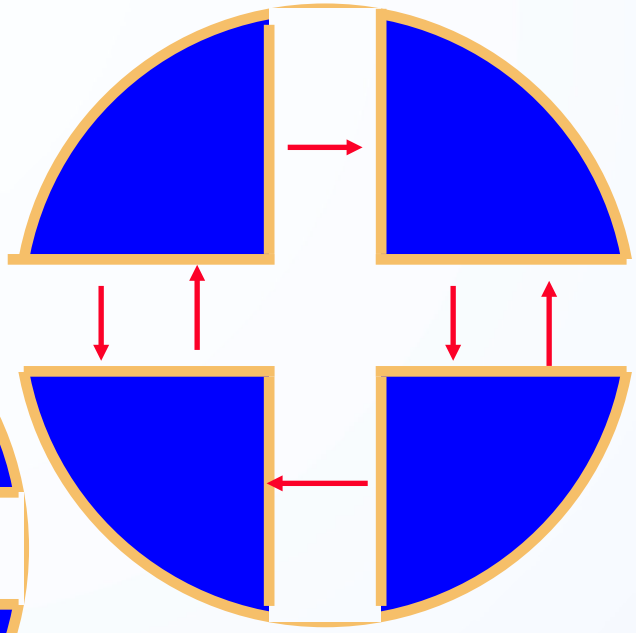
强耦合



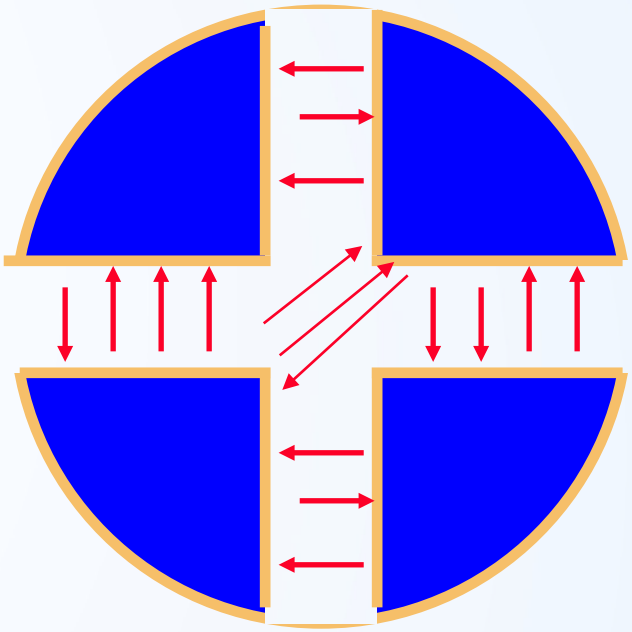




无耦合 - 没有依赖关系



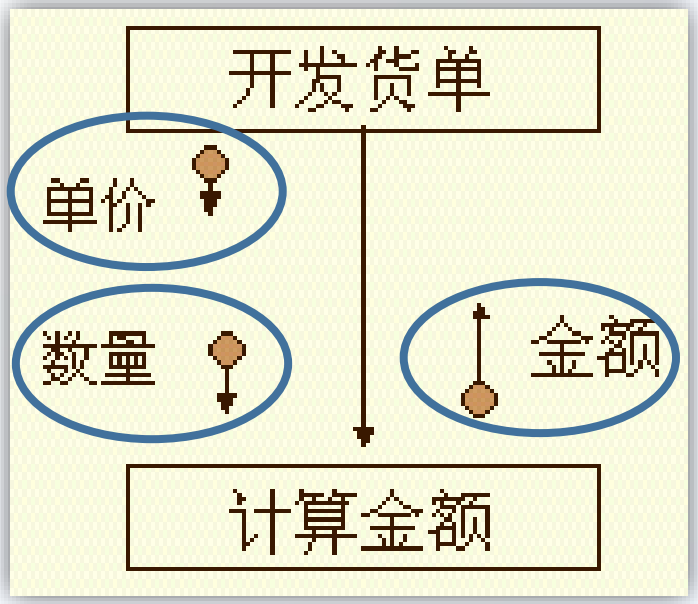
松散耦合 - 有少量依赖关系



紧密耦合 - 有很多依赖关系

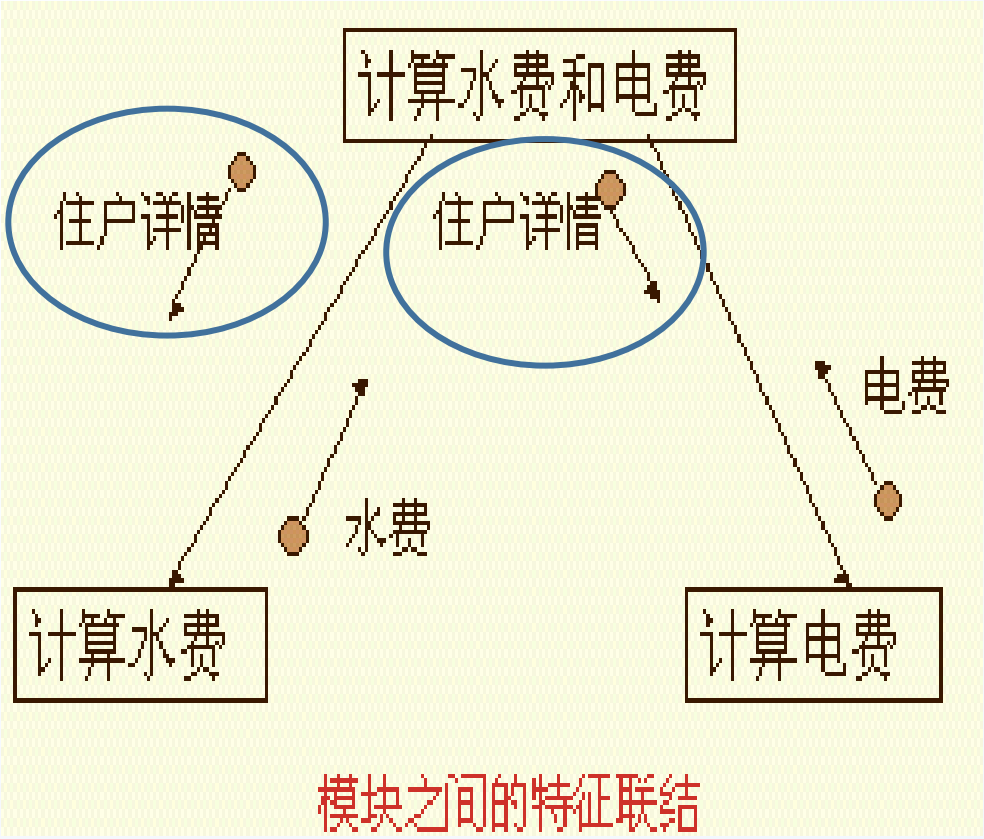
◆ 数据耦合（Data Coupling）：

如果一个模块访问另一个模块时，彼此之间是通过数据参数（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，则称这种耦合为数据耦合。这是模块之间影响最小的耦合关系。



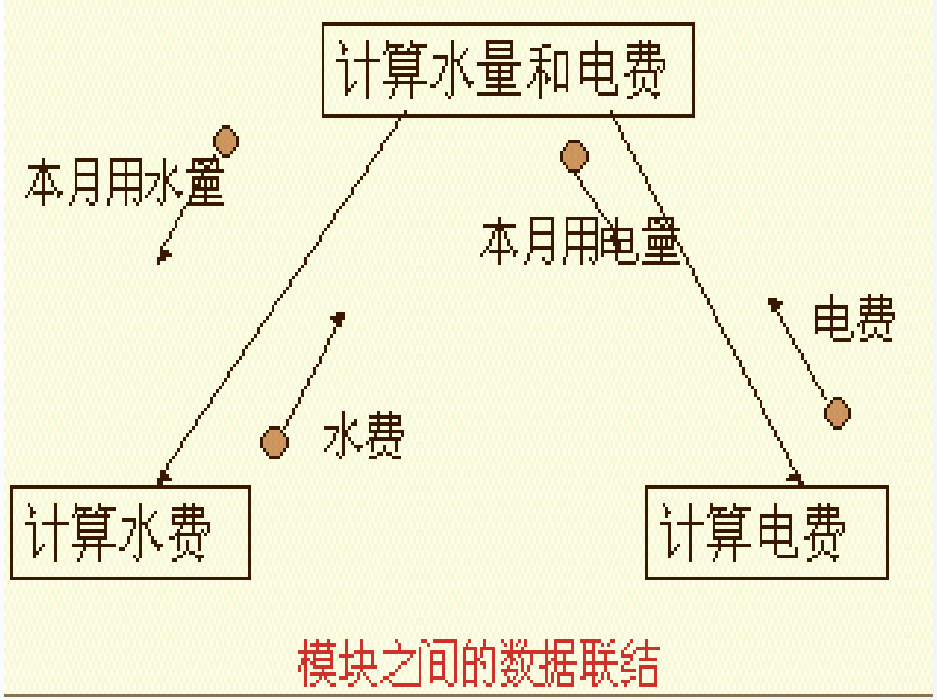
◆ 标记耦合（Stamp Coupling，特征耦合）：

如果两个模块间的通讯信息是**若干参数**，其中某一传递参数为某一数据结构类型或用户创建的数据类型，称这种耦合为标记耦合。



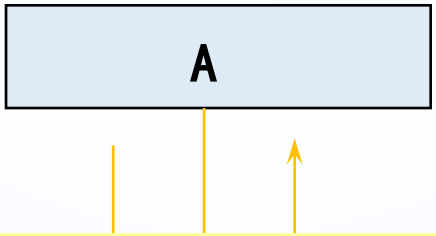
标记耦合与数据耦合之间的相互转化：

住户详情数据结构中包括“本月用水量”、“本月用电量”。上面的“标记耦合”图可改进为下面的“数据耦合”图。

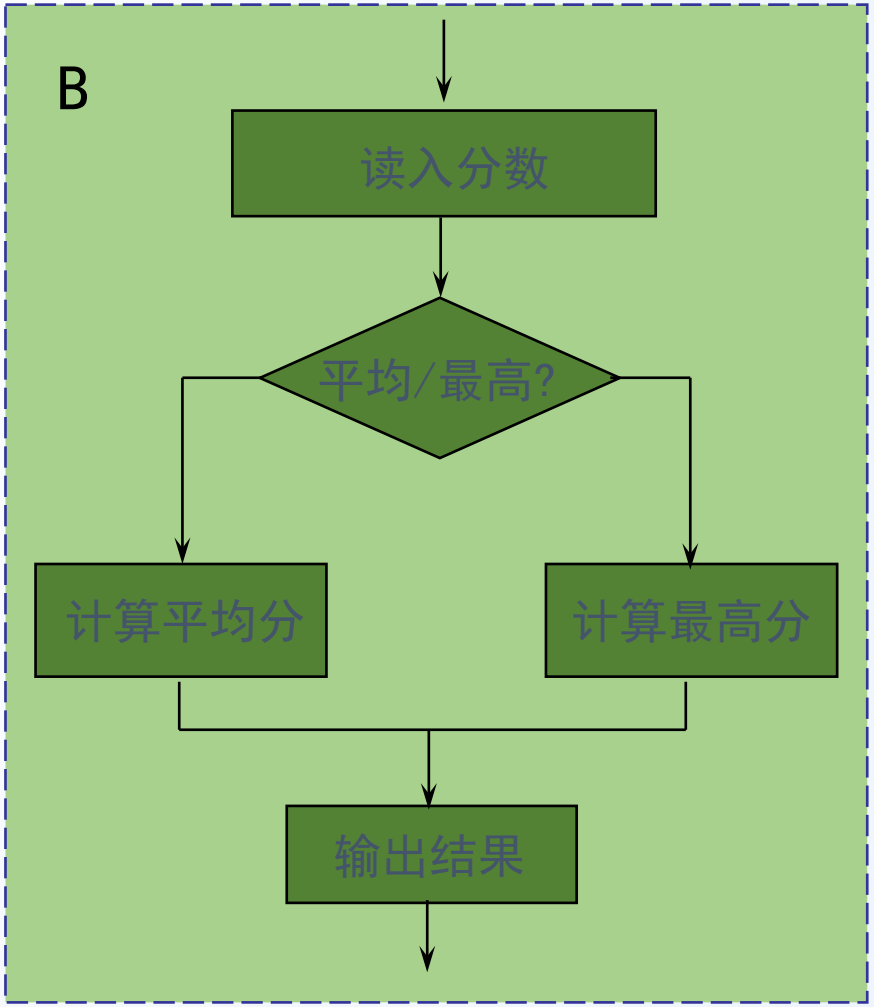


◆控 制 耦 合 （ Control Coupling）：

一模块通过开关量、标志、名字等控制信息，明显地控制另一模块的功能，则称为控制耦合。



调用逻辑性模块B时，须先传递控制信号(平均分/最高分)，以选择所需的操作。控制模块必须知道被控模块的内部逻辑，增强了相互依赖。

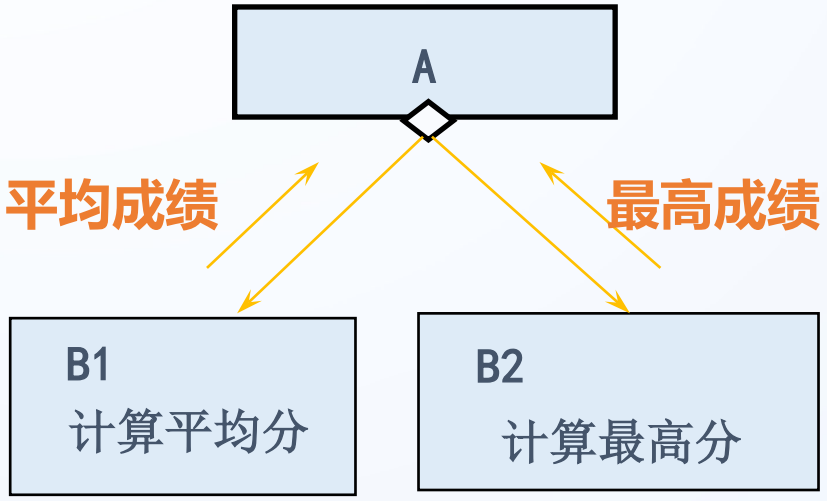


修正控制耦合的方法：

控制耦合增加了理解和编程的复杂性，调用模块必须知道被调模块的内部逻辑，增加了相互依赖。

去除模块间控制耦合的方法：

- (1) 将被调用模块内的判定上移到调用模块中进行；
- (2) 被调用模块分解成若干单一功能模块。



◆ 外部耦合（External Coupling）：

两个模块均与同一外部环境关联，如外部引入的数据格式、通信协议或设备接口等，它们之间便存在外部耦合

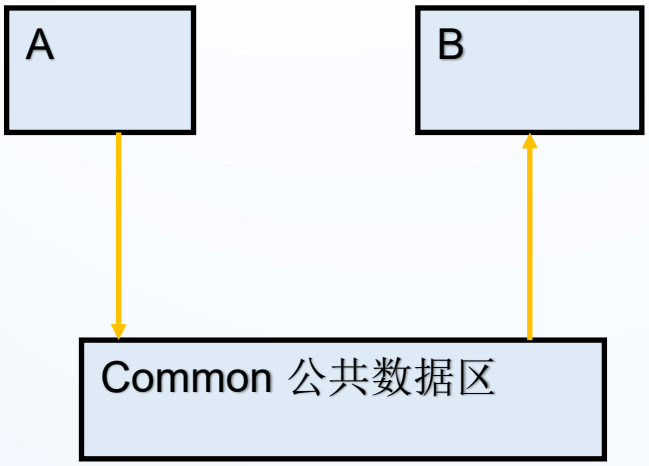
外部耦合必不可少，但在系统中应该控制外部耦合模块的数量，使这类模块数目应尽量少。

公共耦合（Common Coupling）：

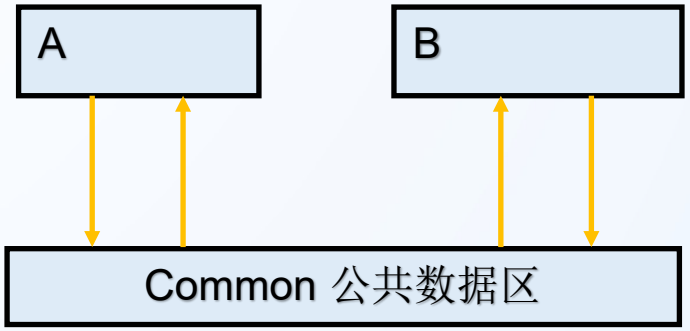
如果一组模块引用了同一个公共数据区，则称为公共耦合。

公共数据区指：

- 全局数据结构
- 共享通讯区
- 内存公共覆盖区等



松散公共耦合

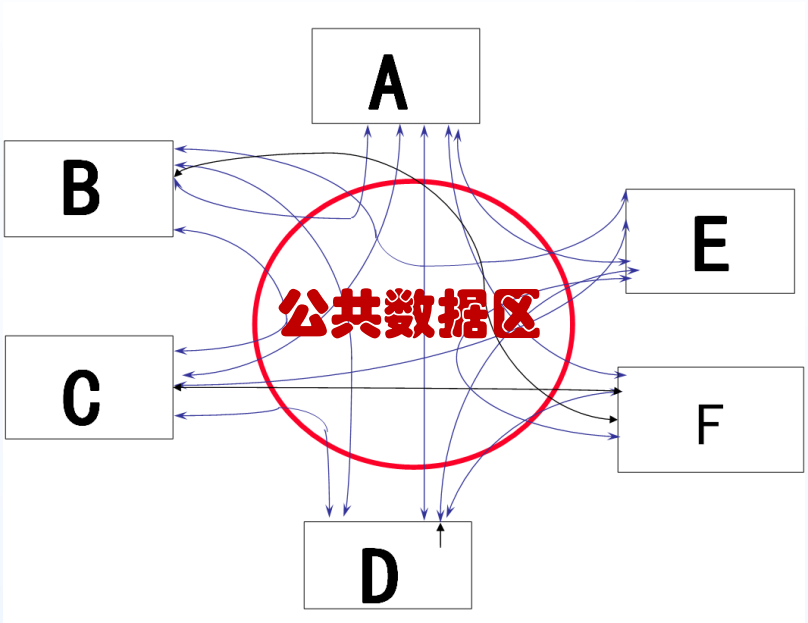


紧密公共耦合

公用耦合是一种不良的耦合关系，它给模块的维护和修改带来困难。

- (1)软件可理解性降低　　(模块间存在错综复杂的联系)
- (2)软件可维护性差　　(修改变量名或属性困难)
- (3)软件可靠性差　　(公共数据区及全程变量无保护措施)

慎用公共数据区和全程变量!!!



6个模块共享一个公共数据区，模块之间存在错综复杂的联系

内容耦合（Content Coupling）：

如果一个模块和另一个模块的内部属性（即运行程序和内部数据）有关，则称为内容耦合。

例如：模块A

TRC：……

模块B

GOTO TRC

模块A与模块B存在内容耦合，这是一种**最不好的耦合形式 !!!**

◆ 模块化设计的原则和目标：

耦合是影响软件复杂程度和设计质量的重要因素

目标：建立模块间耦合度尽可能松散的系统。

◆ 如何降低模块间耦合度?

(1) 如模块必须存在耦合，选择适当的耦合类型

原则：尽量使用数据耦合

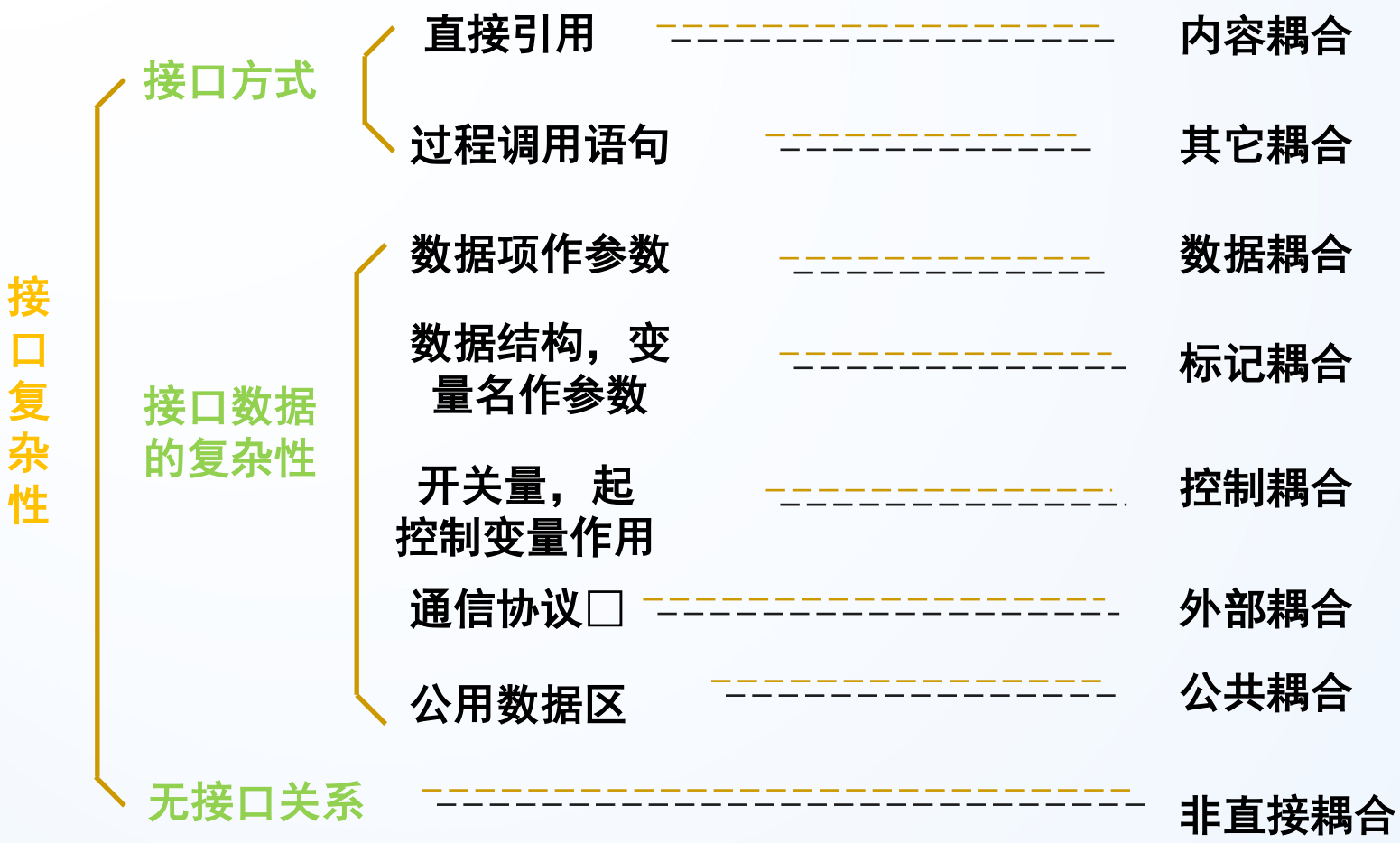
少用控制耦合

限制公共耦合的范围

坚决避免使用内容耦合

浙江大学 ZHEJIANG UNIVERSITY

(2) 降低模块间接口的复杂性



2.2.2 模块的内聚性

内聚:(cohesion)在模块内部各部分之间的联系，它描述模块功能的相对强度--块内联系；

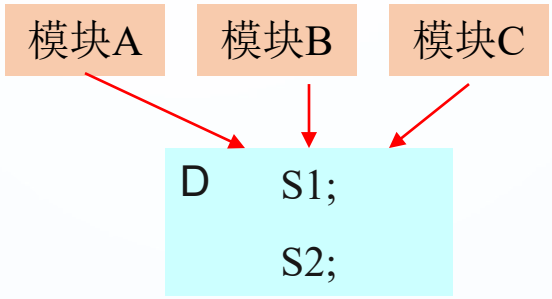
- 内聚性是指一个模块内部各组成部分之间（如语句之间或语句段之间）的联系的紧密程度。
- 总希望它越高越好。也就是说，在模块划分时应该尽量把相互有密切联系的部分划分在同一模块，而把不相关的部分尽量不要凑在一个模块。
- 内聚性常见的几种类型有7种：功能内聚、顺序内聚、通信内聚、过程内聚、时间内聚、逻辑内聚、偶然内聚。

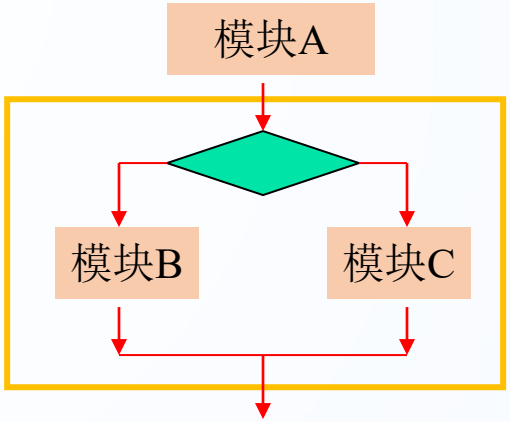
高 ← 内聚性 → 低

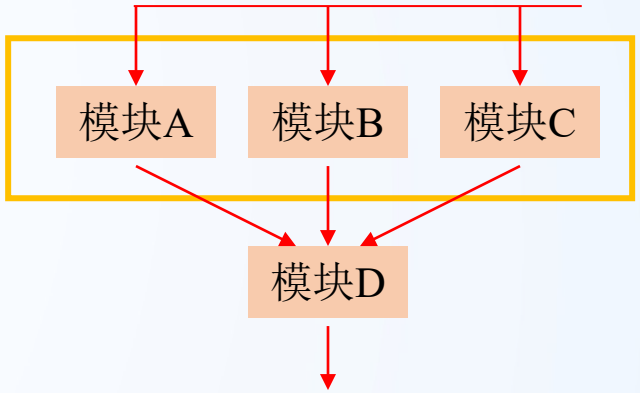
功能内聚	顺序内聚	通信内聚	过程内聚	时间内聚	逻辑内聚	偶然内聚

强（功能单一） ← 模块独立性 → 弱（功能分散）

低内聚
- 偶然内聚：一个模块完成一组任务，任务之间的关系很松散，类似于公共语句。
- 逻辑内聚：若干个逻辑功能类似的任务组成一个模块。
- 时间内聚：若干个任务必须在同一段时间内执行。如初始化工作。







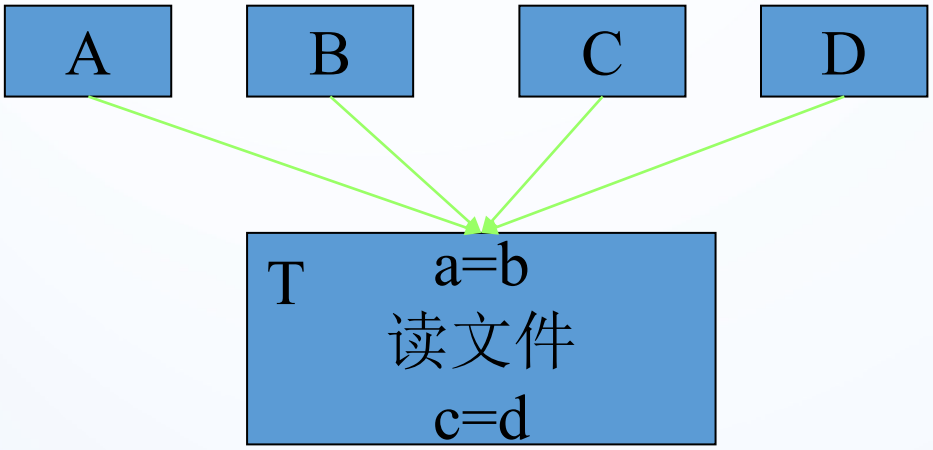
中内聚
- 过程内聚：模块内的处理元素是相关的，且必须以特定次序执行。
- 通信内聚：模块中所有元素都使用同一个输入数据，和/或产生同一个输出数据。

高内聚
- 顺序内聚：模块中所有处理元素和同一个功能密切相关，且这些处理必须顺序执行。
- 功能内聚：所有处理元素属于一个整体，完成一个单一的功能。

偶然内聚：

如果一个模块的内部各组成部分的处理动作彼此没有任何联系，则称为**偶然内聚**。往往因为有多个模块有一段程序是相同的，为了节省程序量，而把这一段程序作成一个模块，而实际上这段程序内部的各部分没有什么联系。

例：



缺点：

（1）不易修改。当模块A，要修改成不需要a=b的语句时，模块T就不适用了。

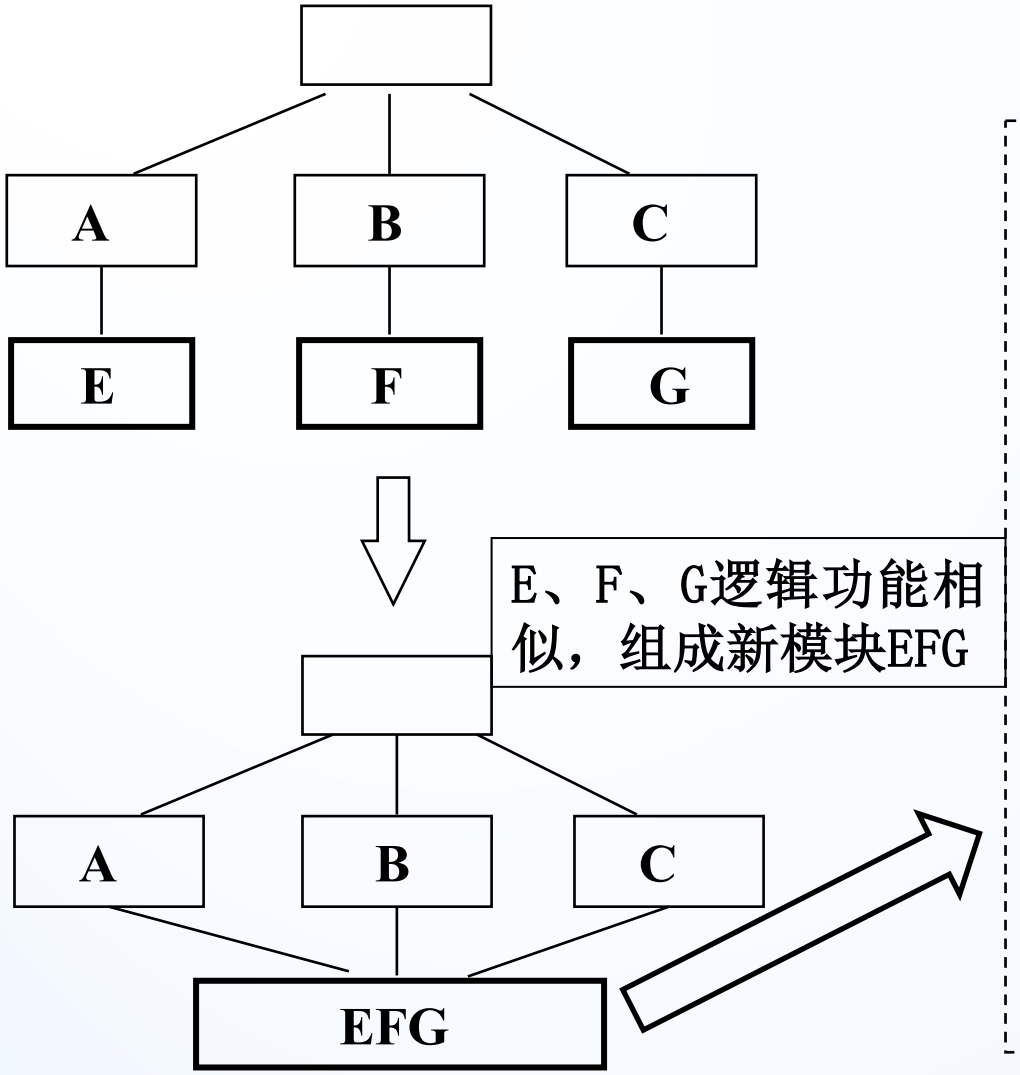
（2）模块T，没有明确的含义，功能不明确，甚至给不出一个名称。

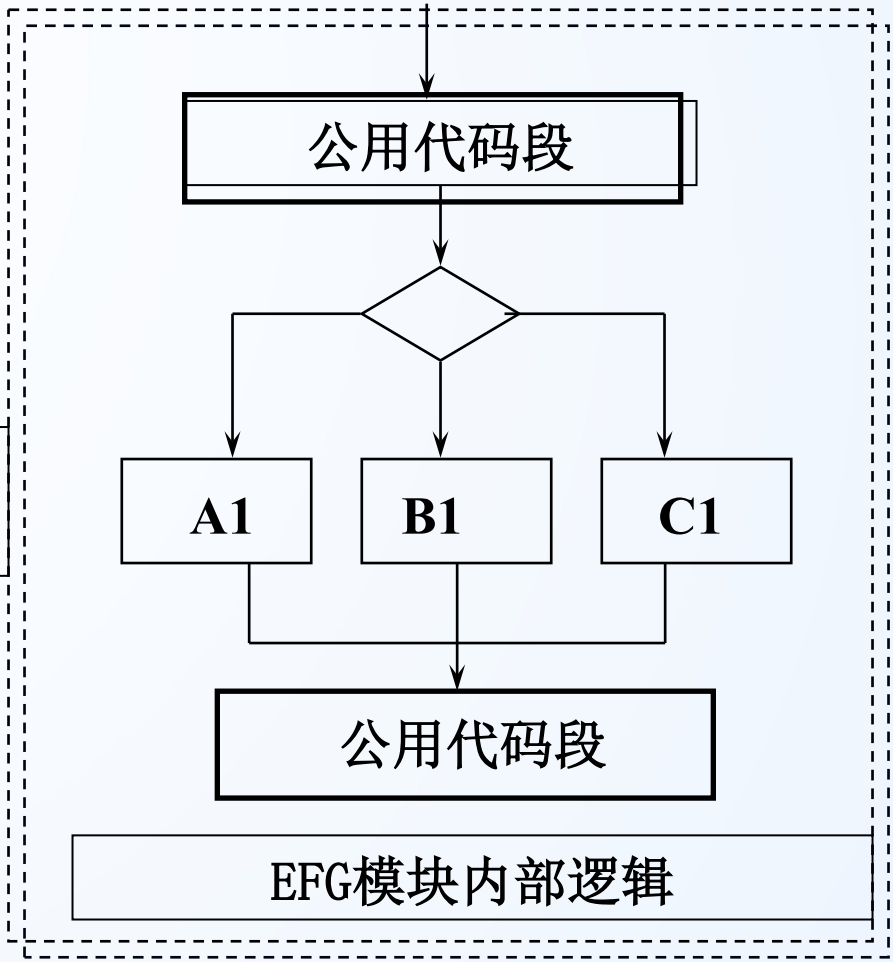
逻辑内聚：

如果一个模块内部的各组成部分的处理动作在逻辑上相似，但功能都彼此不同或无关，则称为逻辑内聚。

一个逻辑内聚模块往往包括若干个逻辑相似的动作，使用时可以选用一个或几个功能。

例如：把编辑各种输入数据的功能放在一个模块中。


A
B
C
E
F
G
E、F、G逻辑功能相似，组成新模块EFG
A
B
C
EFG


公用代码段
A1
B1
C1
公用代码段
EFG模块内部逻辑

时间内聚：

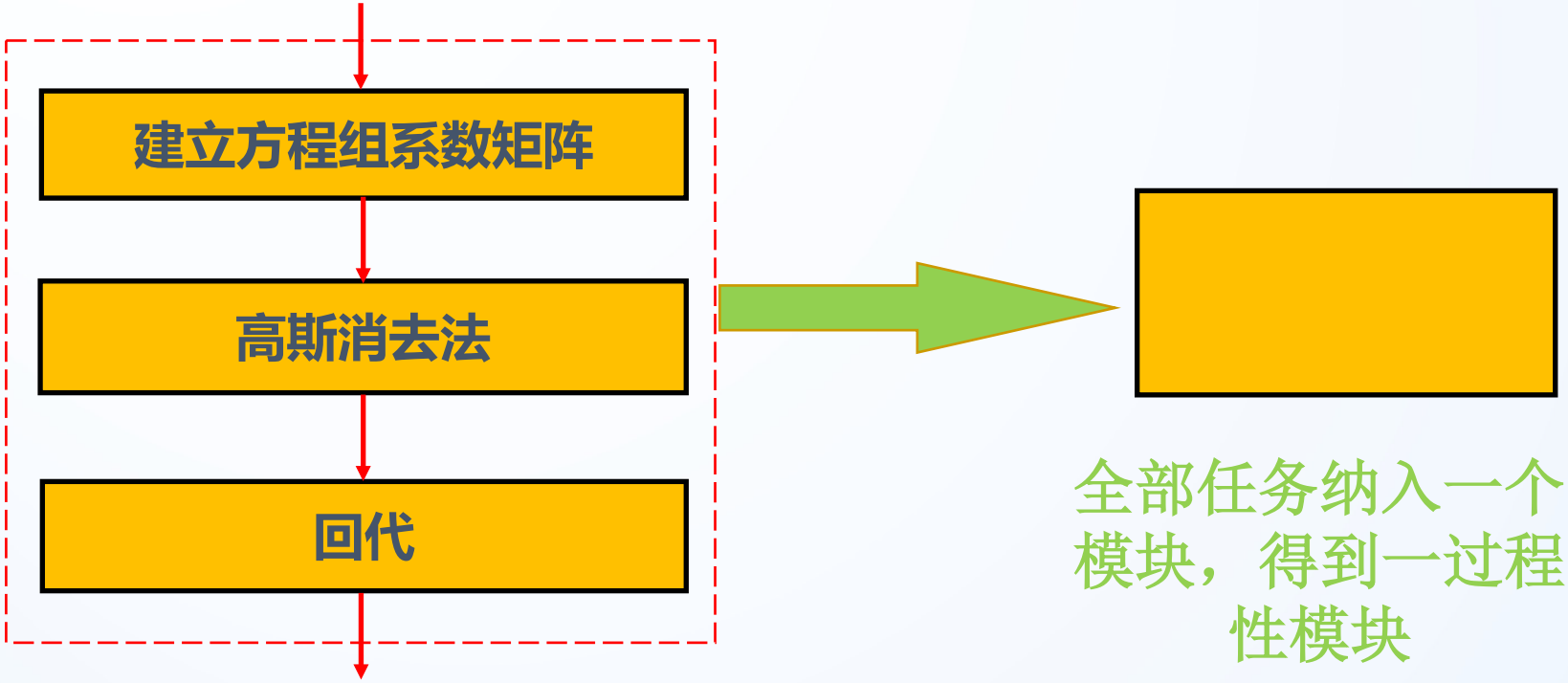
模块完成的功能必须在同一时间内执行，这些功能只因时间因素关联在一起。

例如：

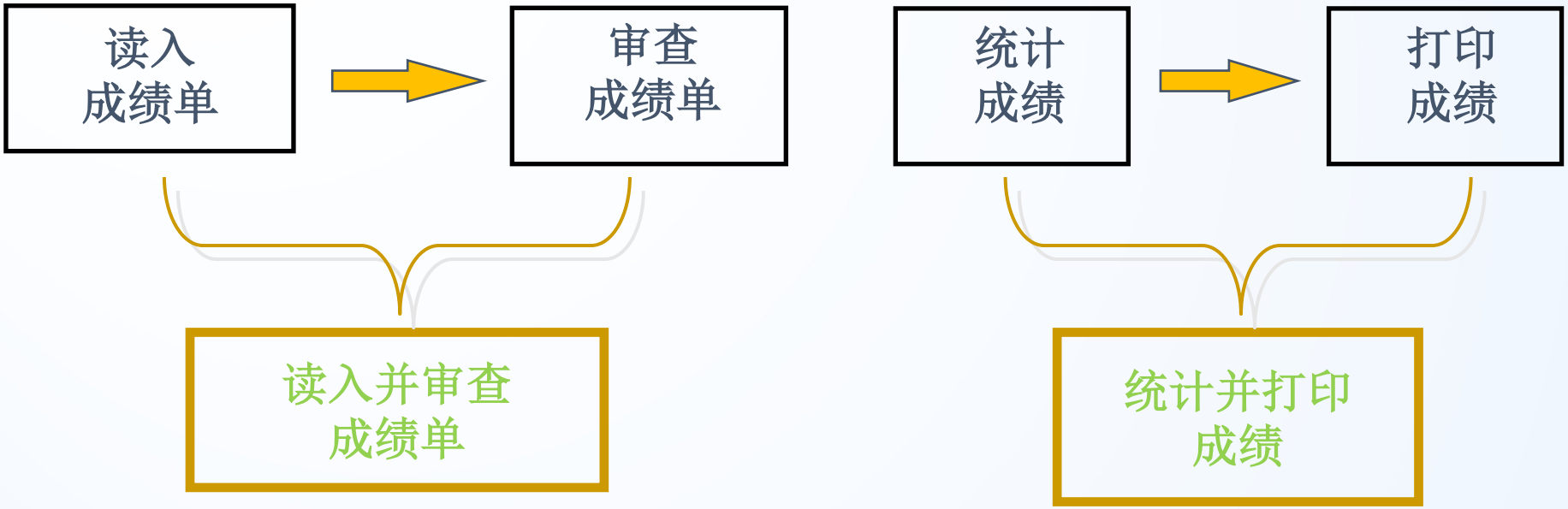
- 初始化系统模块、
- 系统结束模块、
- 紧急故障处理模块等。

过程内聚：

如果一个模块内各组成部分的处理动作各不相同，彼此也没有联系，但它们都受同一个控制流支配，决定它们的执行次序，称为过程内聚。

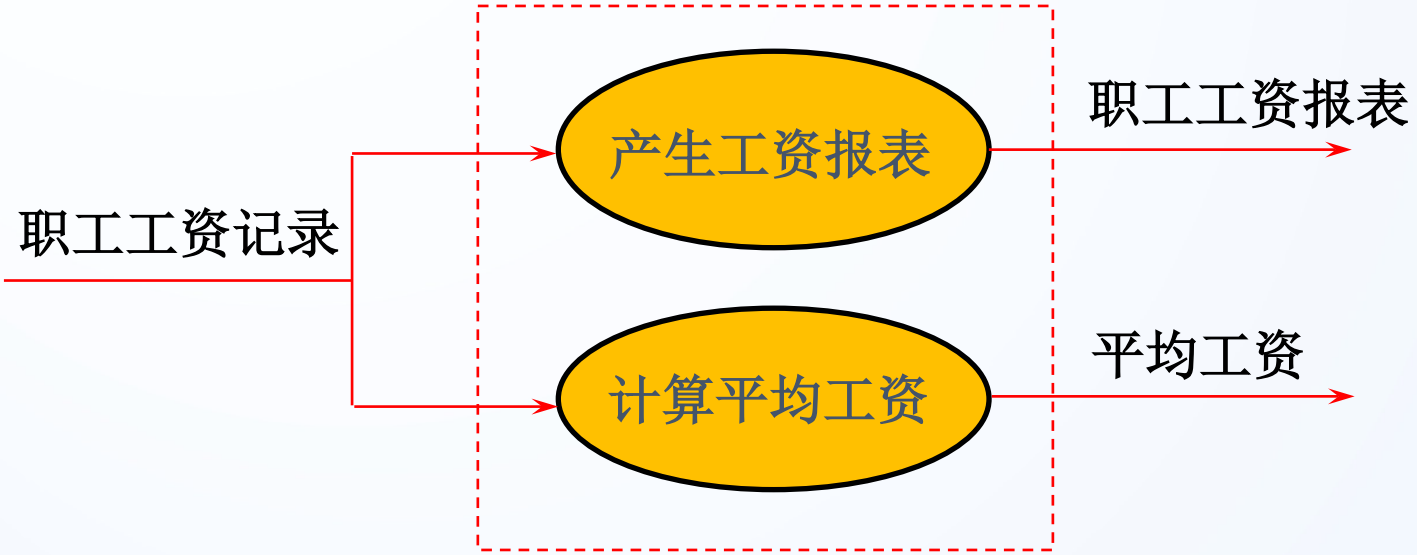


高斯消去法解题流程


读入
成绩单
审查
成绩单
统计
成绩
打印
成绩
读入并审查
成绩单
统计并打印
成绩

通信内聚：

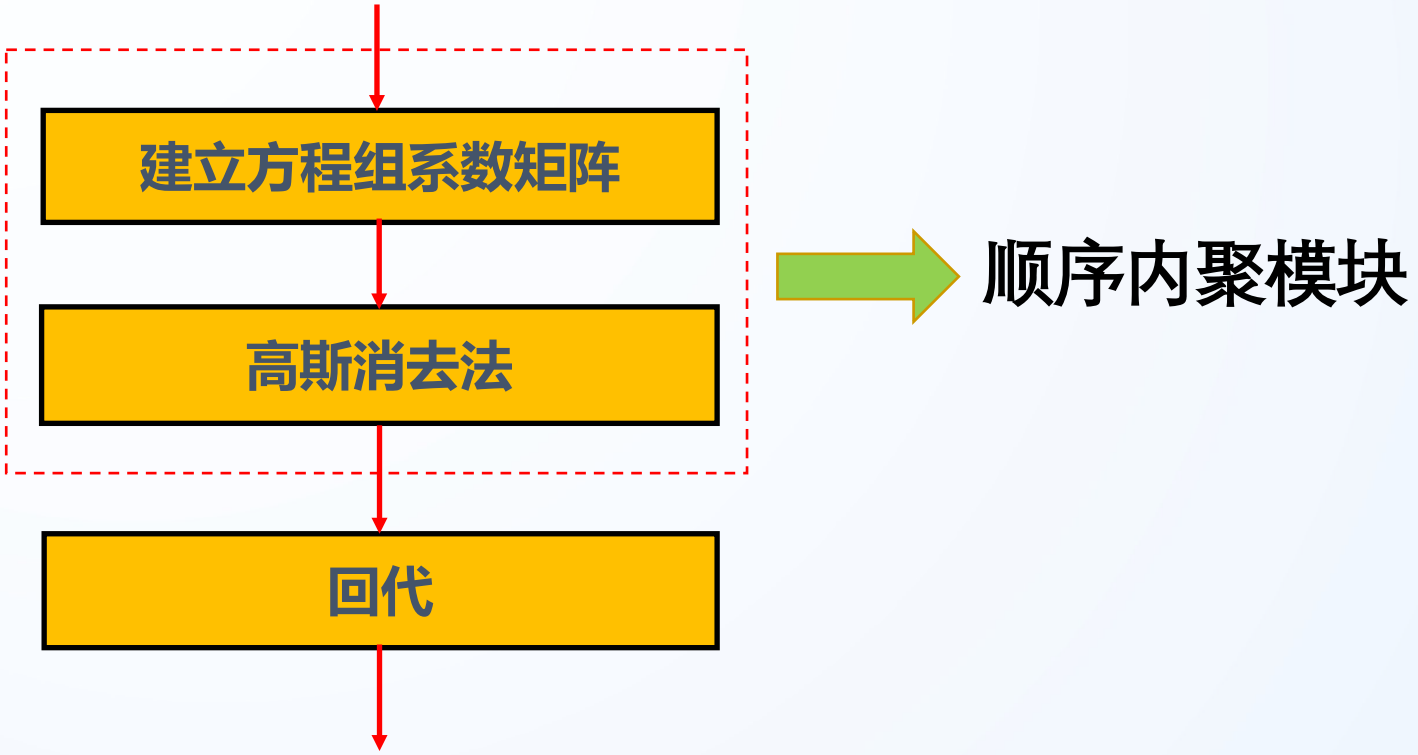
如果一个模块内各组成部分的处理动作都使用相同的输入数据或相同的输出数据，称为通信内聚。



产生职工工资报表并计算平均工资模块

顺序内聚：

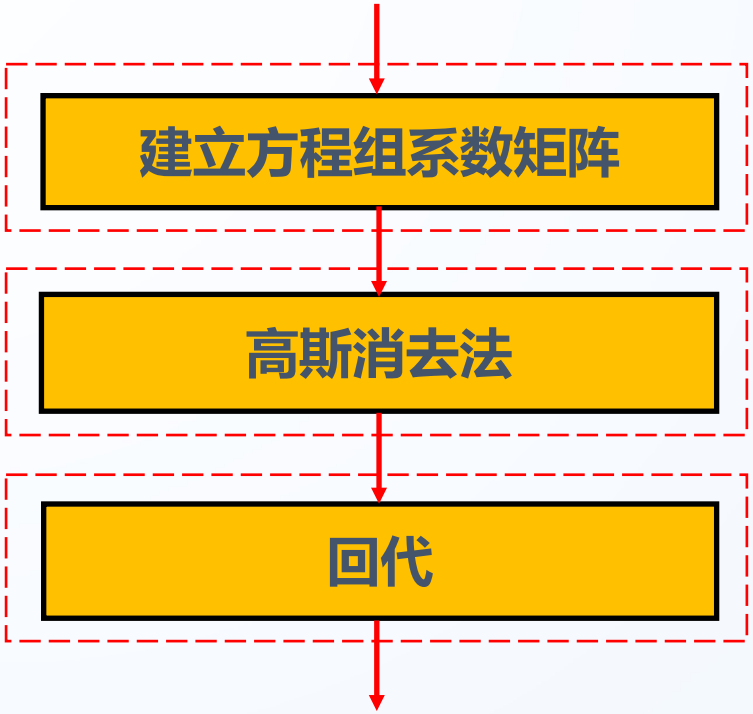
如果模块内一个内部成分的输出是另一个内部成分的输入，则属顺序内聚。也就是说，模块中一个成分依赖于另一个成分。



高斯消去法解题流程

功能内聚：

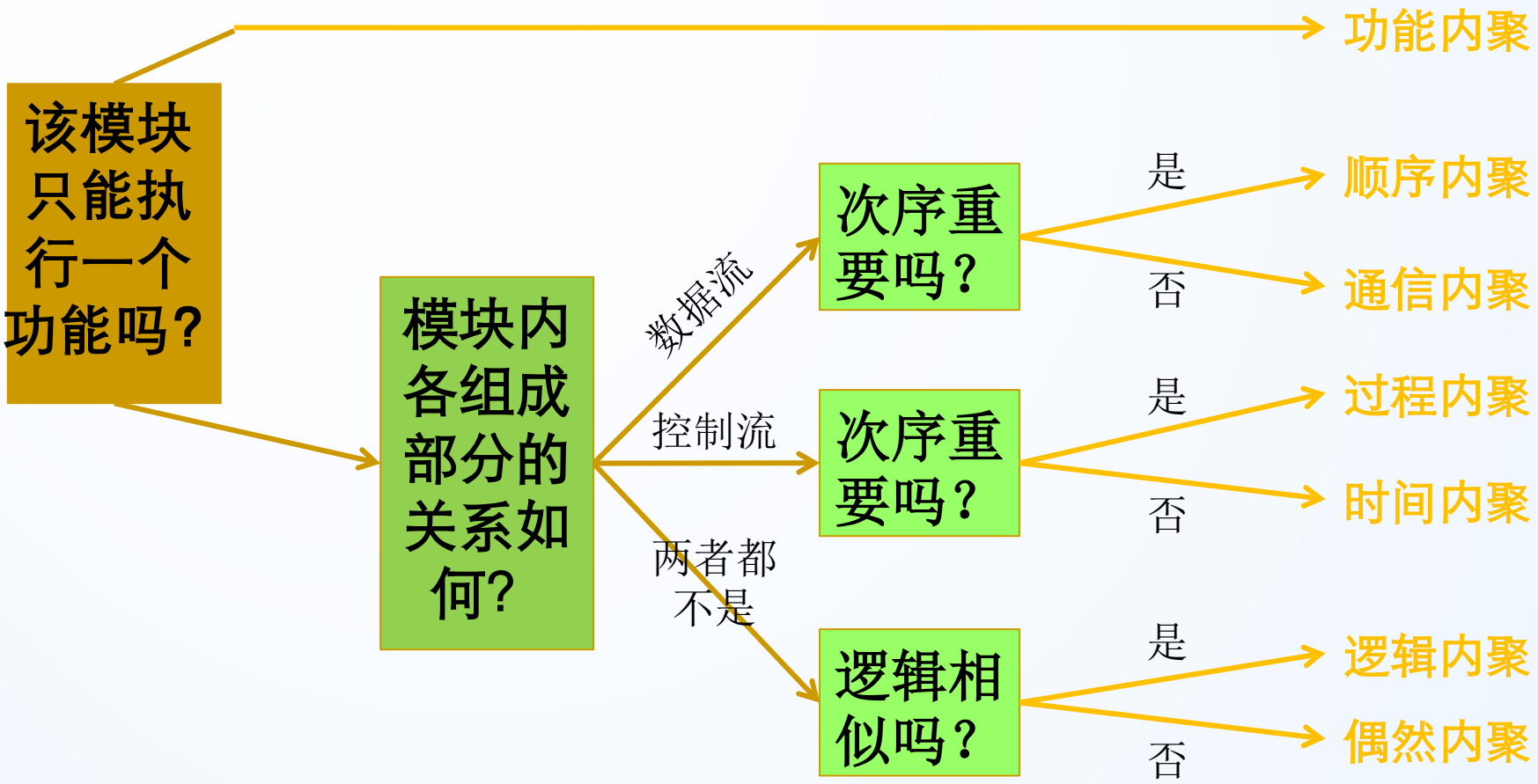
一个模块仅包括为完成某个功能所必须的所有成分，即模块所有成分共同完成一个功能，缺一不可，则成为功能内聚。**内聚性最强！！**



高斯消去法解题流程

浙江大学 ZHEJIANG UNIVERSITY

模块内聚性的判定方法：



2.3 程序结构图

结构图(Structured Charts，简称SC)是准确表达程序结构的图形化表示方法，它能清楚地反映出程序中各模块间的层次关系和联系。与数据流图反映数据流的情况不同，结构图反映的是程序中控制流的情况。

(1)程序结构图的组成

程序结构图中主要包括：模块，调用，数据和控制

模块：以矩形框表示，框中标有模块的名字。对于已定义（或者已开发）的模块，则可以用双纵边矩形框表示，模块的名字应该能准确地反映模块的功能。

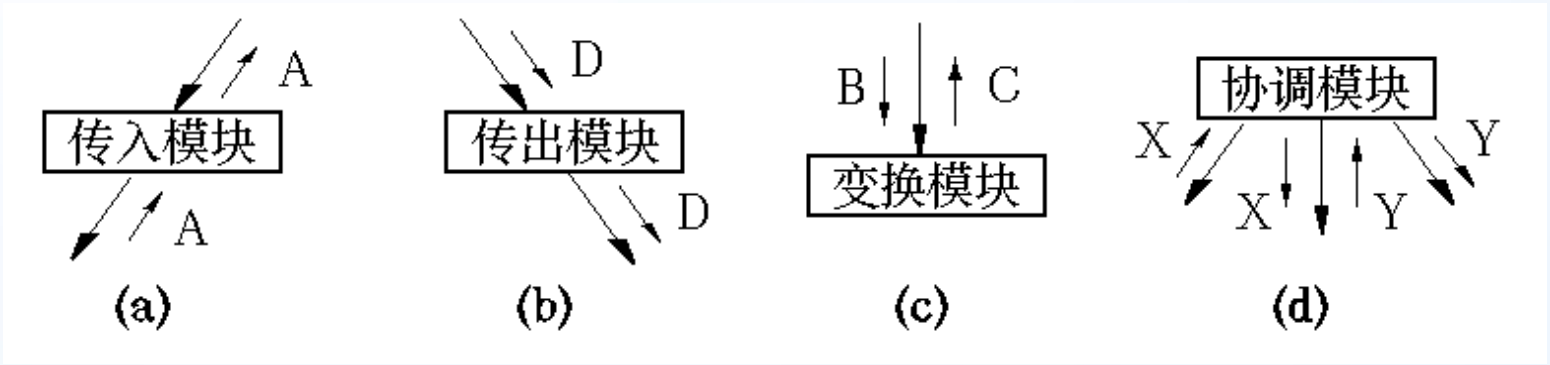
结构图中可能出现以下四种类型的模块

传入模块： 从下属模块取得数据，经过某些处理，再将其传送给上级模块

传出模块： 从上级模块取得数据，进行某些处理，传送给下属模块

变换模块： 从上级模块取来数据，进行特定处理后，送回原上级模块

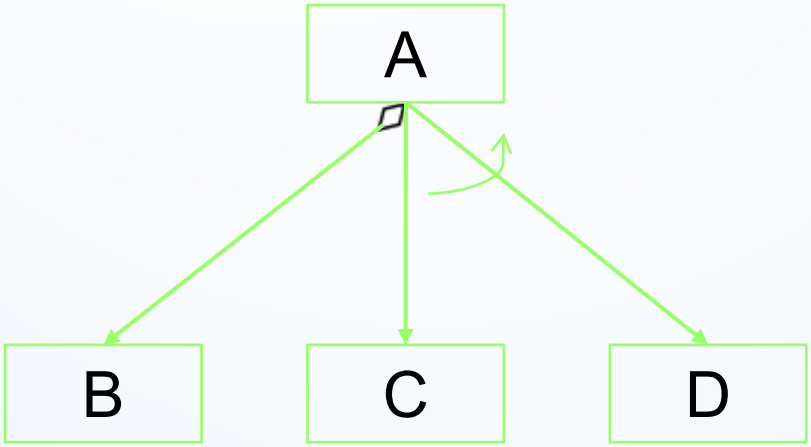
协调模块： 对其下属模块进行控制和管理的模块



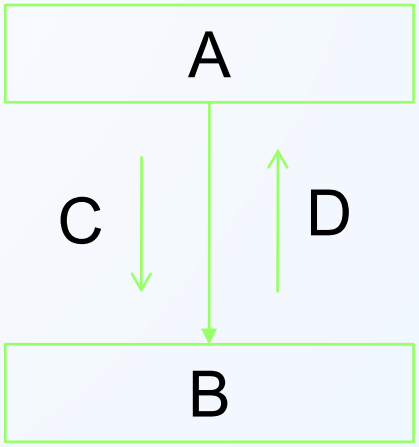
调用： 从一个模块指向另一个模块的箭头表示了前一个模块中含有对后一个模块的调用.

数据： 以表示调用关系的长箭头旁边的短箭头表示，短箭头的方向和名字分别表示调用模块和被调用模块之间信息的传递方向和数据.

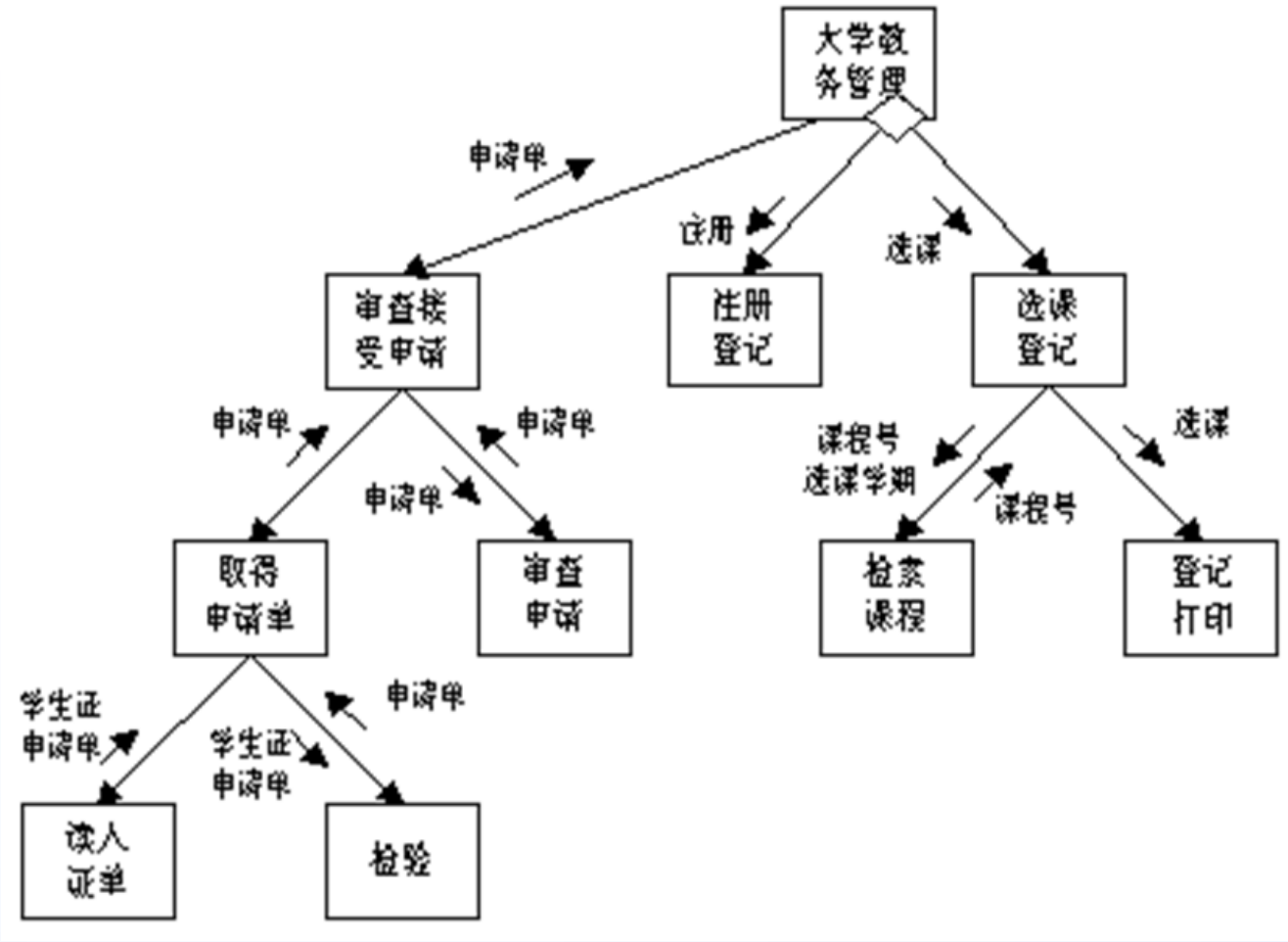
控制： 当模块A有条件的调用模块B时，在箭头的起点标以菱形。模块A反复地调用模块D时，另加一环状箭头。



控制

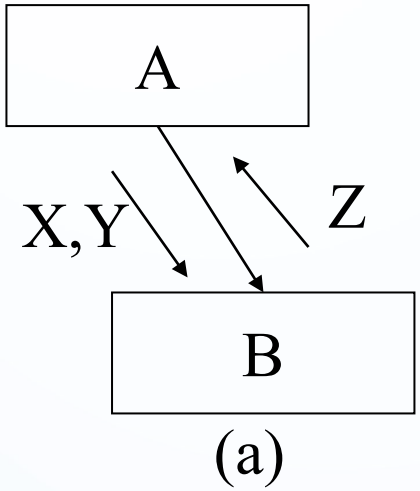


调用和数据



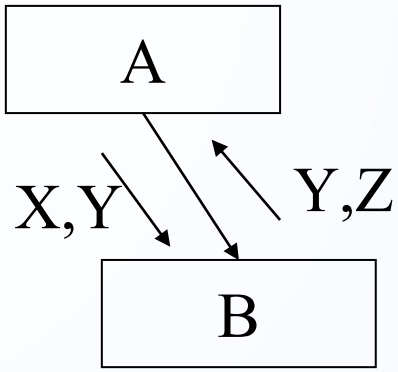
大学教务管理系统的程序结构图

(2)一些常见的结构图例



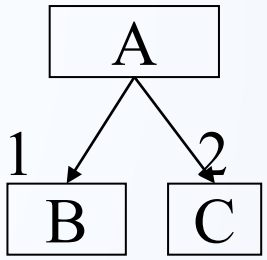
(a)

图a,表示模块A含有对B的调用.当A调用B时,A将数据X,Y传入B.B执行完成后,返回到A时,将数据Z传送给A.



(b)

图b表示,当A调用B时,A将数据X,Y传入B.B执行完成后,返回到A时,将数据Z传送给A,同时,将修改过的Y也传送到A



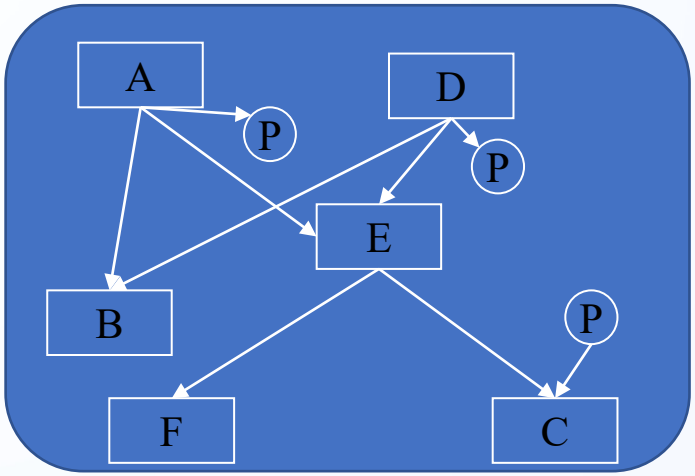
	IN	OUT
1	X,Y	Z
2	Z	

(C)

(3)画结构图注意事项

一个模块在结构图中只能出现一次，否则修改模块结构时就需要修改多处，很容易造成错误.

为了避免线条过多交叉，增强图的易理解性，可以如图用代号表示调用关系的。



值得注意的是，结构图着重反映的是模块间的隶属关系，即模块间的调用关系和层次关系。它和程序流程图（常称为程序框图）有着本质的差别。程序流程图着重表达的是程序执行的顺序以及执行顺序所依赖的条件。结构图则着眼于软件系统的总体结构，它并不涉及模块内部的细节，只考虑模块的作用，以及它和上、下级模块的关系。而程序流程图则用来表达执行程序的具体算法。

2.3.1从数据流图导出程序结构图

数据流图一般有两种典型的结构，分别反映两种问题类型：**变换型问题和事务型问题**，它们的数据流图和结构图都有明显的特征。

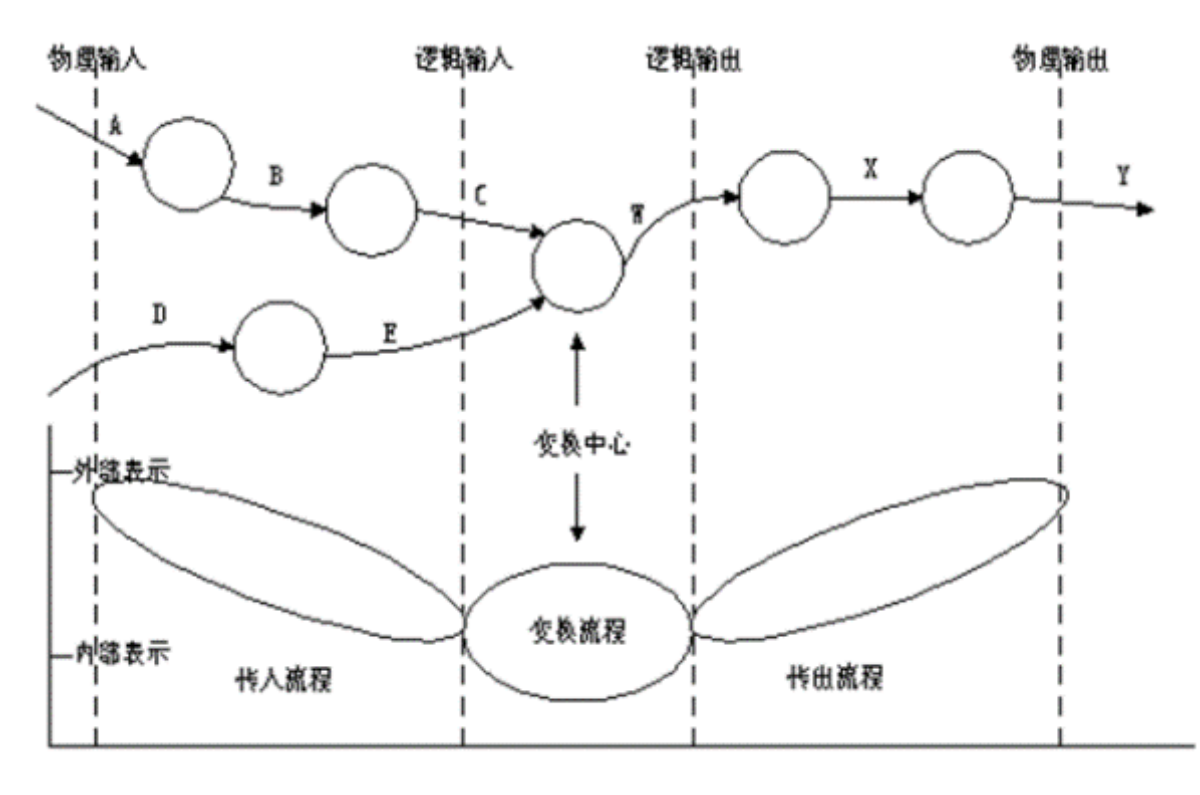
(1) 变换型问题

数据处理问题中，我们通常会遇到这样一类问题，即从（程序）“外部”取得数据（例如从键盘、磁盘文件等），对取得的数据进行某种变换，然后再将变换得到的数据传回给“外部”。当数据流图或其中某一段数据流表现出上述特征时，该数据流图或该段数据流图表示的就是一个变换型问题.

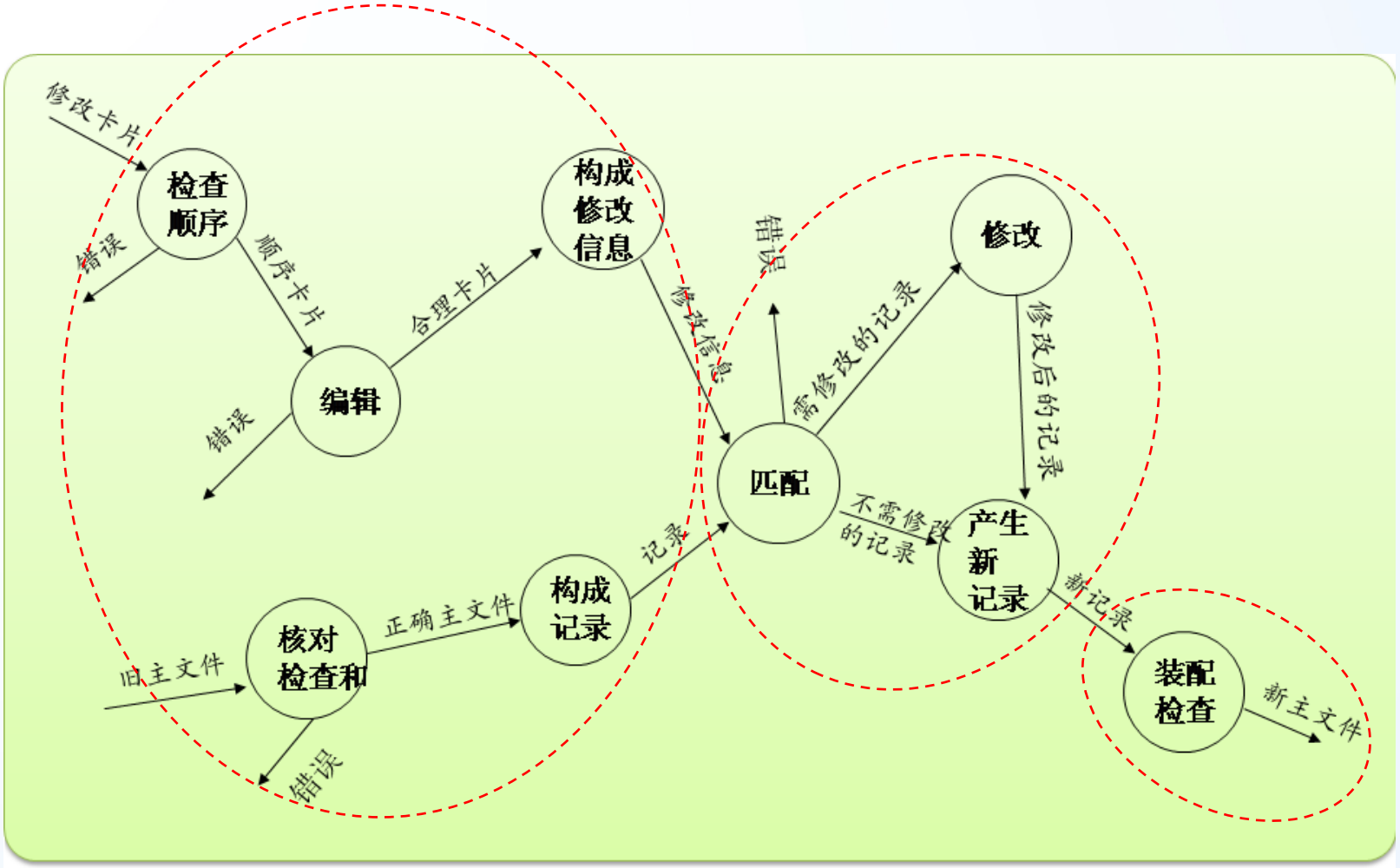
在数据流图中，变换型结构表现为一种线形结构，它明显地分为三个部分：

- **输入**：取得数据这一过程称为传入信息（数据）流程
- **主加工（变换中心）**：变换数据的过程称为变换信息（数据）流程
- **输出**：传回数据过程称为传出信息（数据）流程

02 程序结构设计



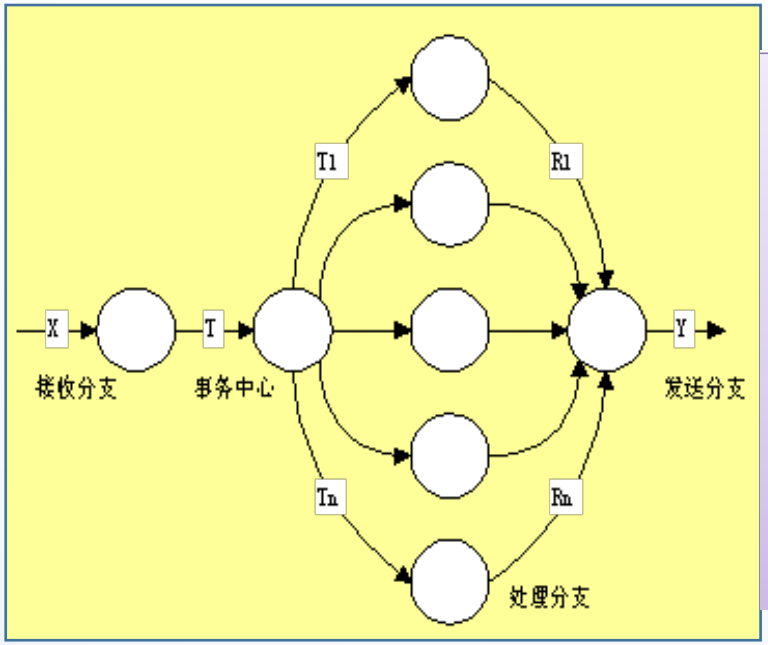
- 主加工的输入数据流成为逻辑输入，
- 系统的输入端的数据流称为物理输入
- 主加工的输出部分成为逻辑输出
- 系统的输出端数据流称为物理输出

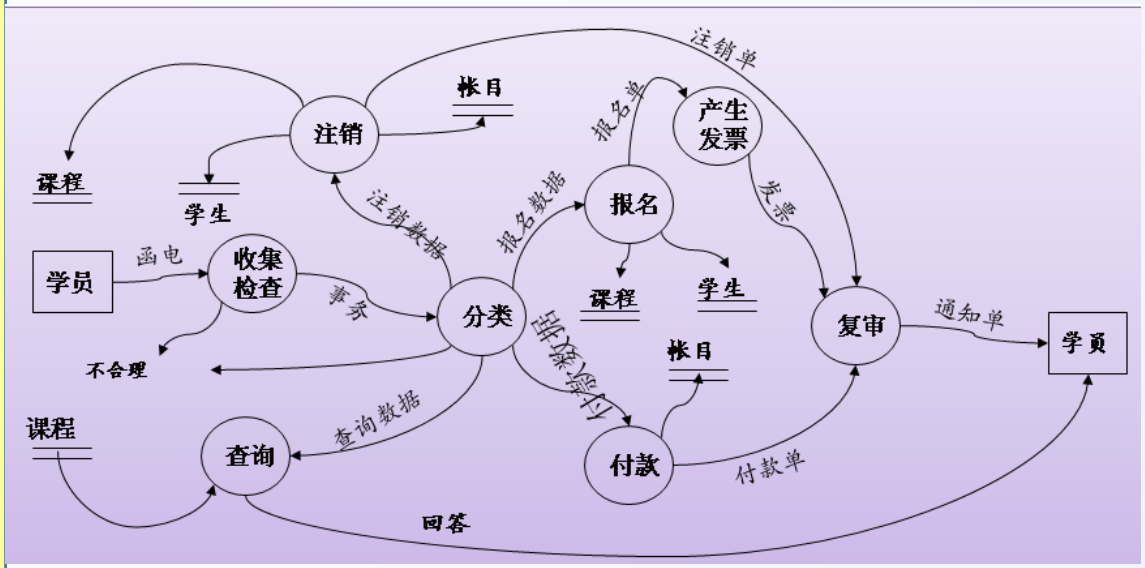

修改卡片
检查顺序
错误
顺序卡片
编辑
错误
合理卡片
构成修改信息
修改信息
匹配
错误
需修改的记录
修改
修改后的记录
不需修改的记录
产生新记录
新记录
装配检查
新主文件
记录
构成记录
正确主文件
核对检查和
旧主文件
错误

(2)事务型问题

在实际中，我们还常常会遇到另一类问题，即通常在接受某一项事务后，根据事务的特点和性质，选择分派给一个适当的处理单元，然后给出结果，这类问题就是事务型问题。

它的特点是，数据沿着接收分支把外部信息（数据）转换成一个事务项，然后计算该事务项的值，并根据它的值从多条数据流中选择其中的某一条数据流。发出多条数据流的处理单元叫事务中心。





(3)从变换型结构的数据流导出程序结构的步骤

第1步 找出主加工，确定其逻辑输入和逻辑输出

- 从物理输入端开始，一步一步向系统中间移动，直到达到这样一个数据流：

  它已经不能在被看作为系统的输入，其前一个数据流就是系统的逻辑输入。即离物理输入最远、但仍可以看作为系统的输入的那个数据流为系统的逻辑输入。

- 从物理输出端开始，一步一步向系统中间移动，也可以找出离物理输出端最远、但仍可以看作为系统的输出的那个数据流，它就是系统的逻辑输出。
- 每一股输入和输出都可以如此找到其逻辑输入和逻辑输出
- 位于逻辑输入和逻辑输出之间的部分就是系统的

几股数据流汇合之处往往是系统的主加工

(3)从变换型结构的数据流导出程序结构的步骤

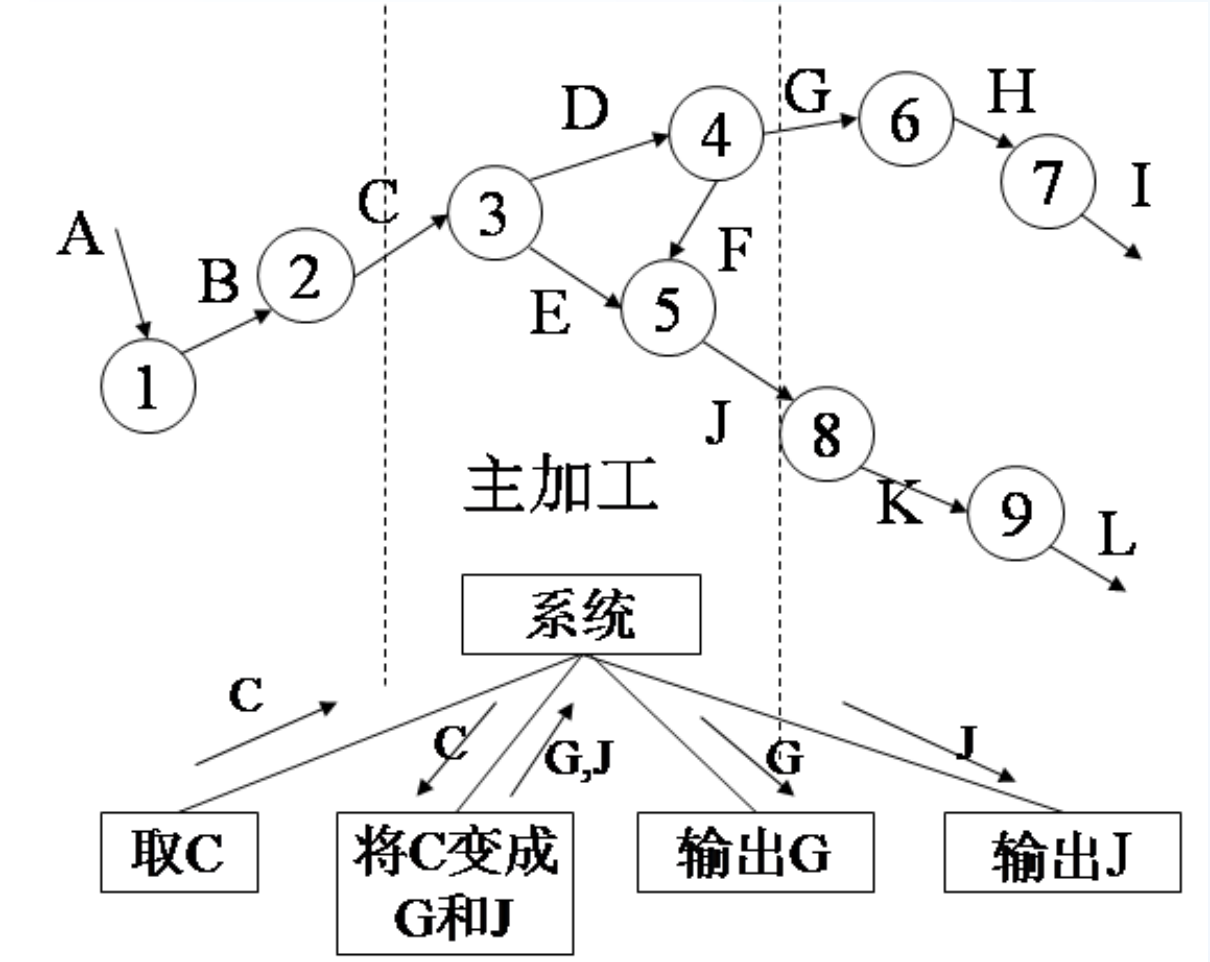
第2步 设计模块结构的顶层（第0层）及其下一层（即第一层）

- 由顶向下设计是先确定顶模块，然后逐步细化其调用的下层模块。那么顶模块在哪里？一般可在与主加工（变换中心）相对应的位置上先设计一个模块，作为结构图的顶，其功能是完成整个程序要做的工作。
- 为顶模块设计输入、输出和变换模块，就形成了第一层。
- 每创建一个新的模块必须决定该模块的外部特征：

  该模块的功能：该模块做什么

  该模块同其调用模块的界面：调用时传送的参数
- 对已创建的模块进行细化，考虑该模块怎么做才能完成它的功能，于是又创建下一个模块，再循环到上一步。

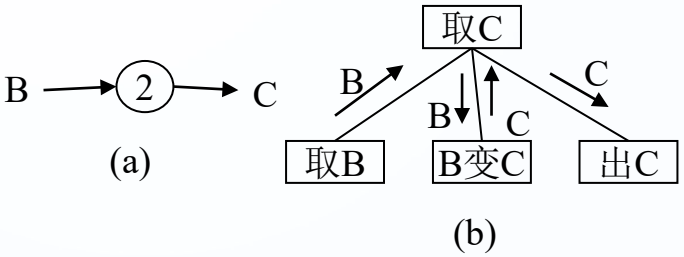
设计模块结构的整个思考过程是按“由顶向下逐步加细”的原则进行。



由主加工起，设计0层和1层

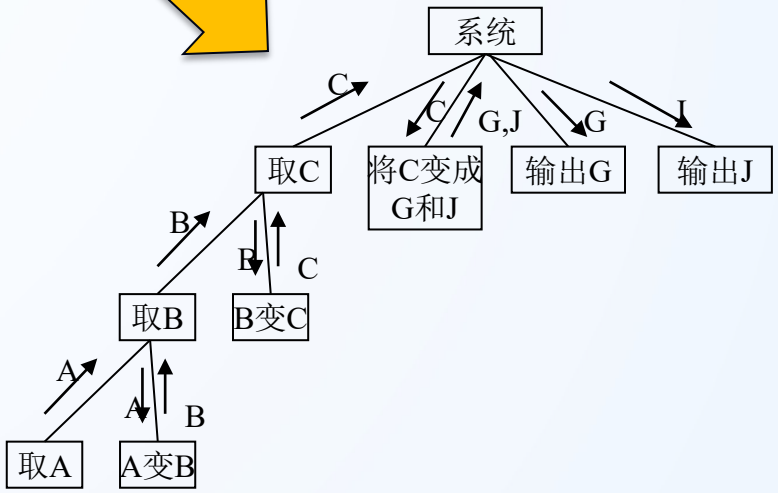
第3步 设计中、下层模块

- 按自上向下逐步细化的原则继续设计每一个模块的下属

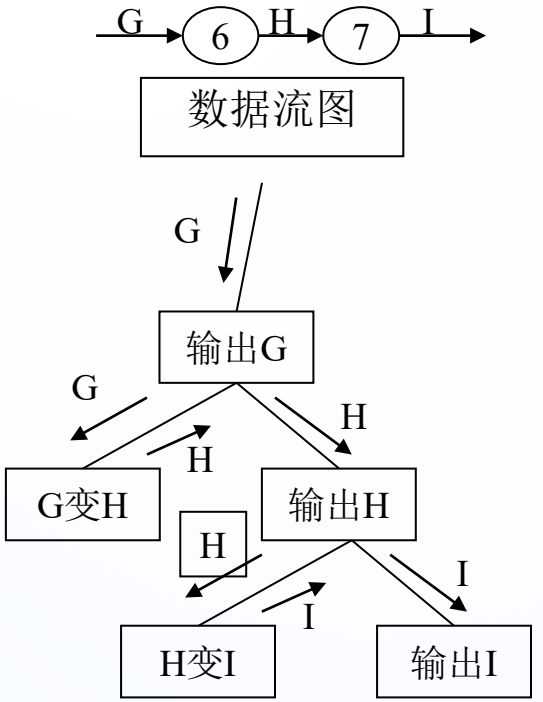


模块细化：

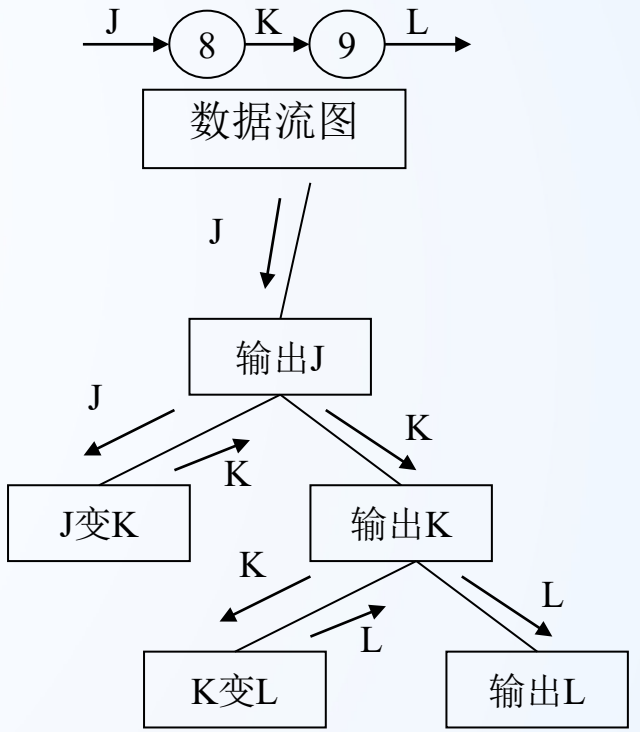
对输入模块 “取C” ,它对应的数据流图a中的如下部分.同样映射为输入模块、变换模块、输出模块,如图b.同理， “取B”模块，细化为图c，综合起来， “取C”模块的结构图综合为图d



“取C的”细化结构图

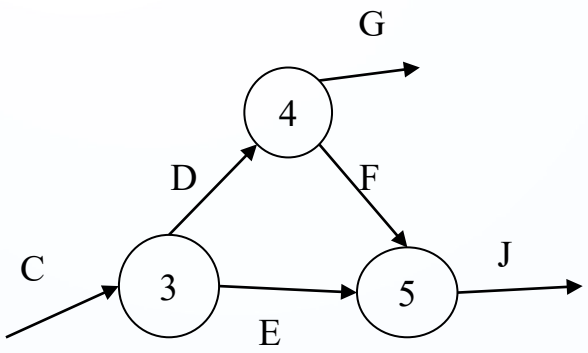


“输出G”模块的细化后的结构图

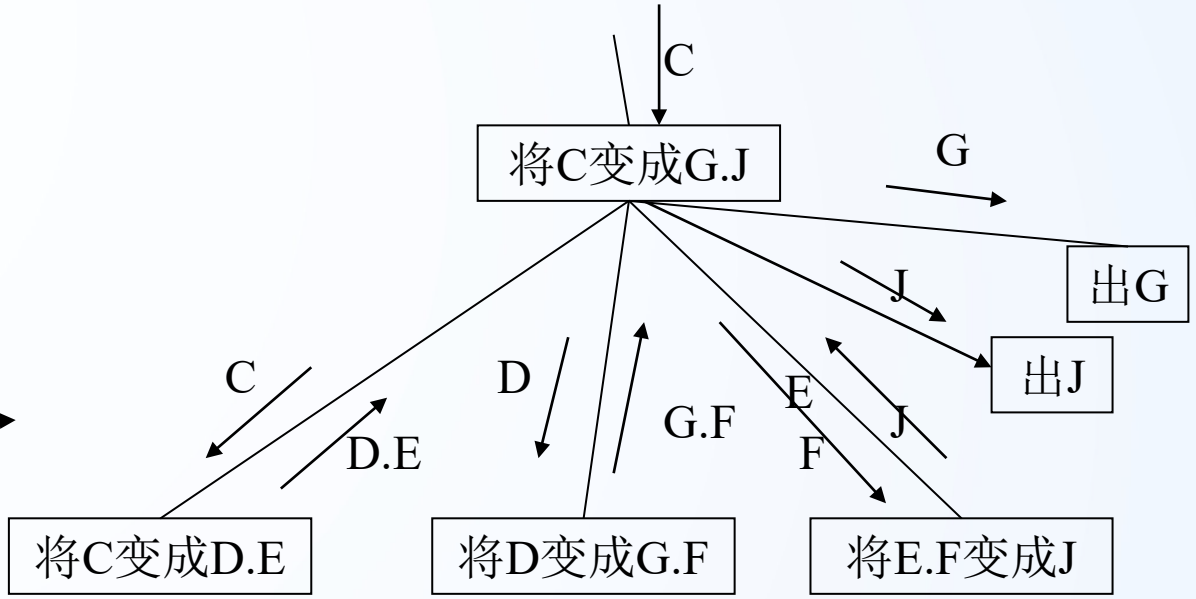


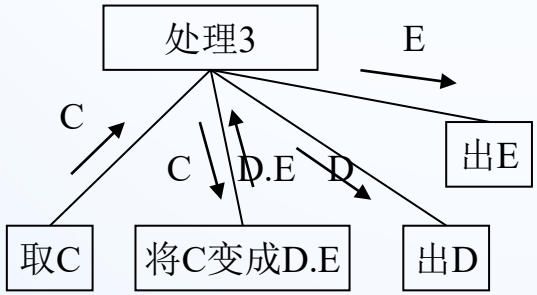
“输出J”模块的细化结构图

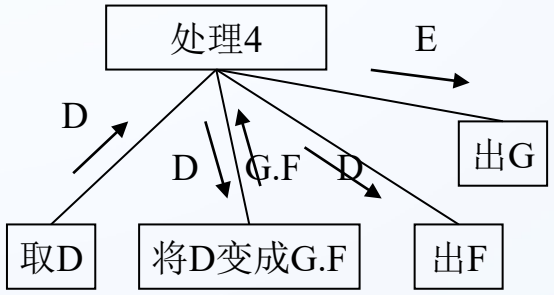
“C变成G.J”模块的细化结构图

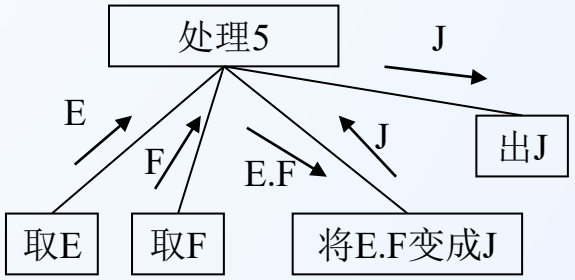


数据流图

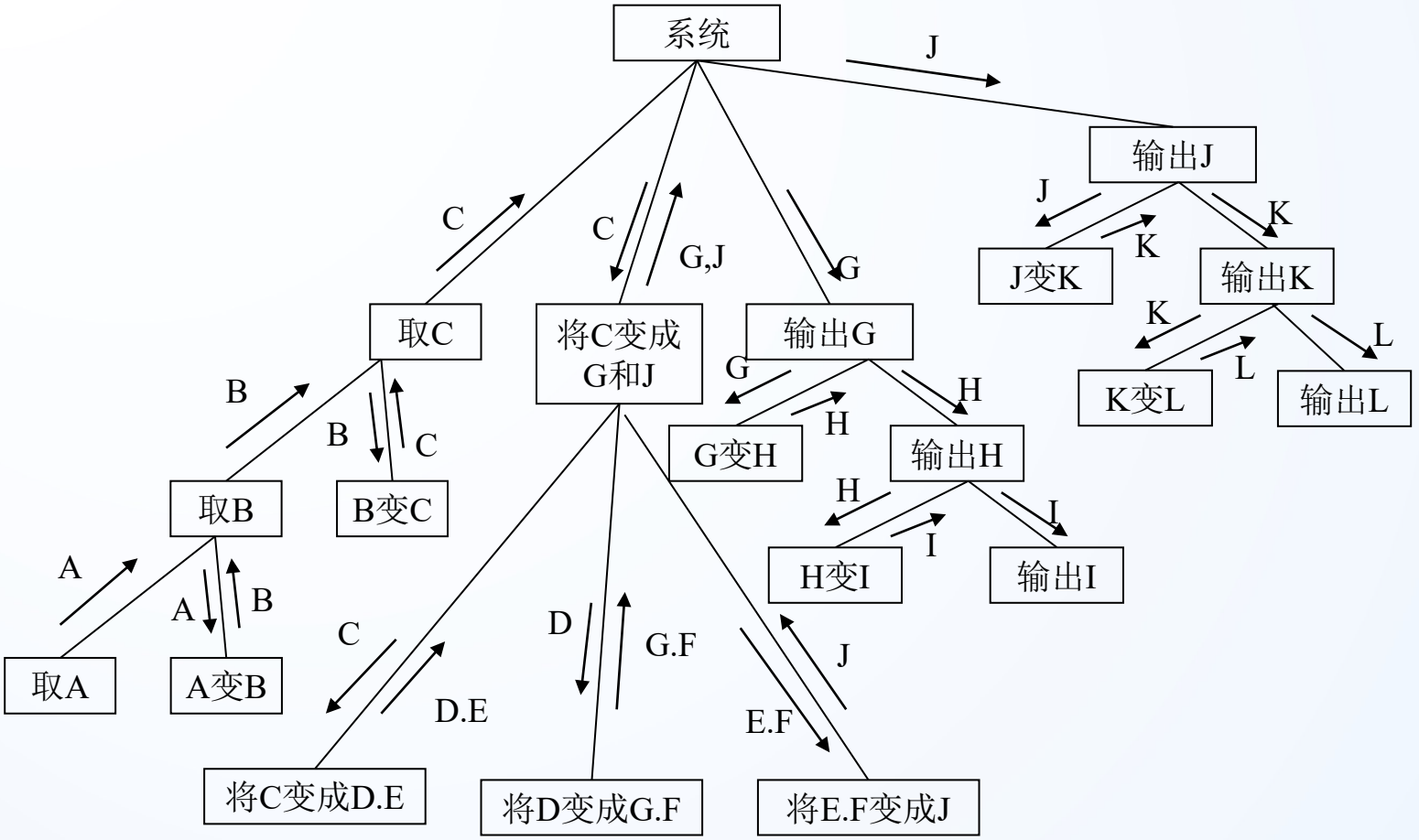




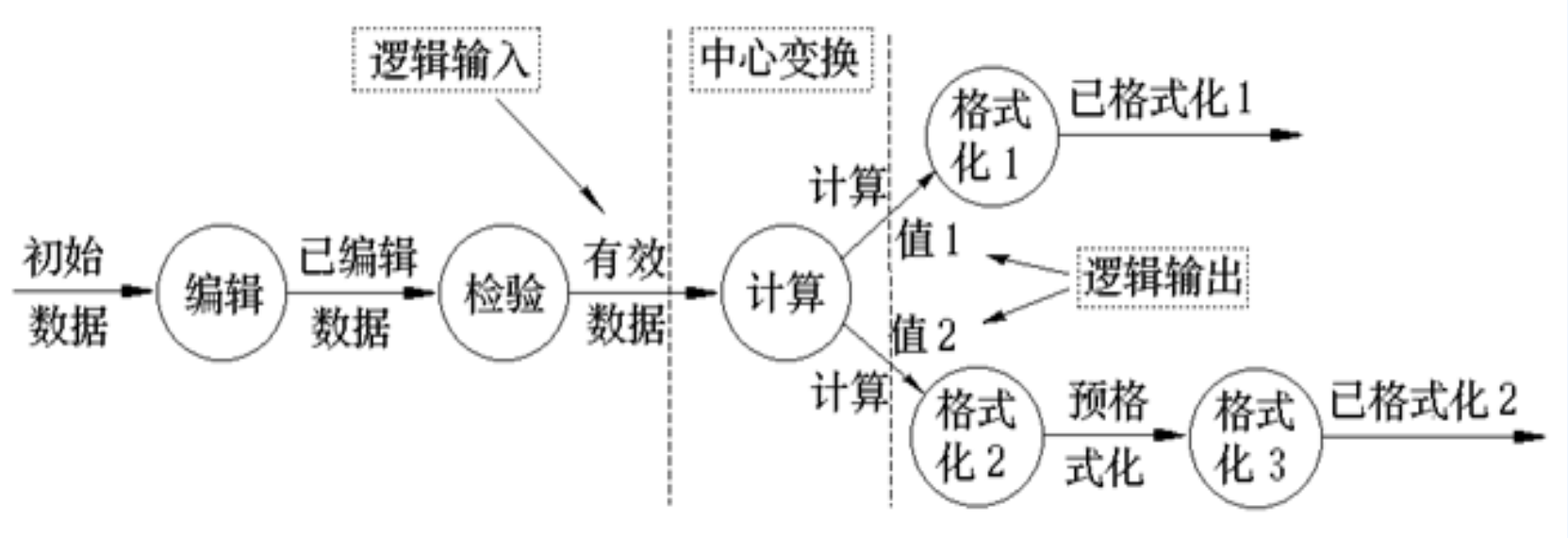




整个系统的结构图

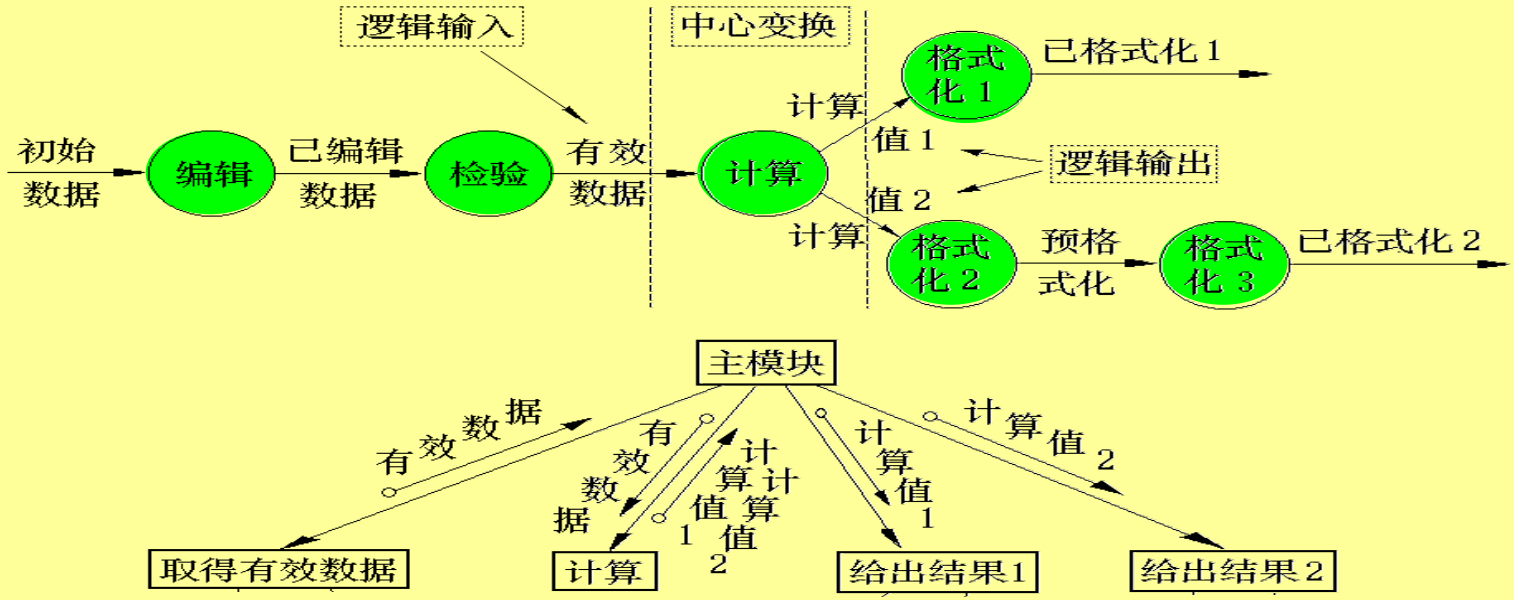

系统
J
输出J
C
C
G,J
G
取C
将C变成
G和J
输出G
J
K
J变K
K
输出K
K
L
K变L
L
输出L
B
B
C
取B
B变C
G
H
G变H
H
输出H
H
I
H变I
I
输出I
A
A
B
取A
A变B
C
D.E
将C变成D.E
D
G.F
将D变成G.F
J
E.F
将E.F变成J

从变换型结构的数据流导出程序结构图示例:



标明数据流的边界

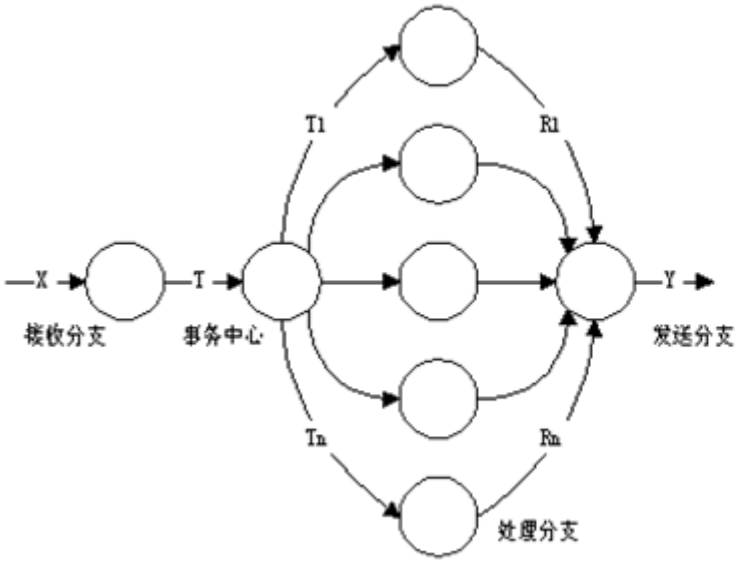
从变换型结构的数据流导出程序结构图示例:

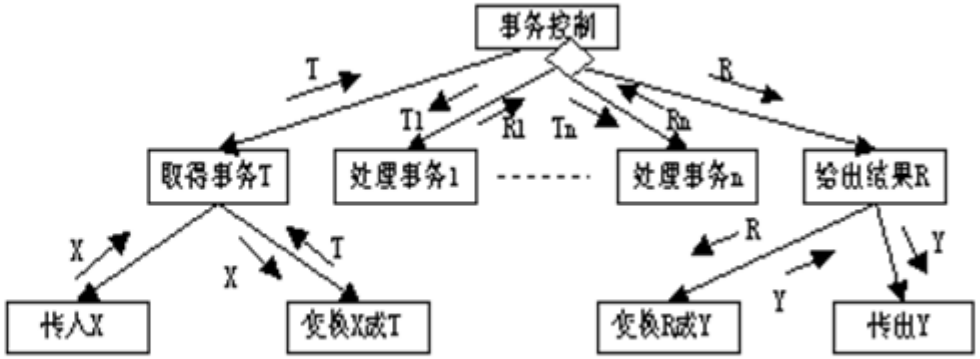


（4）从事务型结构的数据流导出程序结构

从事务型结构的数据流图也可以导出标准形式的程序结构图。

也采用由顶向下逐步细化的方法。先设计主模块，在为每一种类型的事务处理设计一个事务处理模块，然后为每个事务处理模块设计下面的操作模块，在为每个操作模块设计细节模块。某些操作模块和细节模块可以被几个上一层模块共用。





（5）混合型问题

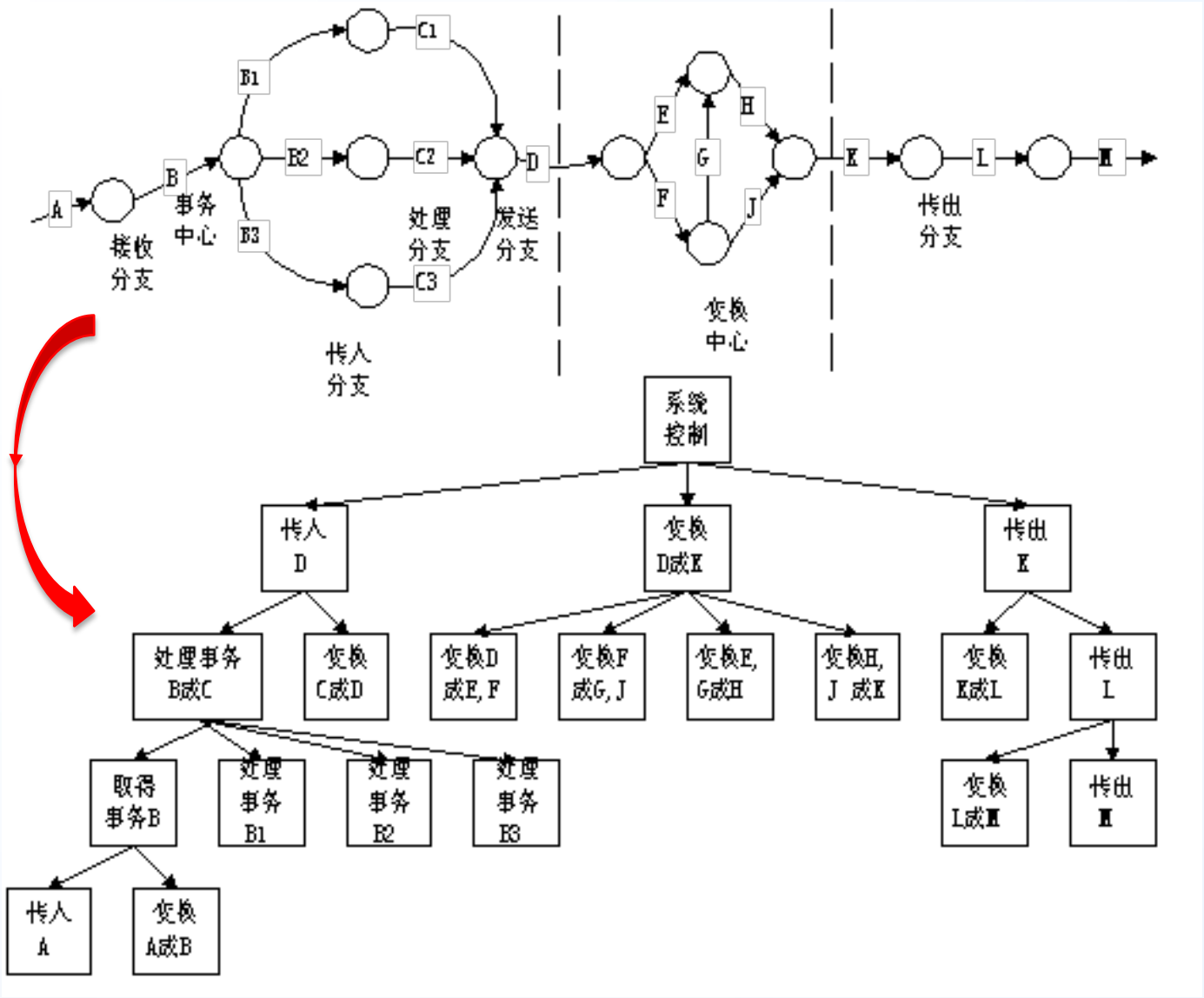
在实际中，一些大型问题往往既不是单纯的变换型问题，也不是单纯的事务型问题，而是两种混合在一起的混合型问题。

对于这种混合型问题，一般以变换型问题为主，

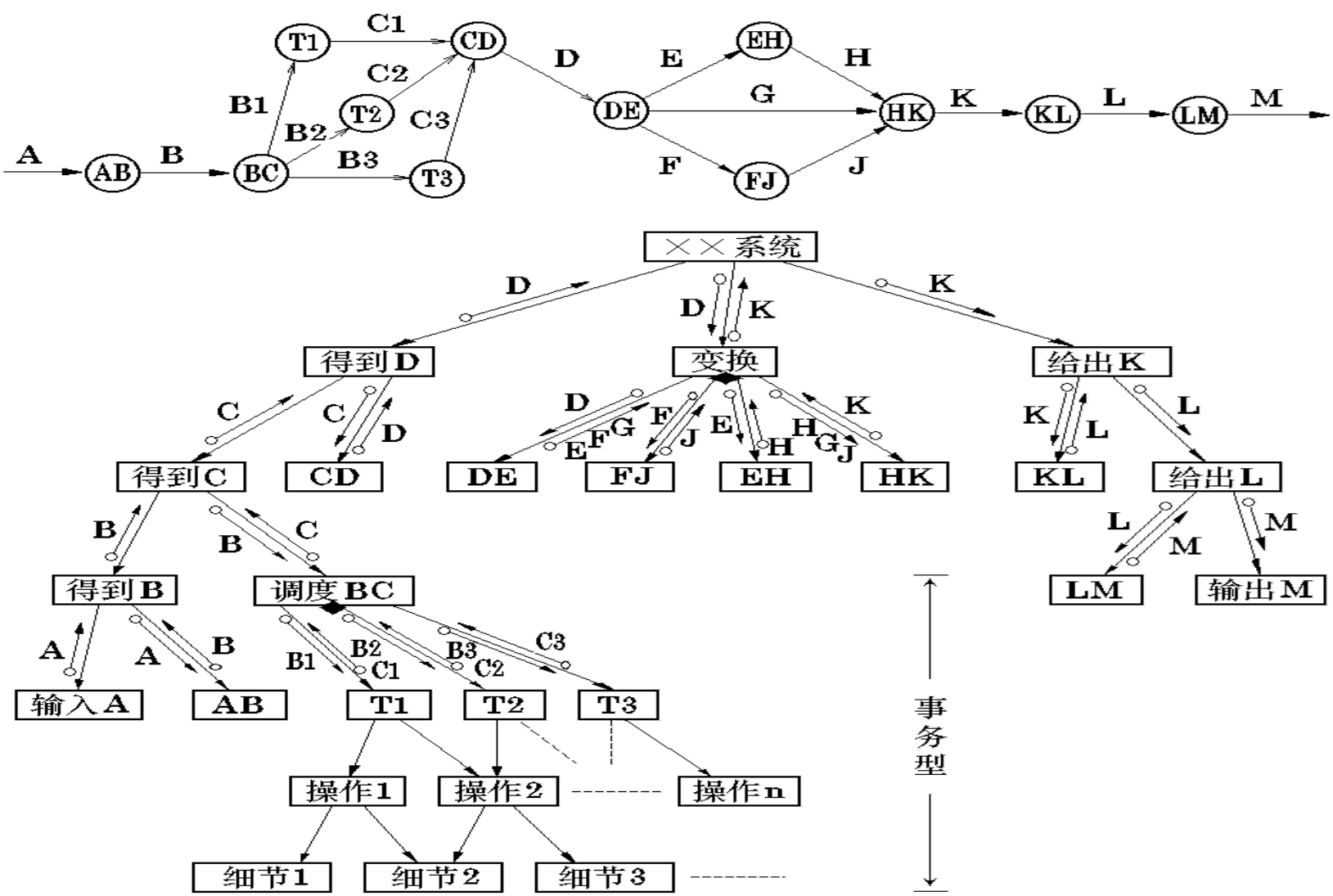
首先找出变换中心，设计出结构图的上层；

然后根据数据流图的各部分具体类型分别映射得到它们的结构图。

从混合型问题数据流导出程序结构图示例1



从混合型问题数据流导出程序结构图示例2



2.3.2 结构图的评价与改进

利用上述结构化设计方法，根据数据流图映射得到的程序结构图，只是程序结构设计的一个初步结果，我们还必须根据一些软件开发原理和设计准则对其进行分析评价和改进。下面结合结构化设计方法的特点，讨论介绍程序结构图的评价和改进原则。

（1）按照结构化设计方法和上述原则评价改进之后，使程序结构具有如下特点：

- 通过模块划分将复杂的问题简单化，使客户需求得到满足，并且每个模块只完成单一的职能 。
- 每个模块都可以独立的进行编码与测试，一个程序员可以完成若干模块，也可以把各模块分配给多个程序员去完成，并行开展工作。
- 把每个模块要解决的问题局限于在一定范围内，处理一个模块问题时不必考虑模块以外的问题，减少了出错的机会。即使出现了错误，在局部范围内也容易解决。
- 使模块中部分程序的修改完全不影响模块以外的程序。减少模块内部程序修改所产生的副作用，即使程序员个人的工作出现差错，所影响的范围也只限在模块以内，不会影响到全局。
- 对关键模块采取特殊措施加以优化处理，以保证整个系统达到特定的要求。
- 使程序的复用成为可能，模块可多次使用，提高了软件产品的利用率。
- 已开发的程序易于理解，每个模块的职能明确，也就不难理解整个软件系统的结 构和性能。
- 有利于估计工作量和开发成本。

(2) 程序结构图的改进措施

1）分析评价程序结构的耦合性和内聚性

2）分析评价程序图的深度和宽度、扇入、扇出，改善程序结构的形态特征

3）分析模块的作用域和控制域，使作用域在控制域之内

4）拟定模块的接口，降低接口的复杂性和冗余度，改善一致性

模块接口的复杂性是程序出错的一个主要原因，接口的设计应尽量使信息传递简单，并与模块功能一致。

5）分析模块的功能，保证模块的可验证性

可验证性是软件开发要遵循的基本原理之一，因此模块必须设计成为能可预测的。当一个模块可以看作成一个黑盒，不管其内部的处理细节如何，都会产生同样的外部数据时，这种模块是可以预测的。

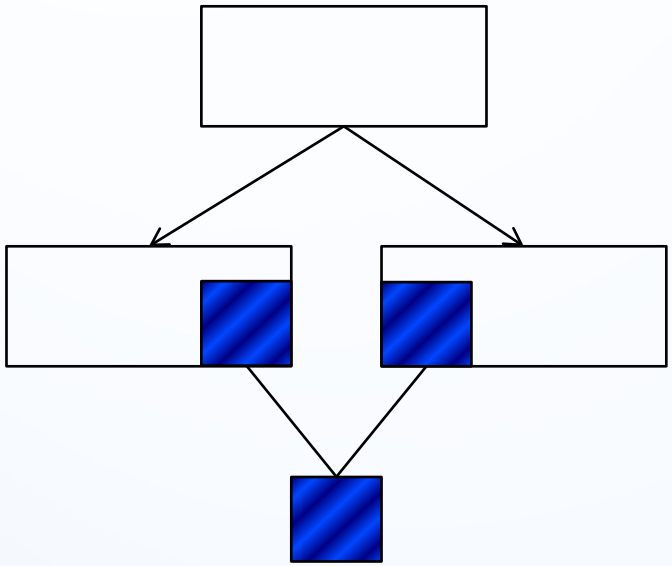
6）恰当地掌握划分模块的大小

模块的大小一般指其篇幅，即源程序的行数。篇幅过大的模块，其复杂性也大，会给理解带来困难，所以要适当控制模块的大小。

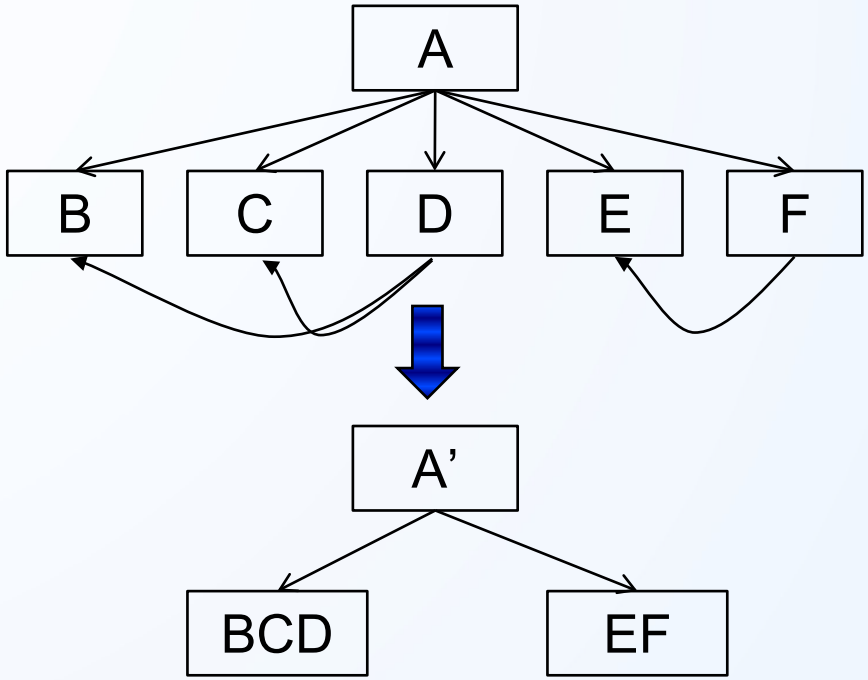
究竟划分多大的模块最为合理，很难给出绝对的标准。但一般认为，程序最好能够写在一页纸内，或者说程序行数在50-100的范围内是比较合适的。同一个问题，如果把模块划得很小，势必增加模块的数量，增加了模块接口的复杂性，也增加了调用和返回的时间开销，降低效率。如果把模块划得过大，将会造成测试和维护工作的困难。

措施1：程序内聚性和耦合性判定和修正

耦合是影响软件复杂程度的一个重要因素，在软件设计过程中，应尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合；一般认为，偶然的、逻辑的和时间的内聚是低内聚性的表现，信息的内聚则属于中等内聚性，顺序的和功能的内聚是高内聚性的表现。为提高模块独立性，可对模块进行分解和合并，以改善模块的内聚性和耦合性。



（a）从多个模块中分解出公共子功能定义一个新的内聚性较高的模块



（b）合并具有较高耦合性的模块，并降低接口的复杂性

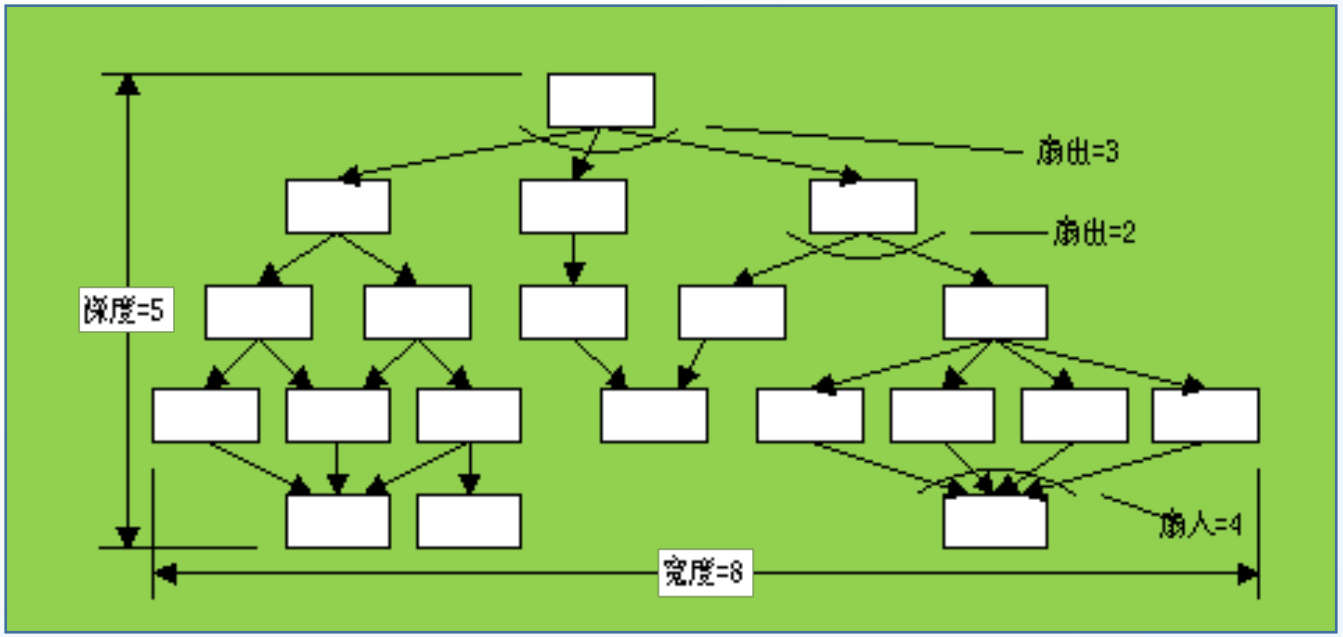
措施2：程序图的深度和宽度、扇入、扇出

模块的层次数称为结构图的深度，它表示出了控制的层数

结构图中同一层次模块的最大模块个数称为结构的跨度（宽度），它表示出了控制的总分布

一个模块直接控制的下属模块的个数称为该模块的扇出数

一个模块可以有同一个下属模块，该下属模块的上级模块的个数称为扇入数。



结构图的深度在一定意义上也能反映出程序物理结构的规模和复杂程度。对于中等规模的程序，其深度约为10左右。如果深度过大，则应考虑结构中的某些模块是否过分简单了。

一般来说，宽度越大系统越复杂。

好的系统的平均扇出数通常是3~4，最多是5~9。扇出过大时可以适当增加中间层次的控制模块；扇出过小时可以把下级模块进一步分解成若干个子功能模块，或者合并到上级模块中去。当然这种分解或合并不能影响模块的独立性。

通常设计得好的程序结构具有橄榄型结构形态，如图所示。在这种结构中，顶层扇 出比较高，中层扇出比较少，底层（共用模块）扇入比较高。

2.3.3 模块说明

完成程序结构的设计和改进工作之后，还必须对每个模块进行具体定义和说明，主要内容包括如下几个方面。

1)功能说明： 描述模块的主要任务

2)接口说明：

列出通过参数表传递的数据、外界的输入输出、从全程数据区取得的数据项、上级模块名和下级模块名

3)局部的和全局的数据结构说明，重点对全局数据进行说明

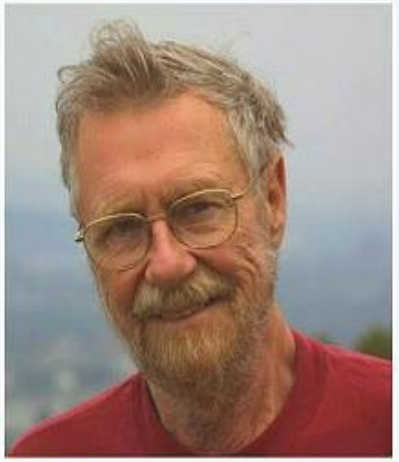
4)所有设计约束和限制说明，包括:

数据类型或数据格式的限制、内存容量或时间的限制、数据结构的边界值、没有考虑的特定情况以及某个模块的专门特性，等等。

03 结构化程序设计

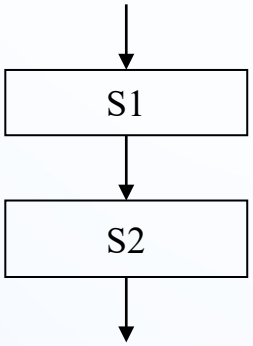
03 结构化程序设计

结构化程序设计(Structured Programming ，SP)方法由E.Dijkstra等人于1972年提出，用于详细设计阶段和编程实现阶段指导人们用良好的思想方法开发易于理解又正确的程序。

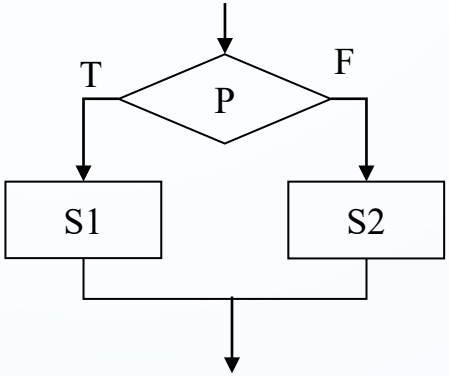


结构化程序设计方法建立在结构定理基础之上，认为：

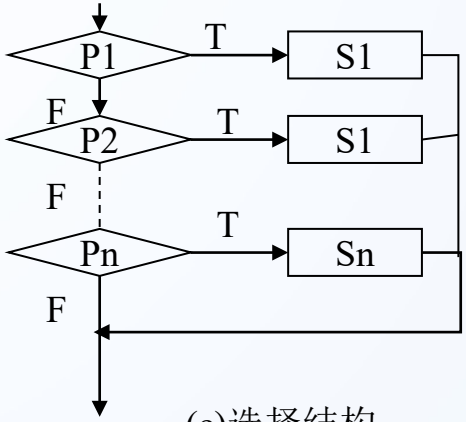
任何程序逻辑都可以用顺序、选择和循环等三种基本控制结构来表示。



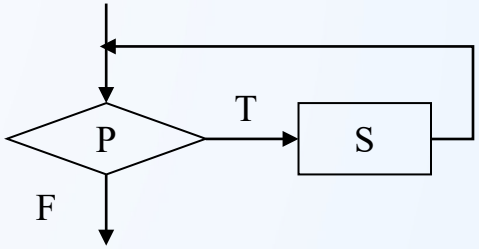
(a)顺序结构



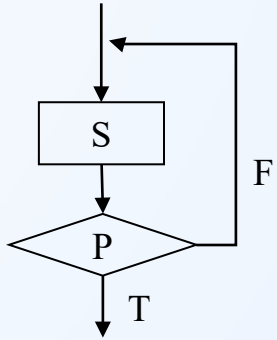
(b)选择结构--条件结构



(c)选择结构



(d)循环结构—当循环



(e)循环型—直到循环

(1)顺序结构（顺序型） 含有连续加工程序段等。 如赋值、读、写语句

(2)选择结构（选择型） 由条件取值决定选择两个或多个加工中的一个的程序段。

(3)循环结构（迭代型） 在控制条件满足的情况下，重复执行特定的加工。

在结构定理的基础上，Dijkstra主张避免使用GOTO语句，而仅仅用上述三种结构反复嵌套来构造程序。在这一思想指导下，一个模块的详细执行过程可按“由顶向下逐步加细”的方式确定。

◆ **“由顶向下逐步加细”方法**

一开始，模块的执行过程是未确定的，亦即模糊不清的，我们可以用下面三种方式对模糊过程进行分解，使不清楚的部分逐步清晰起来。

1）用顺序方式对过程作分解，确定模糊过程中各个部分的时间顺序。

2）用选择方式对过程作分解，确定选择模糊过程中某个部分的条件。

3）用循环方式对过程做分解，确定模糊过程的主体部分进行重复的开始和结束条件。

4）对仍然模糊的部分反复使用这三种分解方法，最终将所有的细节全部确定下来。

◆程序过程表示方法

程序的过程设计是软件详细设计中的重要内容，常用的程序过程表示方法有：

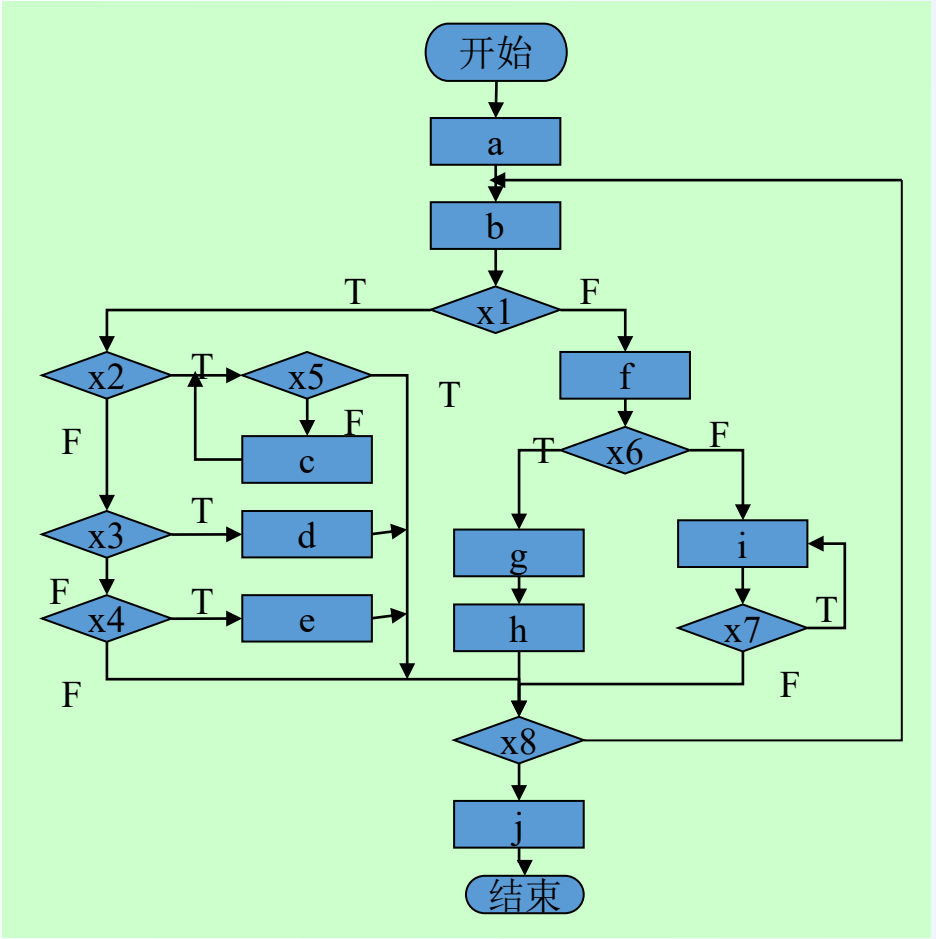
- ✓程序流程图(FC，Flow Chart)
- ✓盒图(N-S图)(N，S两发明人的名字首字母)
- ✓问题分析图(PAD， Program Problem Analysis Diagram)
- ✓程序设计语言(PDL， Program Design Language)—伪码

（1）程序流程图

流程图是用图形方法描述一个模块的内部执行过程的最常用、使用历史最悠久的方法。

流程图包含三种基本成分：

- ✓加工步骤—用方框表示
- ✓逻辑条件—用菱形框表示
- ✓控制流—用箭头表示

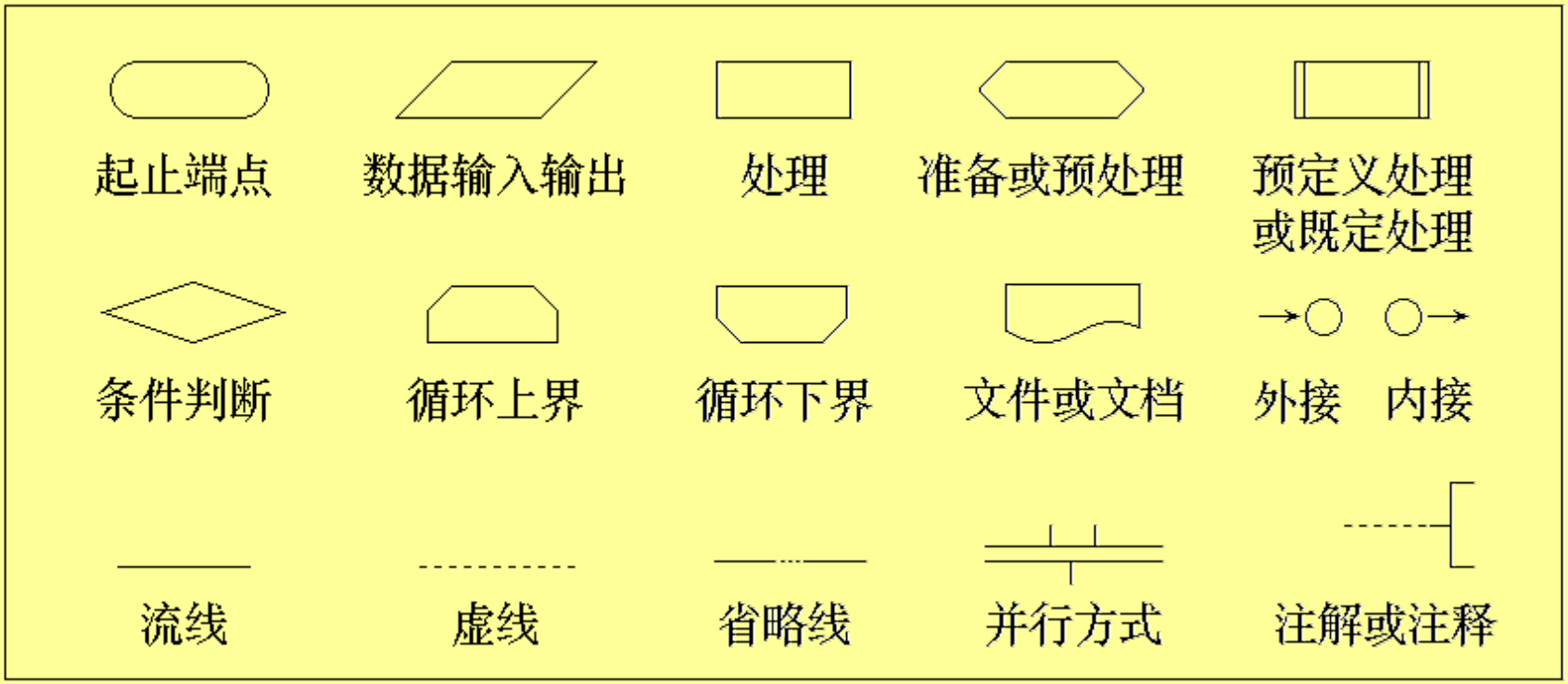


优点： 将SA，SD，SP衔接起来，用数据流图描述用户需求，用结构图描述软件结构，用流程图描述摸块的执行过程是常见的软件描述方法。

仅使用三种标准结构反复嵌套来绘制流程图，称为结构化流程图，它的结构清晰易于理解又易于修改。

缺点： 流程图并不强制设计人员用结构化的方法进行详细设计，设计人员用箭头可以实现向任意位置转移，如使用正确，流程图是简单易懂，但如使用不当，流程图可能非常难懂，而且无法进行维护。所以箭头是一个隐患，使用时必须小心谨慎，流程图的质量取决于设计人员的水平。流程图只能描述执行过程，不能描述有关的数据。

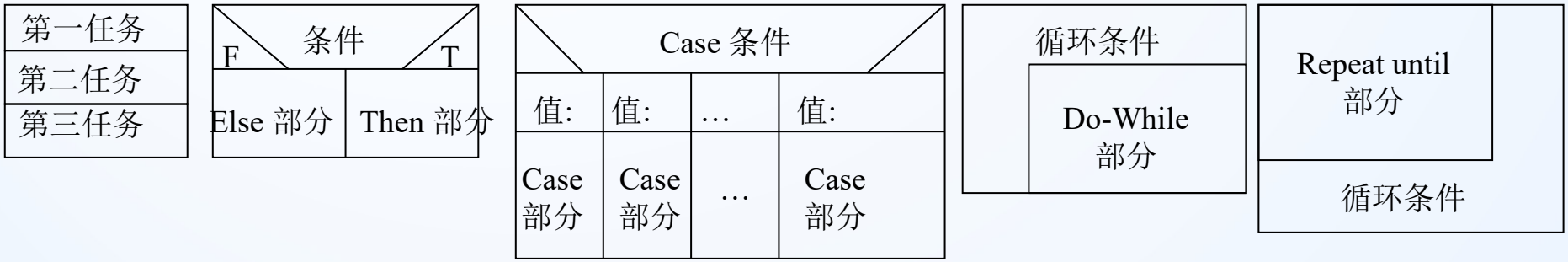
浙江大学 机械工程学院 航空制造工程研究所 王 青



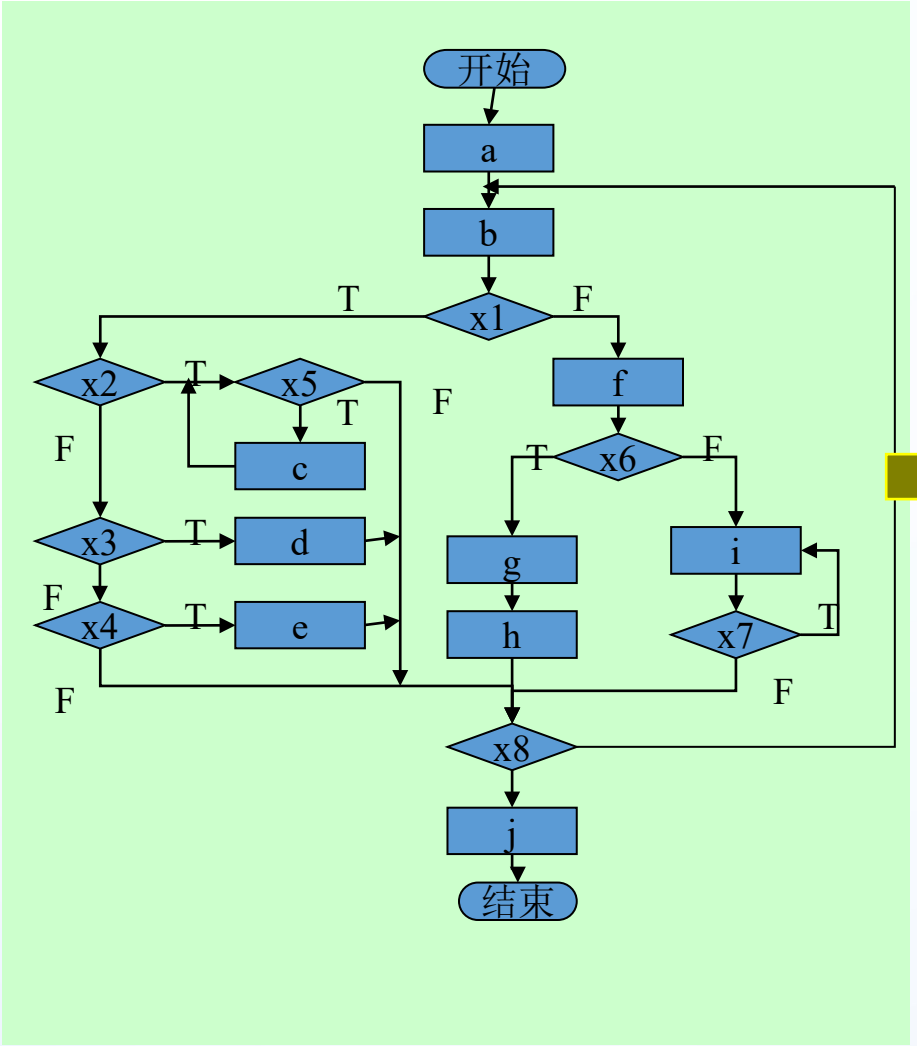
国家标准局批准的国家标准(GB1525-89)

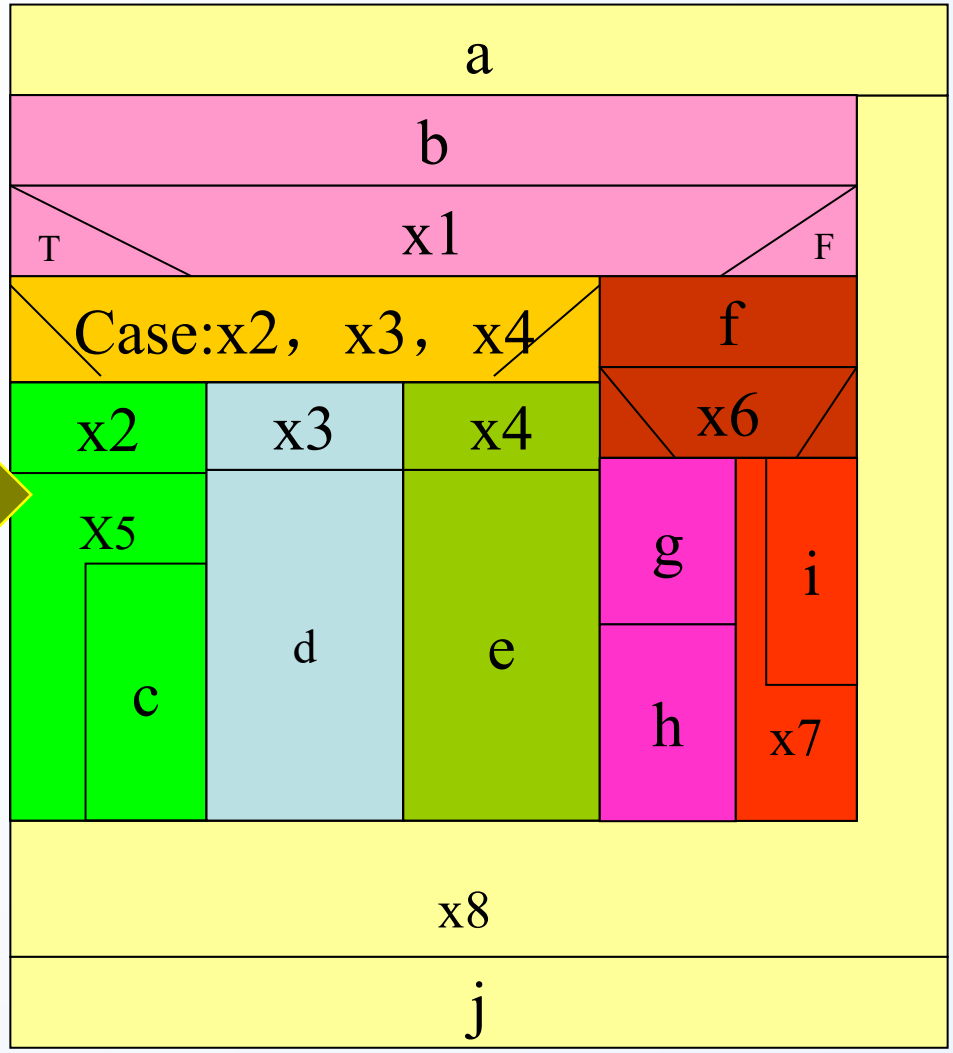
(2) 盒图(NS图)

- NS图是Nassi和Shneiderman于1973年提出的用于取代传统流程图的一种程序过程描述方法。
- 在NS图中，每个“处理步骤”用一个盒子表示；
- 盒子可以还可以嵌套另一个盒子，嵌套深度没有限制；
- NS中仅含5种基本成分，分别表示SP方法的几种标准控制结构。
- NS中常用两个盒子，数据盒和过程盒：
  - ✓数据盒描述有关数据，包括全程数据、局部数据和模块界面上的参数；
  - ✓过程盒描述执行过程；



绘制盒图:


开始
a
b
x1
T
F
x2
x5
c
x3
d
x4
e
f
x6
g
h
i
x7
x8
j
结束


a
b
x1
T
F
Case:x2，x3，x4
f
x6
x2
x3
x4
X5
c
d
e
g
h
i
x7
x8
j

NS图的优点：

- ✓强制设计人员按SP方法进行思考并描述设计方案，因为除表示几种标准结构符号外，不再提供其他描述手段，有效地保证了设计的质量。
- ✓NS图形直观，具有良好的可见度，例如循环的范围、条件语句的范围都一目了然，所以容易理解设计意图，为编程、复查、选择测试用例、维护都带来了方便。
- ✓NS图简单、易学易用，可用于软件教育和其他方面。

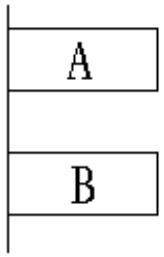
NS图的缺点：

- ✓ 手工修改比较麻烦。

(3)问题分析图（PAD)

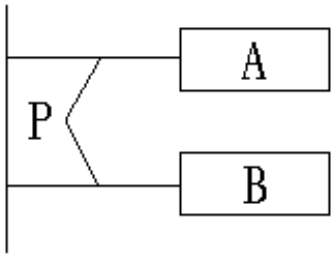
- 问题分析图是由日本日立公司的二村良彦等人于1979年提出，用于取代流程图的程序过程图形化描述方法。
- 特点：

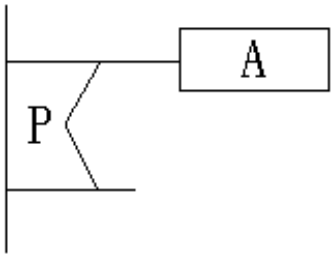
  它把程序过程控制结构表示成二维树的图形；
- PAD支持SP方法，它只有顺序、选择、循环三类基本成分，其中选择和循环又有几种不同形式。



按顺序先执行A，再执行B

① 顺序型



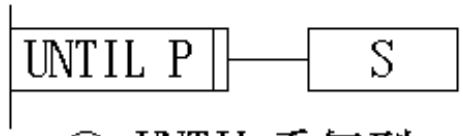


条件P成立时执行A，否则执行B

② 选择型

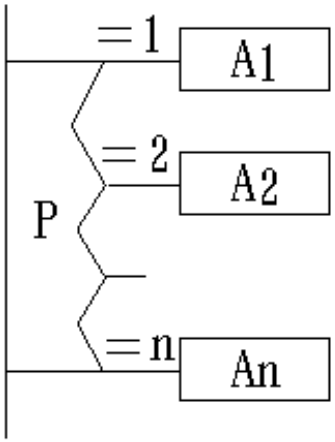
③ WHILE重复型

P为循环条件，S为循环体，P成立时执行S



④ UNTIL重复型

P为循环条件，S为循环体，执行S，直至P成立为止



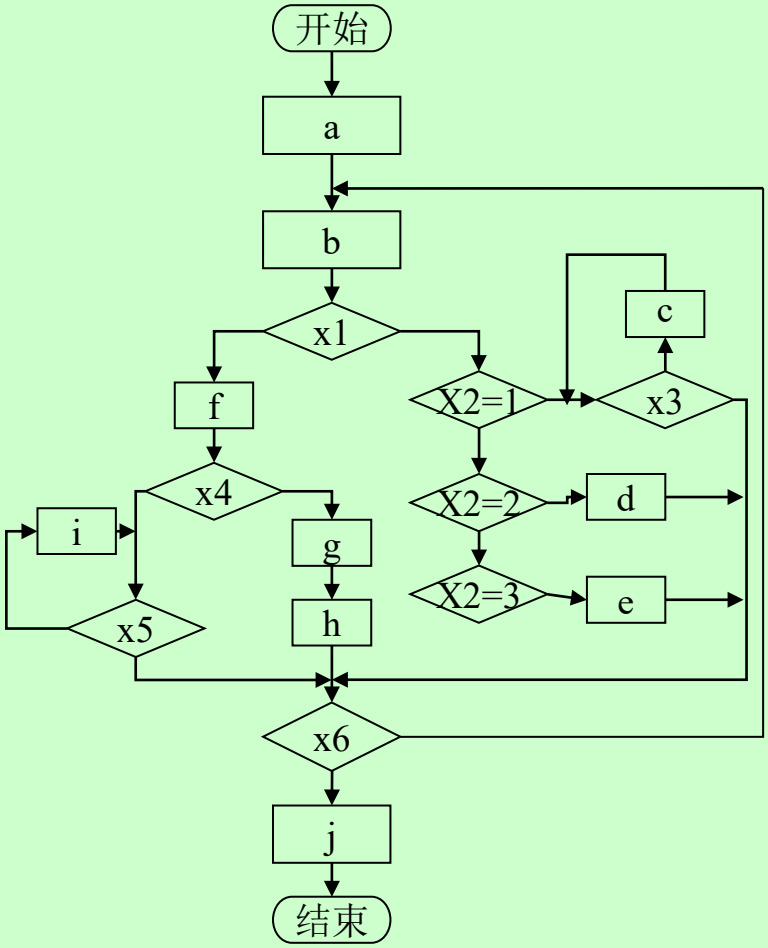
⑤ 多分支选择型（CASE型）

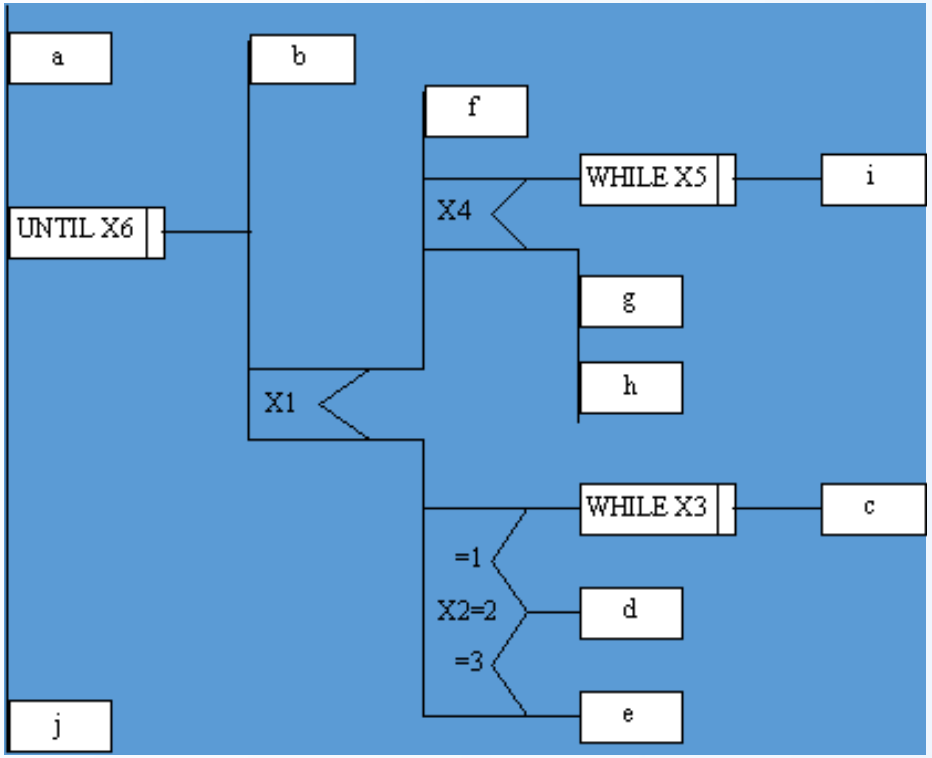
按P的取值，执行相应的A1，A2，A3等框的内容。

(3)问题分析图（PAD)

利用问题分析图进行程序过程设计具有如下优点：

- 设计出的程序必定是结构化程序。
- 所表达的程序，其结构非常清晰。图中最左线是程序的主中线，也就是程序过程中控制的第一层结构，随着层次的增加，图形逐渐向右扩展。每增加一个层次，图形向右扩展一道纵线，因此程序中含有的层次数即为问题分析图中的纵线数。
- 作为一种详细设计的图形工具，问题分析图比流程图更容易读。可以把每个问题分析图看作为一个树形结构，程序的执行顺序为从最左主干线上端的结点开始，自上而下，自左而右遍历所有结点。
- 由于问题分析图的树形特点，使我们比流程图更容易在计算机上进行处理。例如，在开发问题分析图向高级语言转换的程序以后，便可从终端输入问题分析图，并自动转换成高级语言。


开始
a
b
x1
f
x4
i
x5
g
h
X2=1
X2=2
X2=3
c
x3
d
e
x6
j
结束


a
b
f
WHILE X5
i
X4
UNTIL X6
g
h
X1
WHILE X3
c
=1
X2=2
d
=3
e
j

(4)程序设计语言-PDL

- 详细设计的结果也可以用语言来描述；
- PDL是一种典型的描述程序过程的语言，PDL是一种非形式化的比较灵活的语言；
- PDL用于描述模块内部的具体算法；
- PDL是开放的；
  - 其外层语法是确定的，用于描述程序的控制结构，如条件、顺序、循环
    - ✓条件：IF-THEN-ELSE
    - ✓循环:WHILE-DO，REPEAT-UNTIL
  - 内层语法则故意不确定，用于描述具体的操作。

程序设计语言-PDL 例1

```
if X is not negative
    then
        return ( square root of X as a real number);
    else
        return ( square root of –x as a real number);
```

外层语法—if …then…else

内层：

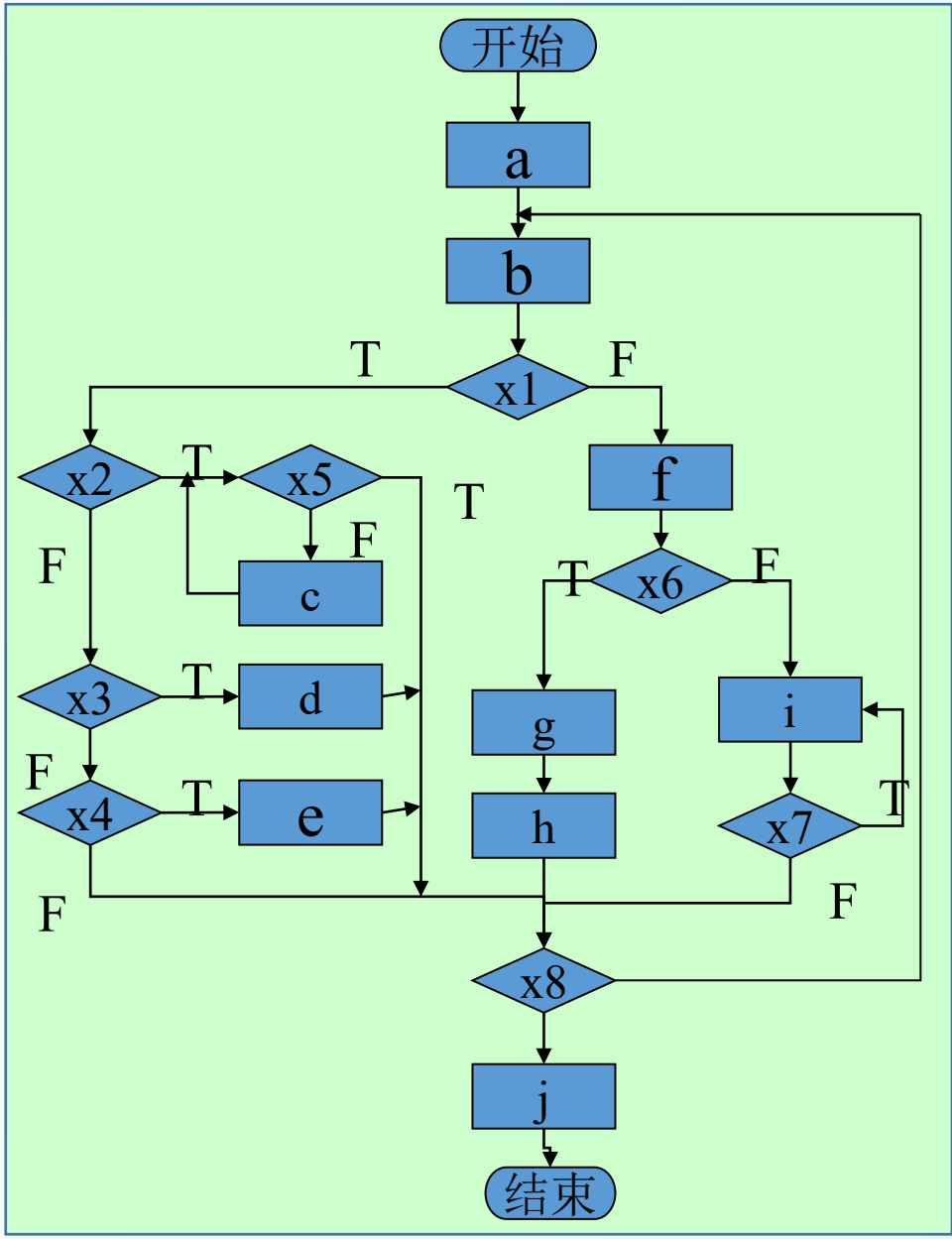
```
X is not negative
square root of X as a real number
square root of –x as a real number
```

程序设计语言-PDL例2：

```
PROCEDURE FIND(string，char，len，occurrence，positions)
   this procedure finds the number of occurrences of a character， char， in an alphanumeric string and stores a 1 in the corresponding bit position of positions array.
DECLARE
  string(len) AS STRING ARG
   char AS CHAR ARG
   len AS SCALAR ARG
   occurrence AS SCALAR ARG
   positions AS BIT ARRAY ARG
 index =0
Set position to 0
Occurrence =0

DO WHILE string(index)!=end-of-string
                  IF string(index)=char
      THEN BEGIN
                occurrence = occurrence+1
                set position(index) to 1
                 END
       ELSE skip
    ENDIF
    increment index
ENDDO
RETURN
END FIND
```

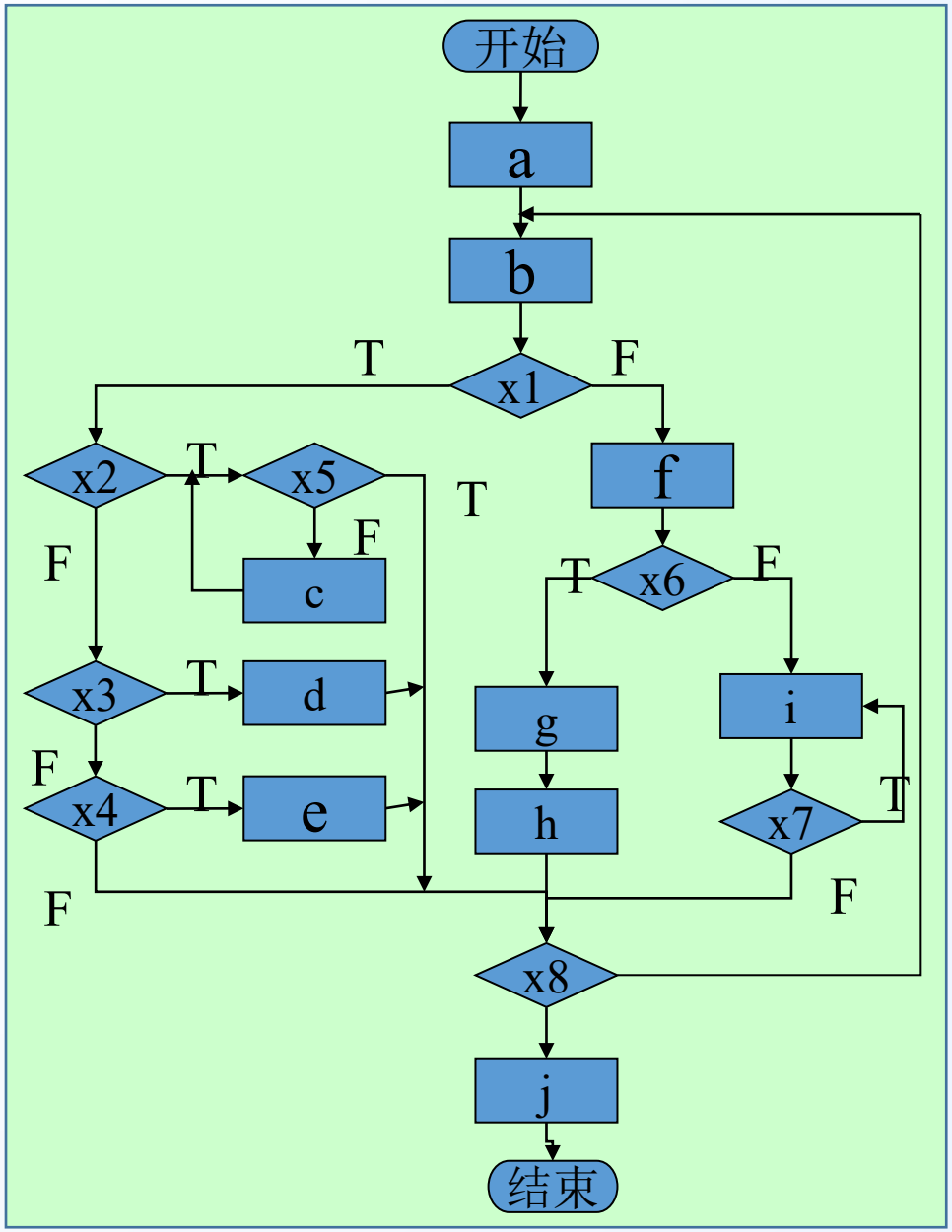
```
BEGIN
execute process a
REPEAT UNTIL condition x8
    execute process b
    IF condition x1  THEN
        CASE OF xi
            WHEN condition x2 SELECT
                DO WHILE condition x5
                    execute c
                ENDDO
            WHEN condition x3 SELECT
                execute d
            WHEN condition x4 SELECT
                execute e
        END CASE
    ELSE BEGIN
        execute process f
```



浙江大学 ZHEJIANG UNIVERSITY

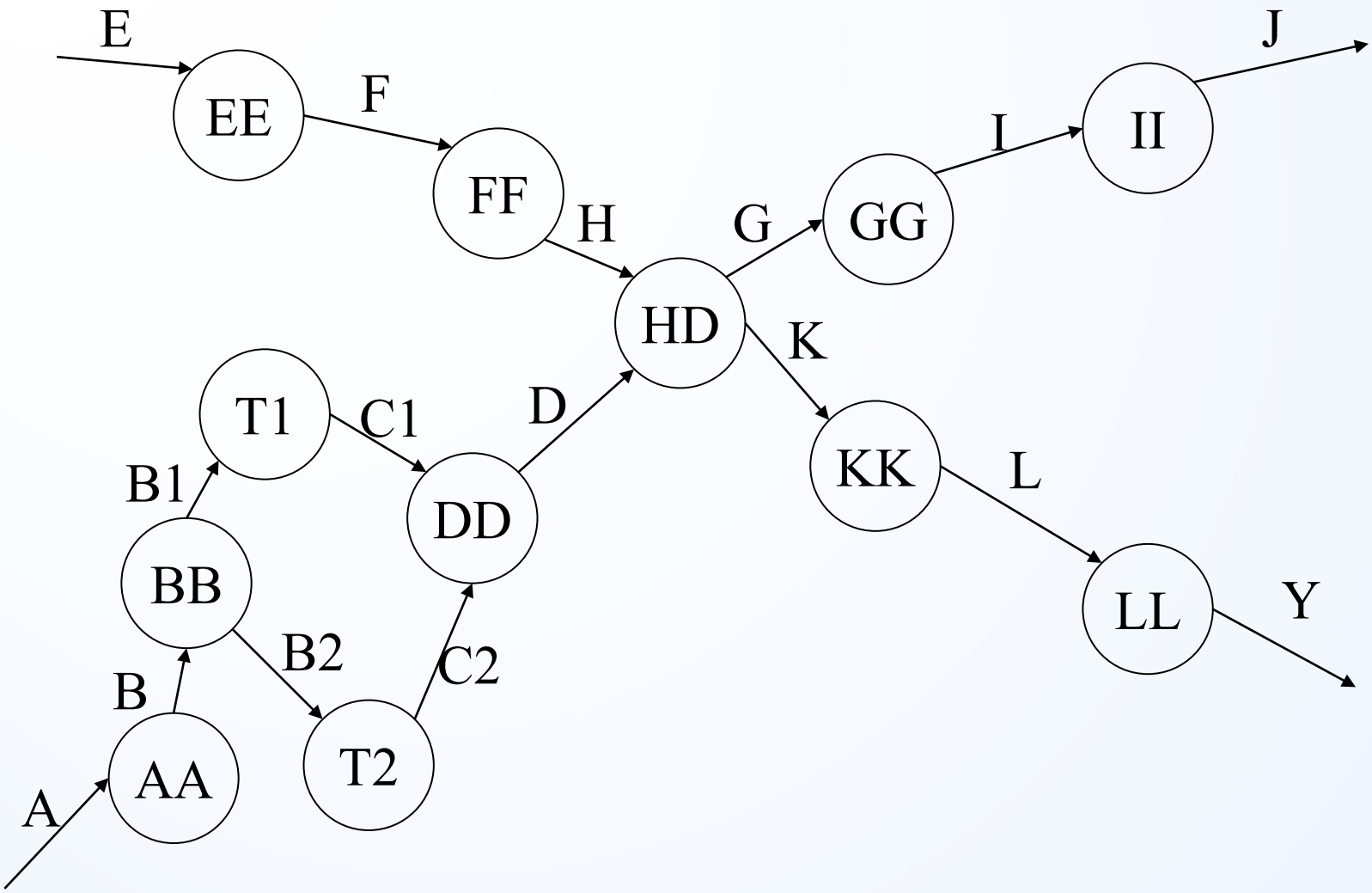
程序设计语言-PDL例3

```
IF condition x6 THEN
        BEGIN
            execute g
            execute h
        END
ELSE
        REPEAT UNTIL condition x7
            execute process i
        END REPEAT
ENDIF
ENDIF
END REPEAT
Execute j
END
```



作业

（1）已知有一抽象的数据流图如下所示，用SD方法画出其相应的结构图。



作业

(2) 在上次作业中的数据流图的基础上画出其程序结构图。

(3) 画出与下列用伪码书写的程序对应的程序流程图、NS图和PAD图。

```
k←n; FLAG←1
WHILE FLAG > 0 DO
  k←k-1
  FLAG←0
  FOR J←1 TO k DO
      IF L(J)>L(J+1) THEN DO
            L(J) = L(J+1)
            FLAG←1
       ENDDO  ENDIF
  END FOR
END WHILE
```

THANK YOU

浙江大学机械工程学院航空制造工程研究所
王 青 李江雄