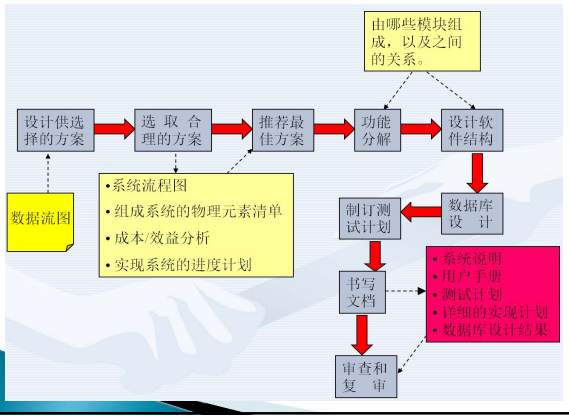
软件工程

第7-1章 结构化设计

- 结构化设计概述
- 结构设计—概要设计
- 过程设计—详细设计

@3.2节.教材

面向数据流的软件设计过程：



结构化设计 (Structured Design, SD)

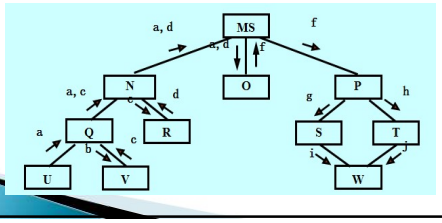
- 结构化设计是将结构化分析得到的数据流图映射成软件结构的一种设计方法
- 强调模块化、自顶向下逐步求精、信息隐蔽、高内聚低耦合等设计准则

结构化设计的内容

- 结构设计—概要设计
  - 结构图(Structure Chart)
  - 物理数据模型
- 过程设计—详细设计
  - 模块的处理过程
    - N-S图，PAD，PDL等

结构设计的工具——结构图

- 用结构图(Structure Chart)来描述软件系统的体系结构
- 描述一个软件系统由哪些模块组成，以及模块之间的调用关系
- 结构图的基本成分有：模块、调用和数据

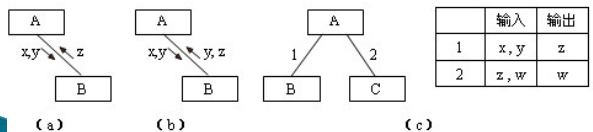


模块(module)

- 模块是指具有一定功能的可以用模块名调用的一组程序语句，包括函数和子程序。
- 一个模块具有其外部特征和内部特征
  - 外部特征包括：模块的接口(模块名、输入/输出参数、返回值等)和模块的功能
  - 内部特征包括：模块的内部数据和完成其功能的程序代码

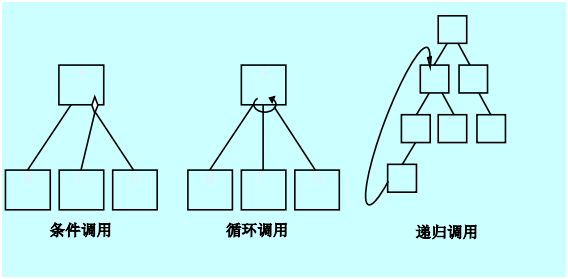
调用和数据

- 调用(call)：用从一个模块指向另一个模块的箭头来表示，其含义是前者调用了后者
  - 为了方便，有时常用直线替代箭头，此时，表示位于上方的模块调用位于下方的模块
- 数据(data)：模块调用时需传递的参数可通过在调用箭头旁附加一个小箭头和数据名来表示



	输入	输出
1	x, y	z
2	z, w	w

结构图中的辅助符号

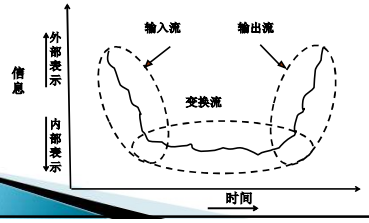


数据流图到结构图的映射

- 结构化设计是将结构化分析的结果(数据流图)映射成软件的体系结构(结构图)
- 将数据流图分为变换型数据流图和事务型数据流图，对应的映射分别称为变换分析和事务分析

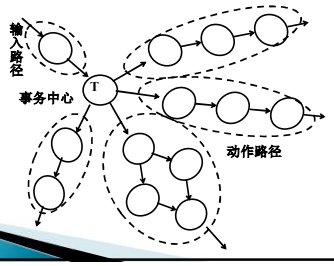
变换流

- 特征：数据流图可明显地分成三部分
  - 输入：信息沿着输入路径进入系统，并将输入信息的外部形式经过编辑、格式转换、合法性检查、预处理等辅助性加工后变成内部形式
  - 变换：内部形式的信息由变换中心进行处理
  - 输出：然后沿着输出路径经过格式转换、组成物理块、缓冲处理等辅助性加工后变成输出信息送到系统外



事务流

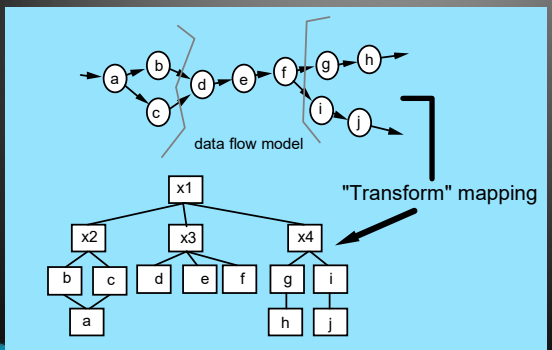
- 特征：数据流沿着输入路径到达一个事务中心，事务中心根据输入数据的类型在若干条动作路径中选择一条来执行
- 事务中心的任务是：接收输入数据(即事务)；分析每个事务的类型；根据事务类型选择执行一条动作路径



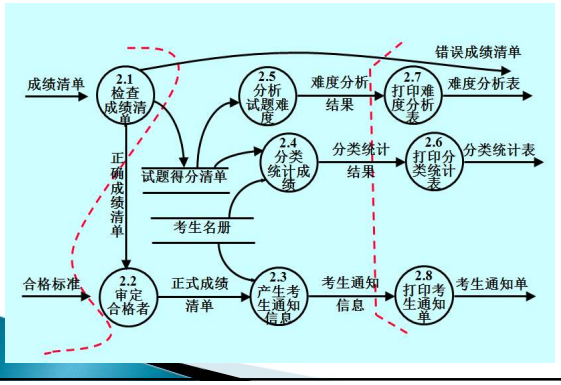
变换分析

- 变换分析的任务是将变换型的DFD映射成初始的结构图，步骤如下：
  - 划定输入流和输出流的边界，确定变换中心
  - 进行第一级分解：将DFD映射成变换型的程序结构
  - 进行第二级分解：将DFD中的加工映射成结构图中的一个适当的模块
  - 标注输入输出信息：根据DFD，在初始结构图上标注模块之间传递的输入信息和输出信息

变换映射

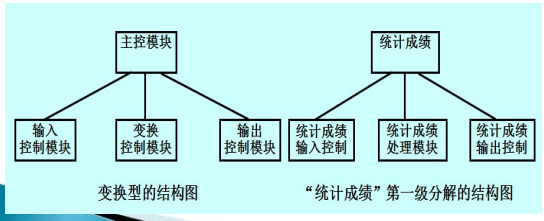


示例：统计成绩子图的输入、输出流边界

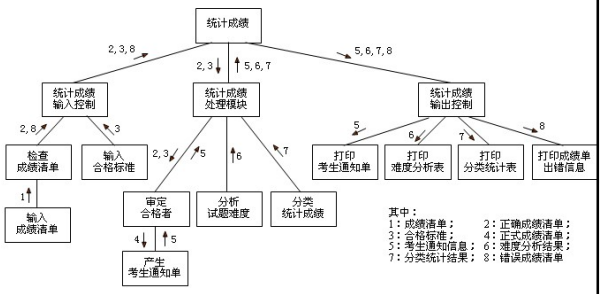


进行第一级分解

- 将DFD映射成变换型的程序结构
- 大型的软件系统第一级分解时可多分解几个模块，以减少最终结构图的层次数
  - 例如，每条输入或输出路径画一个模块，每个主要变换功能各画一个模块

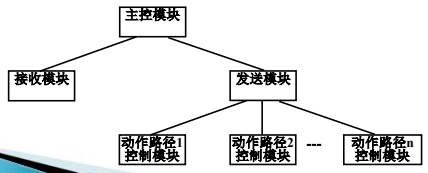


“统计成绩”第二级分解的结构图



事务分析

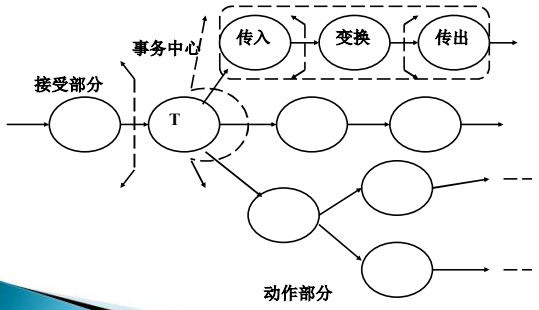
- 将事务型DFD映射成初始的结构图
  - 实例：银行业务中有存款、取款、查询余额、开户、转帐等多种事务，这种软件通常是接收一个事务，然后根据事务的类型执行一个事务处理的功能
- 事务型的结构图包括：
  - 主控模块：完成整个系统的功能
  - 接收模块：接收输入数据(事务)
  - 发送模块：根据输入事务的类型，选择一个动作路径控制模块
  - 动作路径控制模块：完成相应的动作路径所执行的子功能



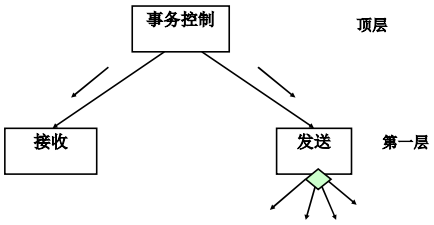
事务分析的步骤

- 在DFD图上确定边界
  - 事务中心
  - 接受部分（包括接受路径）
  - 发送部分（包括全部动作路径）
- 画出SC图框架
  - DFD图的三个部分分别映射为事务控制模块，接受模块和动作发送模块
- 分解和细化接受分支和发送分支

1）划分DFD

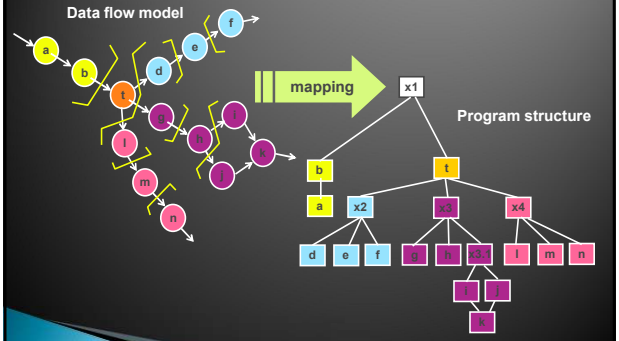


2）画出SC图框架

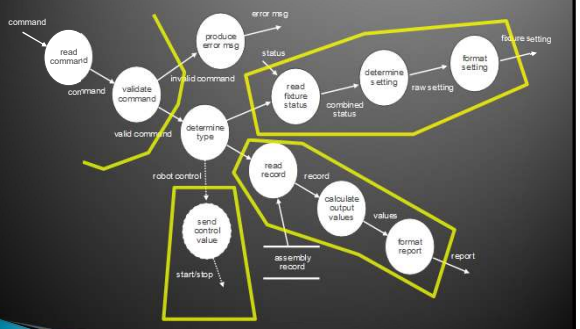


需进一步分解和细化接受分支和发送分支

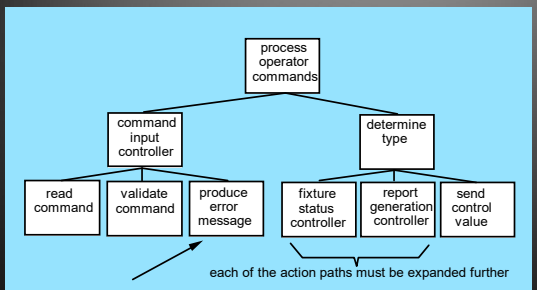
事务映射



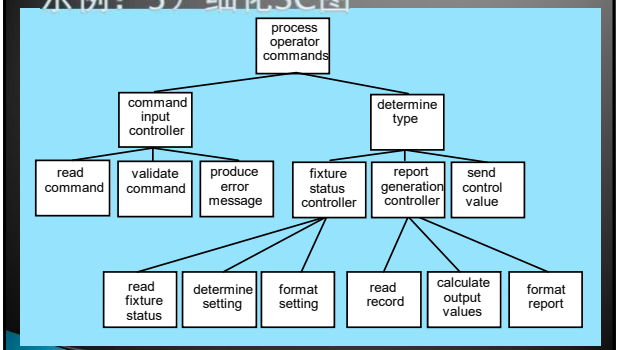
示例：1）划分DFD



示例： 2）画出SC图框架



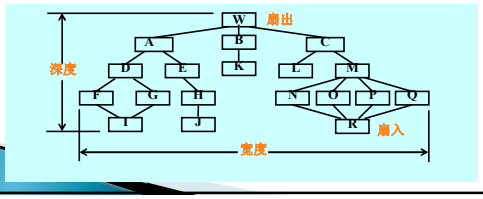
示例：3）细化SC图



优化结构图

结构图的相关度量：

- ***深度***：程序结构图中控制的层数，例如图中所示的结构图的深度是5
- ***宽度***：程序结构图中同一层次上模块总数的最大值，例如图中所示的结构图的宽度为7
- ***扇出(fan out)***：该模块直接调用的模块数目。例如，例如图中模块M的扇出是4，模块A的是2，模块B的扇出是1
- ***扇入(fan in)***：能直接调用该模块的模块数目。例如图中模块G的扇入是1，模块I的扇入是2，模块R的扇入是4



相关指标的含义

- **深度和宽度在一定程序上反映了程序的规模和复杂程度**
  - 相对而言，如果程序结构图的深度和宽度较大，则说明程序的规模和复杂程度都较大。
  - 模块的扇入扇出会影响结构图的深度和宽度，例如减少模块的扇出，可能导致宽度变小而深度增加
- **一个模块的扇出过大通常意味着该模块比较复杂，然而扇出太少，可能导致深度的增加**
  - 一般情况，一个模块的扇出以3～9为宜
- **一个模块的扇入表示有多少模块可直接调用它，它反映了该模块的复用(reuse)程度，因此模块的扇入越大越好**

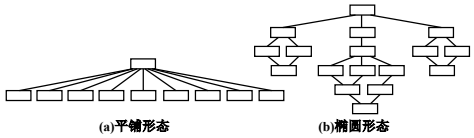
启发式设计策略

按照模块化设计原则，相应的启发式设计策略如下：

1) **降低耦合度，提高内聚度**

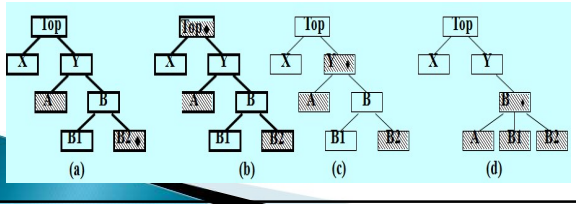
2) **避免高扇出，并随着深度的增加，力求高扇入**

- 避免如图a那样的“平铺”形态，较好的结构图形态是如图b那样的“椭圆”型

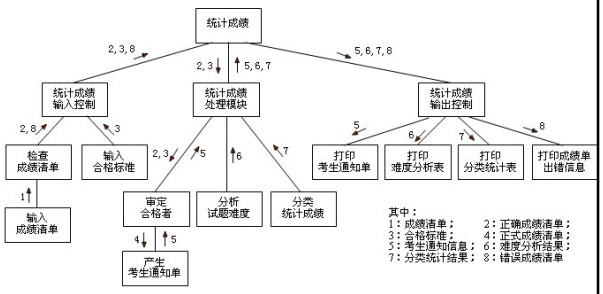


3) **模块的影响范围应阻制在该模块的控制范围内**

- 图a中，模块B2的影响范围(模块A)不在其控制范围(模块B2)内
- 图b中，决策控制是在顶层模块，其影响范围(A、B2)在控制范围内，但是从决策控制模块到被控模块之间相差多个层次
- c和d较合适，d为最好

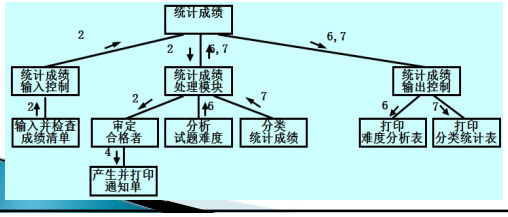


“统计成绩”结构图的改进-0



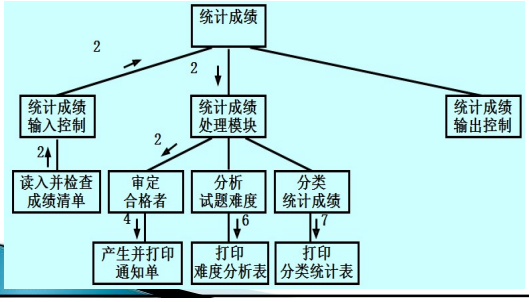
“统计成绩”结构图的改进-1

- **先将一些比较简单的模块合并到与其功能相一致的模块中，以减少耦合度**
  - 将“输入成绩清单”、“检查成绩清单”、“打印成绩单出错信息”合并成“输入并检查成绩清单”
  - 将“输入合格标准”与“审定合格者”合并，仍取名“审定合格者”，但它包含读入合格标准功能
  - 将“产生考生通知单”与“打印考生通知单”合并成“产生并打印考生通知单”



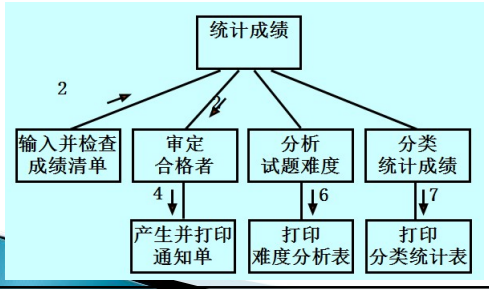
“统计成绩”结构图的改进-2

- **降低模块间的耦合程度**
  - 将“打印难度分析表”模块和“打印分类统计表”模块分别作为“分析试题难度”模块和“分类统计成绩”模块的下属模块



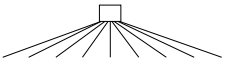
“统计成绩”结构图的改进-3

- 删去“统计成绩输出控制”
- “统计成绩输入控制”模块和“统计成绩处理模块”均为“管道”模块，也可删去



结构图改进技巧

- 减少模块间的耦合度
- 消除重复功能
- 消除“管道”模块
- 模块的大小适中
- 避免高扇出
- 应尽可能研究整张结构图，而不是只考虑其中的一部分



(a)高扇出

(b)重新分解

高扇出时重新分解

作业:

为方便储户，某银行拟开发计算机储蓄系统。储户填写的存款单或取款单由业务员键入系统，如果是存款，系统记录存款人姓名、住址、存款类型、存款日期、利率等信息，并印出存款单给储户；如果是取款，系统计算利息并印出利息清单给储户。

（1） 根据已画出数据流图（DFD）导出计算机储蓄系统的软件结构。

过程设计（详细设计）

- 目标： 确定应该怎样具体地实现所要求的系统。
  - 精确地描述整个目标系统，从而在编码阶段可以把这个描述翻译成用某种程序设计语言书写的程序。
- 任务：编写软件的“过程设计说明书”
  - 为每个模块确定采用的算法
  - 确定每一模块使用的数据结构
  - 确定模块接口的细节

过程设计的原则

- 清晰第一的设计风格
- 结构化的控制结构
- 逐步细化的实现方法

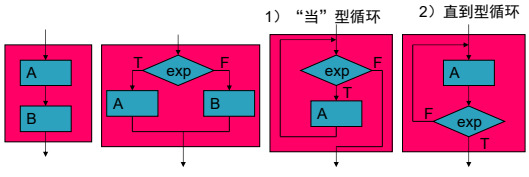
过程设计工具

- 流程图（Flow Diagram）
- N-S图
- PAD图（Problem Analysis Diagram）
- PDL语言

结构程序设计

Structrued Constructs：

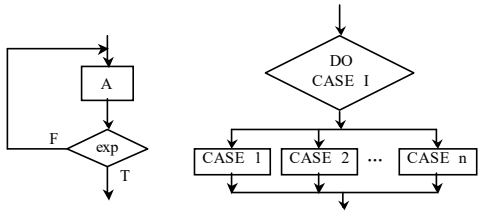
只有顺序、选择、循环这三种基本结构就能实现任何单入口单出口的程序。



顺序结构 (sequence construct)　选择结构 (condition construct)　循环结构 (repetition construct)

结构程序设计：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。

- 经典的结构程序设计：顺序，选择，当型循环
- 扩展的结构程序设计：顺序，选择+多分支，当型循环+直到型循环
- 修正的结构程序设计：顺序，选择+多分支，当型循环+直到型循环，break结构

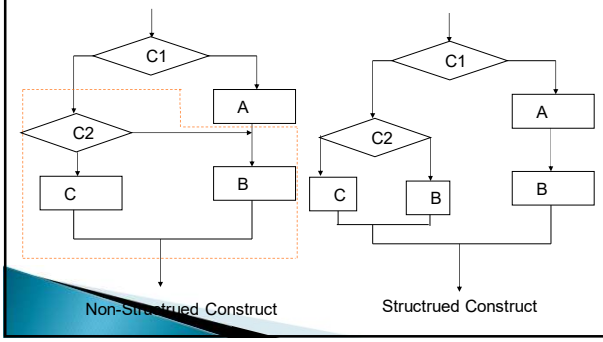


(a) DO_UNTIL 型循环结构　　(b) DO_CASE 型多分支结构

其他常用的控制结构

怎样把一个非结构化程序转换成结构化程序

- 重复编码技术

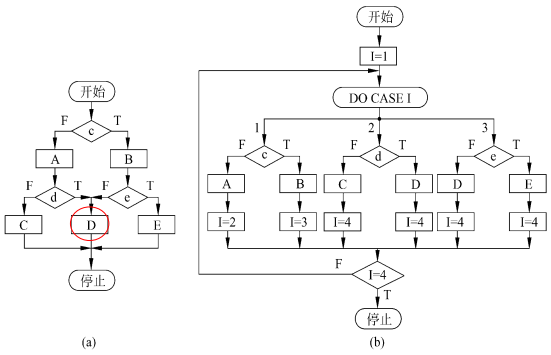


Non-Structrued Construct　　Structrued Construct

- 设标志量技术

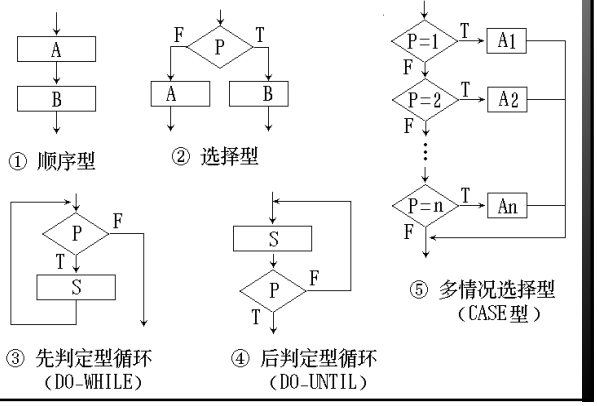
```
While (p)                          while(p&&!c)
{ s1;                              { s1;
  if(c) break;     =>?               s2;
  s2;                                }
}
            flag=1;
            while(p&&flag)
  =>?       { s1;
              if (c ) flag=0;
              else    s2;
            }
```

- 状态变量法

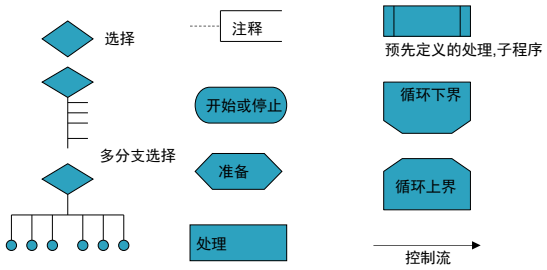


Non-Structrued Construct　　Structrued Construct

程序流程图

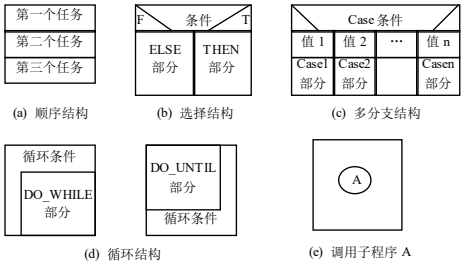


程序流程图中常用的符号



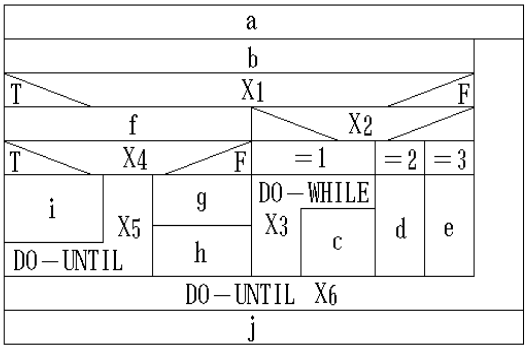
尽量保证流线的上进下出！

盒图（N-S图）（Box Diagrams）

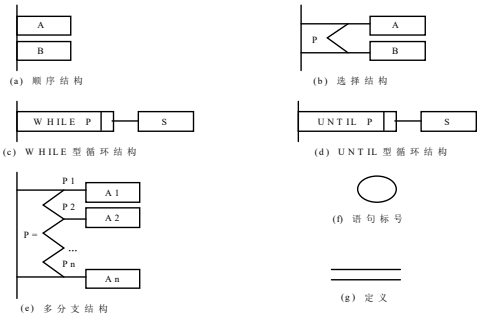


N-S图的基本符号

N-S图示例

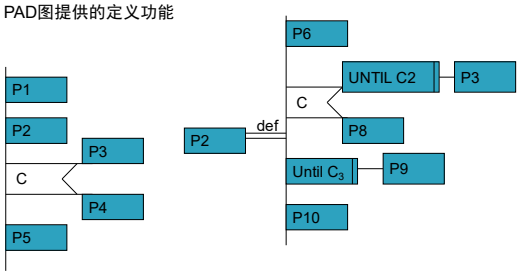


6.2.3 PAD图（Problems Analysis Diagrams）



PAD图的基本符号

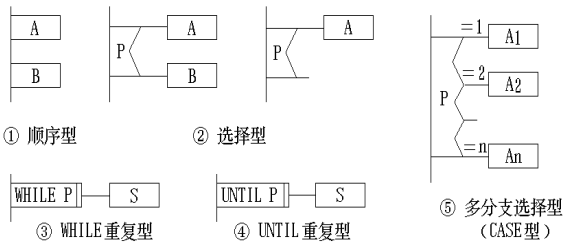
PAD图提供的定义功能



PAD图的主要优点:

- 使用PAD符号设计的程序必然是结构化的程序.
- PAD图所描绘的程序结构十分清晰.
- 用PAD图表现程序逻辑,易读，易记，易懂.
- 容易将PAD图 转换成高级语言源程序.可用软件工具实现自动转换.
- 即可以表示程序逻辑,也可以描绘数据结构.
- 支持自顶向下,逐步求精方法的使用.

问题分析图(PAD)



PDL (Program Design Language)

过程设计语言(伪码 pseudocode)

- PDL是一种用于描述功能模块的算法设计和加工细节的语言。称为设计程序用语言。它是一种伪码。
- 伪码的语法规则分为“外语法”和“内语法”。
- PDL具有严格的关键字外语法，用于定义控制结构和数据结构，同时它的表示实际操作和条件的内语法可使用自然语言的词汇。

示例: 拼词检查程序

- PROCEDURE spellcheck IS
  BEGIN
    split document into single words
    look up words in dictionary
    display words which are not in dictionary
    create a new dictionary
  END spellcheck

作业：

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。