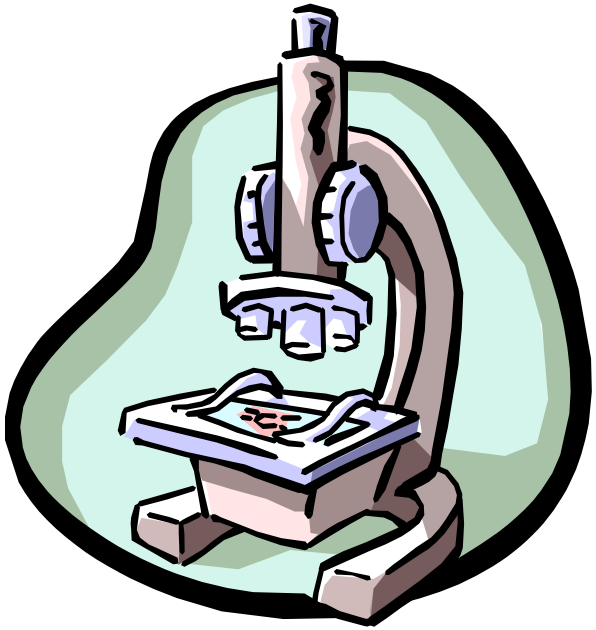
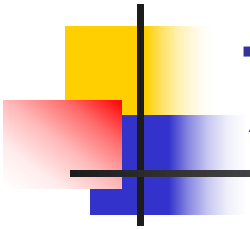
第7章 详细设计



中南大学计算机学院　　任胜兵

7.1 详细设计的目标与任务

- 概要设计确定了软件系统的总体结构，详细设计则对概要设计结果进一步细化，给出目标系统的精确描述，以便在编码阶段直接翻译成计算机上能够运行的程序代码。
- 在详细设计阶段，结构化设计和面向对象设计没有本质区别，都是确定算法和数据结构。
- 详细设计的结果“软件详细规格”（相当于工程施工图纸），与编程思想、方法和风格等密切相关。



7.1.1 详细设计的目标

- 详细设计阶段具体地设计所要求的系统，得出新系统的软件详细规格。同时，要求设计出的规格简明易懂，便于下一阶段用某种程序设计语言在计算机上实现。
- 如何高质量地完成详细设计是提高软件质量的关键。

7.1.2 详细设计的任务

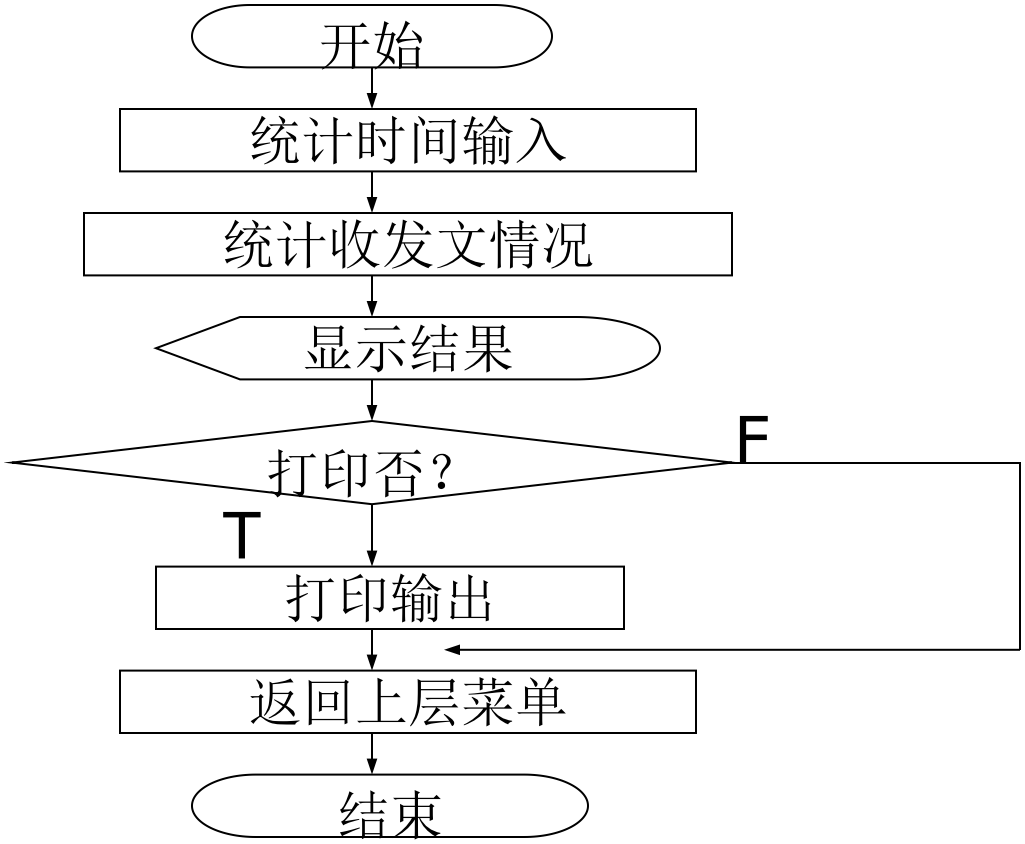
- 算法过程的设计;
- 数据结构的设计;
- 数据库物理设计;
- 信息编码设计;
- 测试用例的设计;
- 编写“详细设计说明书”。

7.2 详细设计图形描述工具

详细设计中用于过程设计的图形工具，包括程序流程图、盒图、问题分析图和协作图。实际上，除此之外的其它工具，如IPO图、判定表/判定树以及伪代码语言等均可用于对过程设计的描述。

每种工具都有它自己的优缺点，在设计时可以针对不同的情况选用，甚至可以同时采用多种工具来描述详细设计的结果。

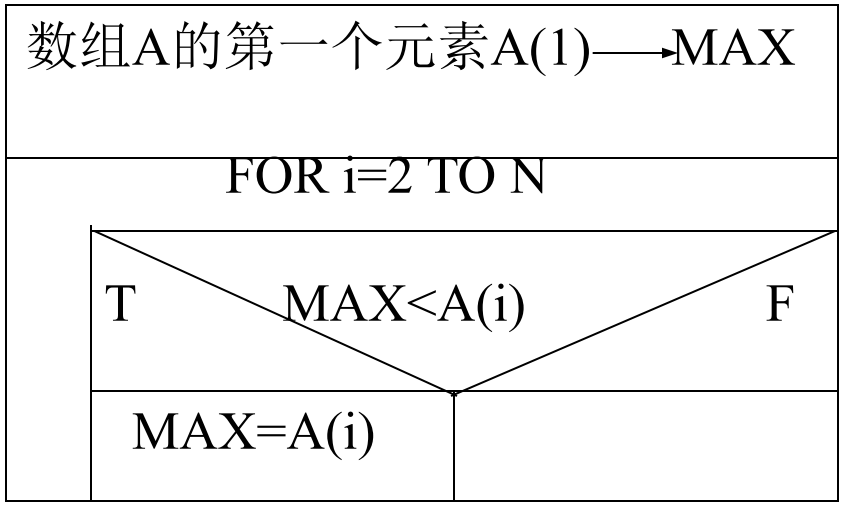
7.2.1 程序流程图

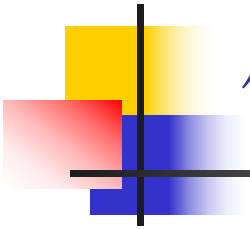


优缺点

- 主要优点：直观、灵活、易使用，便于初学者掌握。
- 缺点：程序流程图本质上并不是逐步求精的好工具，它诱使程序设计人员过早考虑程序的控制流程，而不去考虑程序的全局结构；另一方面，用箭头表示控制流方向，可以实现控制流的任意转移，如果使用得当则简单易懂且灵活，否则非结构化的程序流程图可能非常难理解，无法进行修改和维护。同时，程序流程图也不便于表示数据结构。

7.2.2盒图

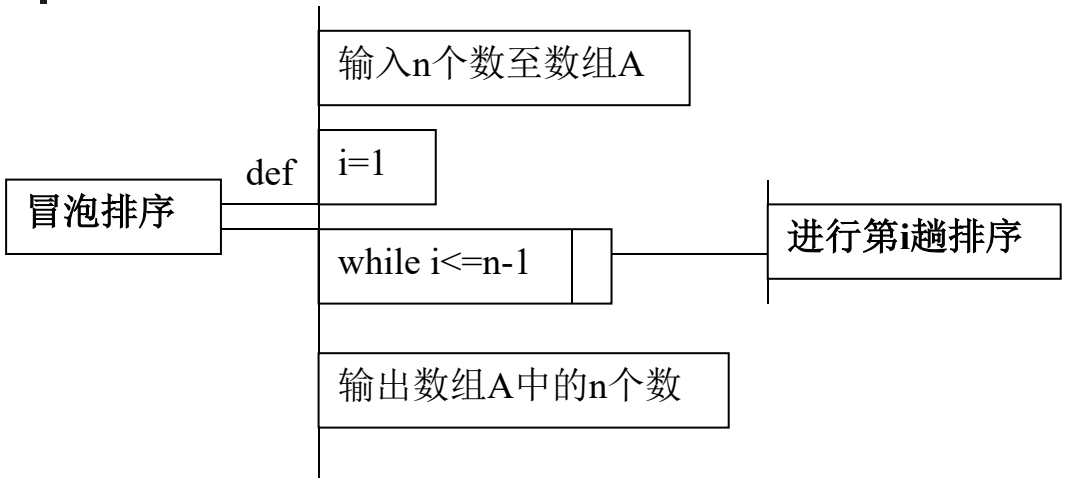


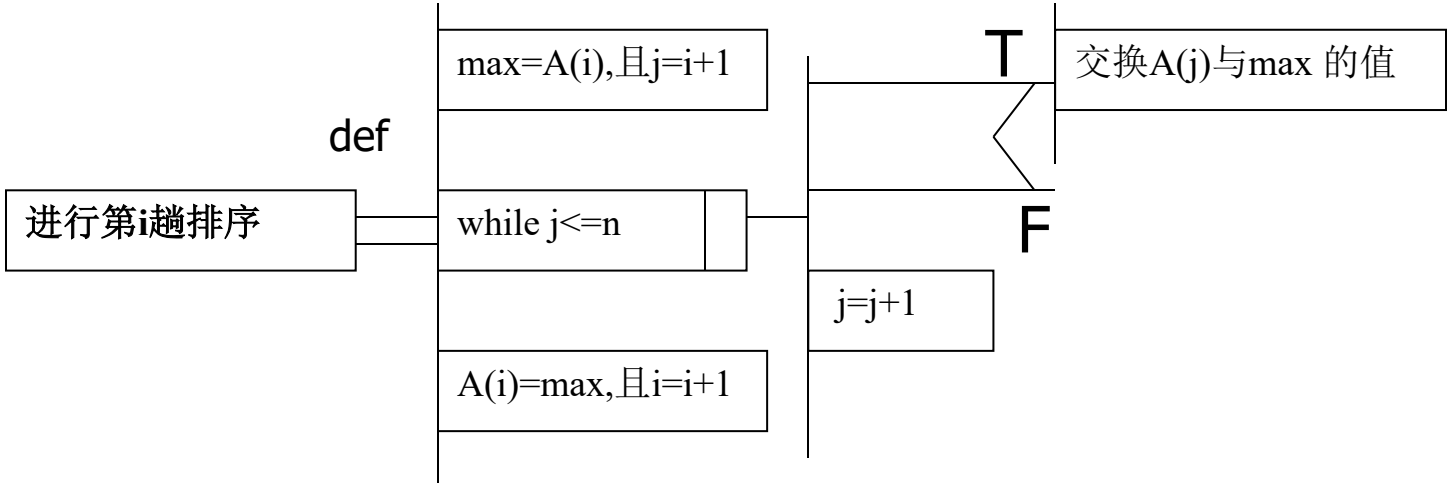


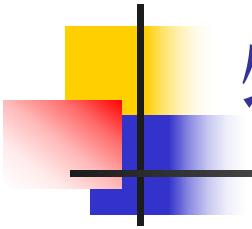
优缺点

- 主要优点：它强制设计人员使用结构化技术，从而可以保证设计的质量。同时，从盒图上可以直观地看出某一特定控制结构的作用范围，为理解设计意图、编程实现、选择测试用例等带来了方便。在使用时还可附上一个描述数据结构的盒子，使得盒图更加适用于详细设计。
- 缺点：盒图的修改比较麻烦，且结构嵌套层次较多时不太容易绘制，以致盒图的使用至今仍不流行。

7.2.3 问题分析图


输入n个数至数组A
def
i=1
冒泡排序
while i<=n-1
进行第i趟排序
输出数组A中的n个数

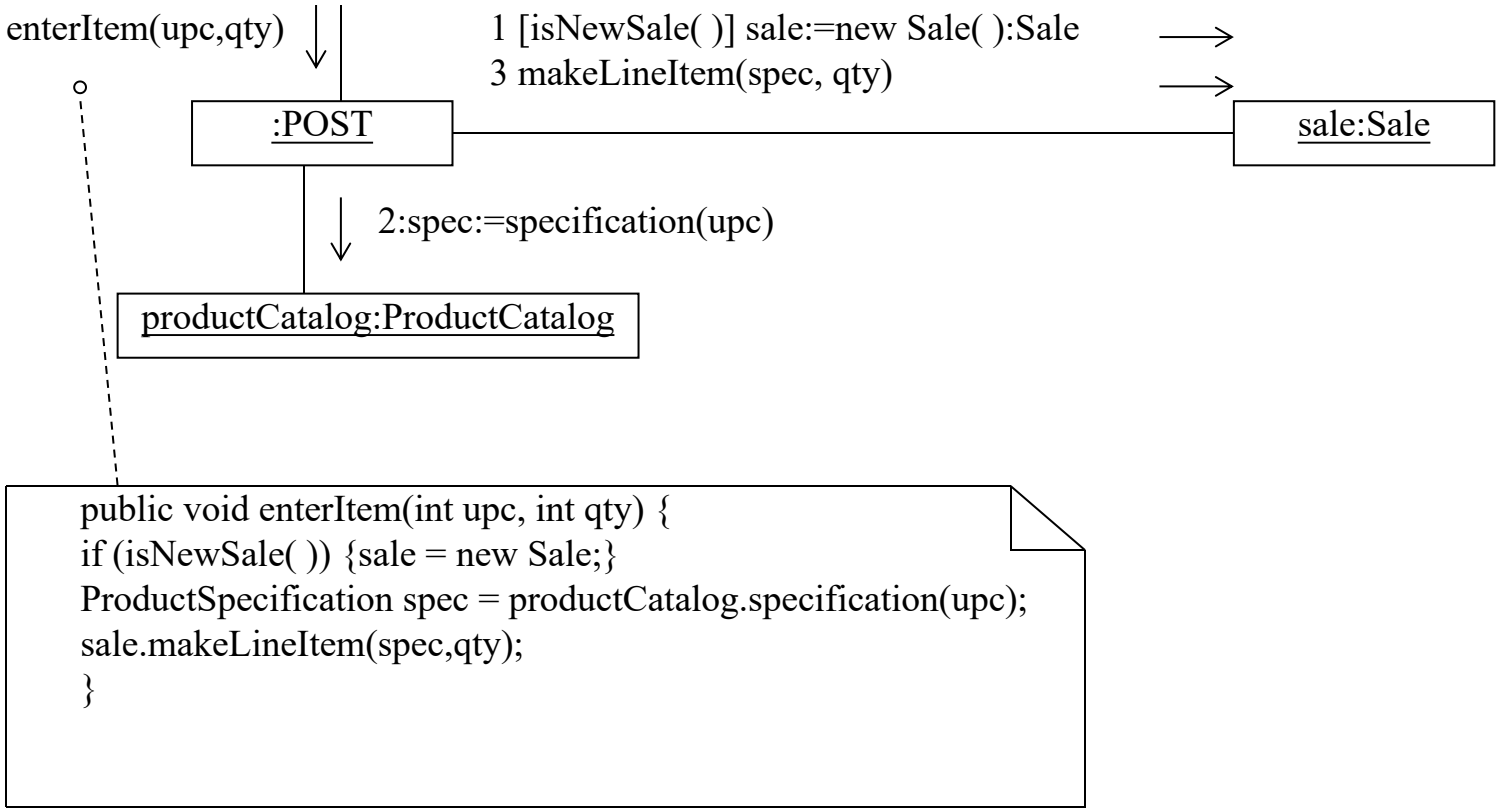

max=A(i),且j=i+1
T
交换A(j)与max 的值
def
进行第i趟排序
while j<=n
F
j=j+1
A(i)=max,且i=i+1



特点

- 问题分析图强制设计人员采用结构化技术。
- 采用树形结构，既克服了程序流程图不能清晰表现程序层次结构的缺点，又不同于盒图将处理约束在一个盒子里而使修改麻烦，并且这种结构为软件的自动生成提供了有力的帮助（树遍历）。

7.2.4协作图


enterItem(upc,qty)
1 [isNewSale( )] sale:=new Sale( ):Sale
3 makeLineItem(spec, qty)
:POST
sale:Sale
2:spec:=specification(upc)
productCatalog:ProductCatalog
public void enterItem(int upc, int qty) {
if (isNewSale( )) {sale = new Sale;}
ProductSpecification spec = productCatalog.specification(upc);
sale.makeLineItem(spec,qty);
}

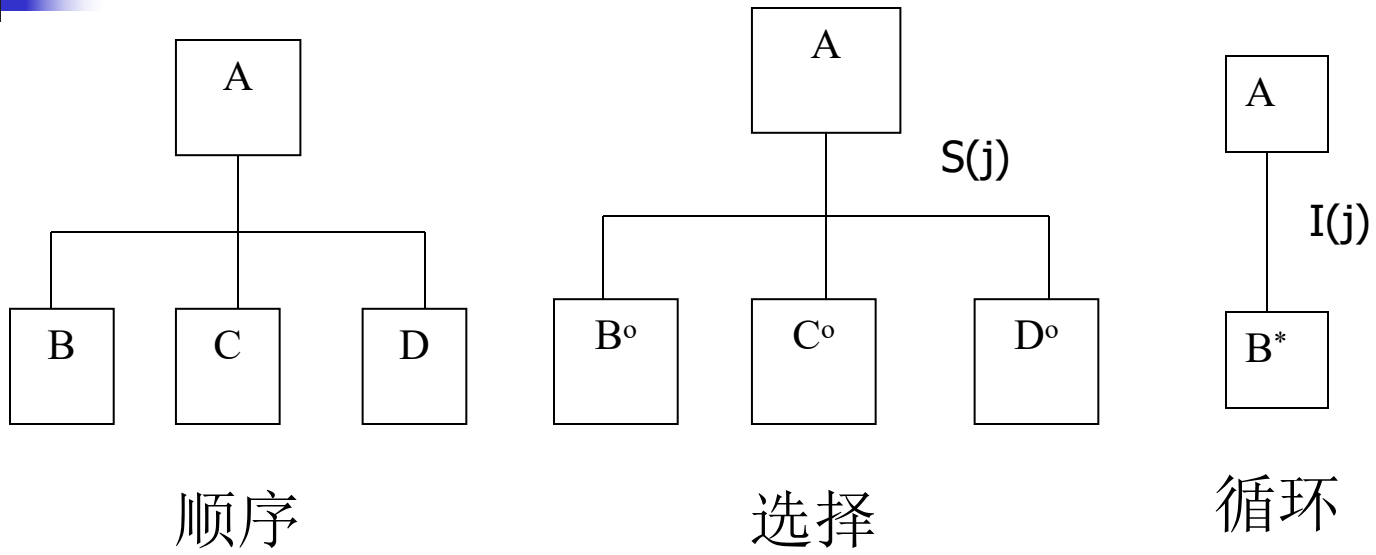
特点

- 协作图和顺序图都可用来表示对象之间的交互关系，两种形式之间可以相互转换。
- 顺序图突出执行的时序，能更方便地看出事情发生的次序。而协作图的布局方法能更清楚地表示出对象之间的静态的连接关系。
- 与顺序图相比，协作图的绘图空间也要节省一些。

7.3 Jackson程序设计方法

- Jackson结构程序设计方法（Jackson Structured Programming，JSP）是英国人M.A.Jackson首先提出来的，是一种面向数据结构的结构化程序设计方法。
- 该方法通过分析问题的输入、输出数据结构（用Jackson图表示）的对应关系，按一定的映射规则将其映射成软件的过程描述，用Jackson伪代码表示。

7.3.1 Jackson图



顺序　　选择　　循环

7.3.2 Jackson伪代码

A **seq**
B
C
D
end A

顺序

A **select** condition S(j)
B
or condition 1
C
or condition 2
D
end A

选择

A **iter** **until** (or **while**) condition I(j)
B
end A

循环

7.3.3 Jackson程序设计方法的步骤

- 分析并确定问题的输入和输出数据结构，并用Jackson图表示。
- 找出输入和输出数据结构中有对应关系的数据单元。
- 将数据结构映射为程序结构。
- 列出完成程序结构图中各处理框功能的全部操作，以及有关条件，并将它们分配到程序结构图的适当位置。
- 用Jackson伪代码写出与程序结构图相对应的过程性表示。

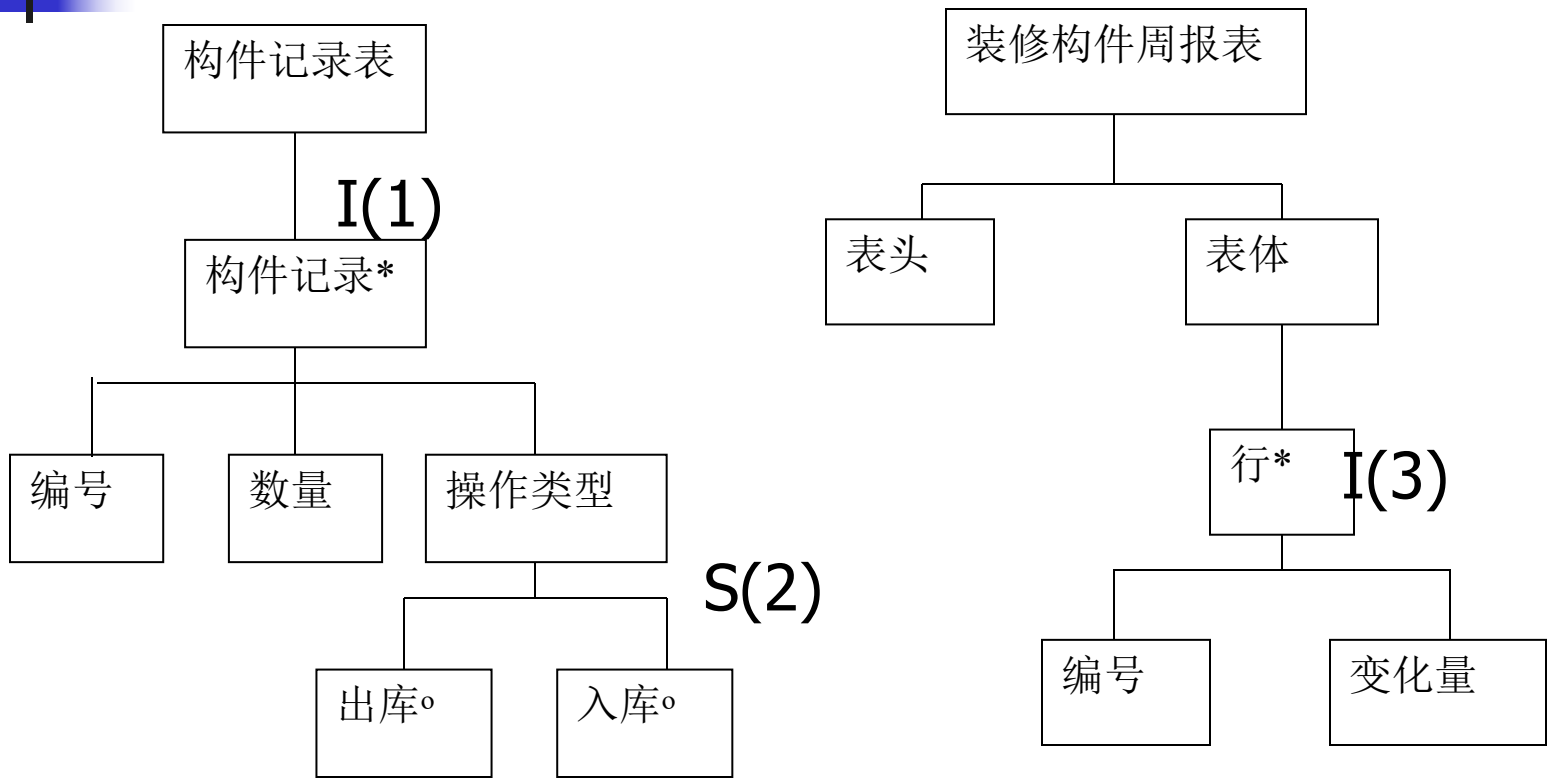
例子

某装修公司的仓库存放有多种装修构件，每种构件的每次进货、出货都有相应的记录（包括构件编号、名称、数量、入/出库操作类型）存于构件记录表中。装修公司每周根据构件记录表打印一张周报表，报表的每行列出某种构件本周变化的数量，以便根据报表及时订货，有关表格如：

编号	名称	数量	操作类型
0130	内角线	50	入
0131	九夹板	20	出
0130	内角线	40	出

×××装修公司构件周报表	
编号	变化量
0130	+10
0131	-20
.....	

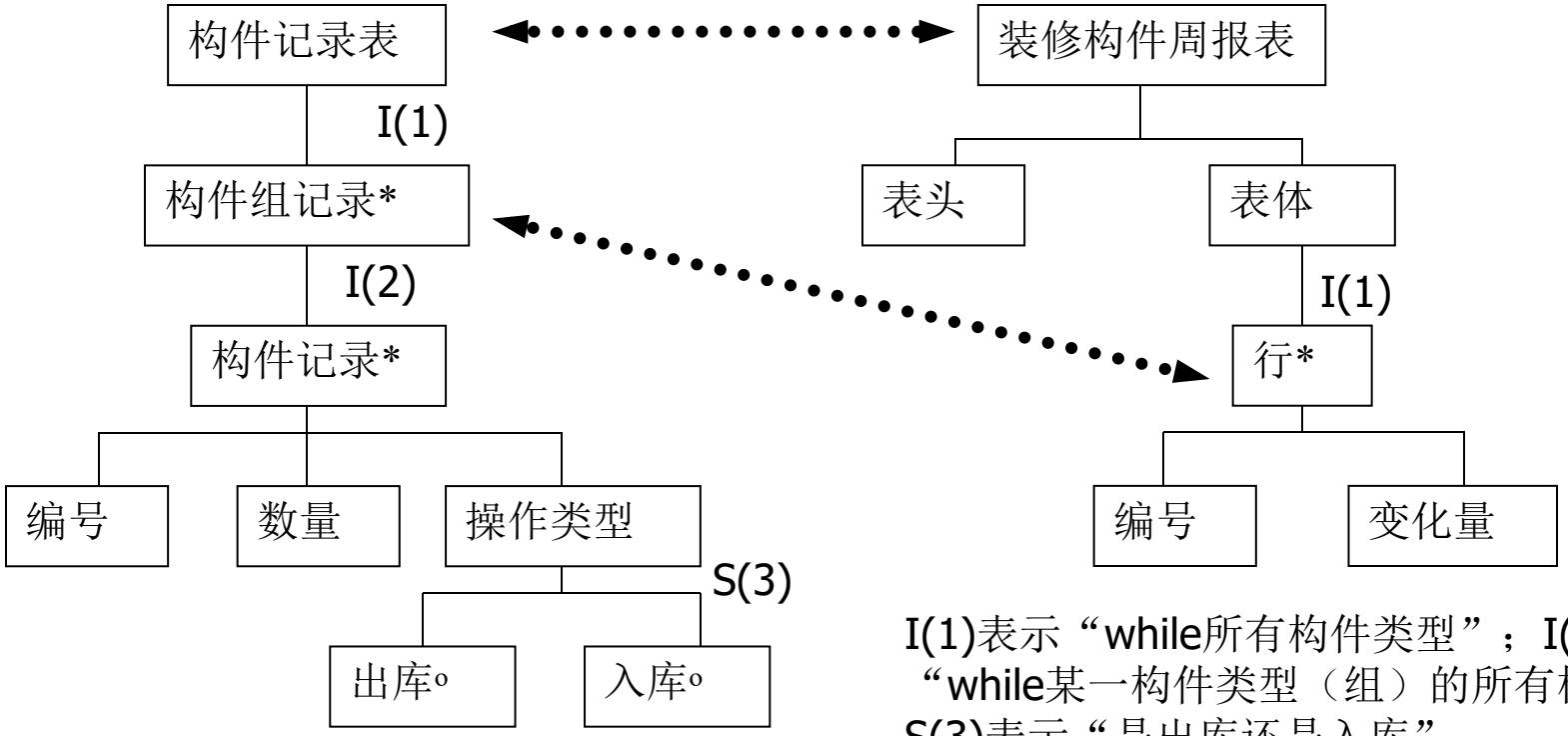
分析并确定输入输出数据结构



I(1)表示“while 构件记录表未空”；S(2)表示“是出库还是入库”；I(3)表示“while所有构件类型”。

找出输入输出数据结构中的对应单元

- 所谓有对应关系是指有直接的因果关系，在程序中可以或需要同时处理的单元。对于重复出现的数据单元必须是重复的次序和次数在输入与输出数据结构中都相同才可能有对应关系。

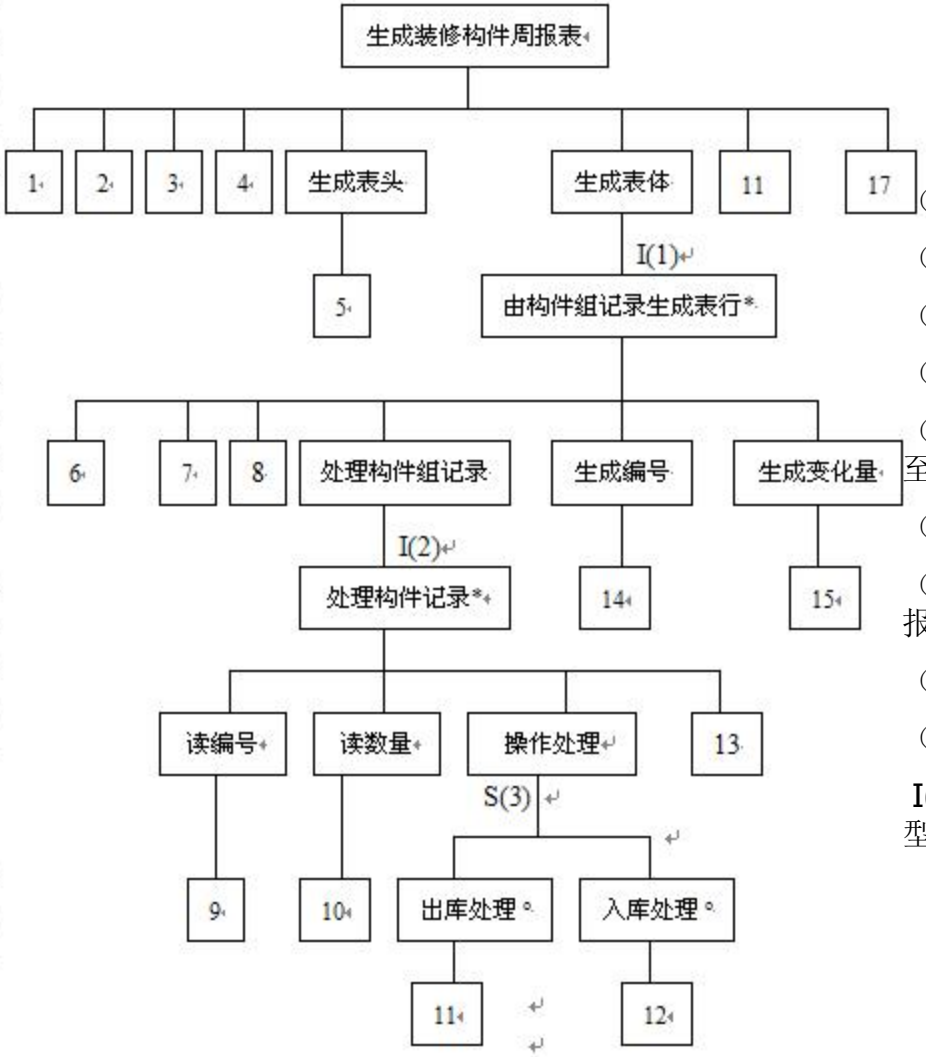


I(1)表示“while所有构件类型”；I(2)表示“while某一构件类型（组）的所有构件”；S(3)表示“是出库还是入库”。

利用映射规则导出程序结构

- 对每对有对应关系的数据单元，按其在数据结构图中的层次在程序结构图相应层次画一个处理框。
- 如果这对数据单元在输入数据结构和输出数据结构中处于不同的层次，则与它们对应的处理框在程序结构图中的层次同它们在数据结构图中层次低的对应（高层在上，低层在下）。
- 对输入数据结构中剩余的数据单元，根据其所在的层次在程序结构图相应层次分别画上处理框。
- 对输出数据结构中剩余的数据单元，根据其所在的层次在程序结构图相应层次分别画上处理框。

列出并分配操作与条件



（1）输入构件记录库文件名　　（2）打开构件记录库文件名

（3）对构件记录库文件按编号索引　（4）创建报表文件

（5）生成表头字符串至报表文件　　（6）送换行符至报表文件

（7）变化量change=0　　（8）ID置为当前构件记录的编号

（9）读构件记录库文件的编号至ID　（10）读构件记录库文件的数量至data

（11）change=change-data　(12）change=change+data

（13）指向构件记录库文件的下一条记录　（14）生成编号ID字符串至报表文件

（15）生成变化量change串至报表文件（16）关闭构件记录库文件

（17）关闭报表文件　　I(1)：构件记录库文件未到文件尾

I(2)：当前构件记录的编号不等于ID　S(3)：当前构件记录的操作类型是出库

用Jackson伪代码表示程序处理过程

- 生成装修构件周报表 **seq**
- 输入构件记录库文件名
- 打开构件记录库文件
- 对构件记录库文件按编号索引
- 创建报表文件
- 生成表头 **seq**
- 生成表头字符串至报表文件
- **end** 生成表头
- 生成表体**iter** **while** 构件记录库文件未到文件尾
- 由构件组记录生成表行 **seq**
- 送换行符至报表文件

……

7.4 Warnier程序设计方法

- Warnier程序设计方法由法国人J.D.Warnier提出的一种LCP（Logical Construction of Programs，逻辑构造程序）程序设计方法。
- 在Warnier程序设计方法中，首先要分析输入和输出数据结构，用Warnier图表示；然后，从数据结构（主要是根据输入数据结构，输出数据结构可用来完善输入数据结构）导出程序结构，同样用Warnier图表示；再将程序结构改用程序流程图来表示；最后，根据程序流程图写出程序的详细过程性描述。

7.4.1 Wariner图

符 号	意 义	说 明	意 义
{	表示层次组成	(1次)	顺序结构
⊙	表示“或”(or)	(n次)	重复结构
¯	表示“非”，如A表示非A	(0或1)次	选择结构

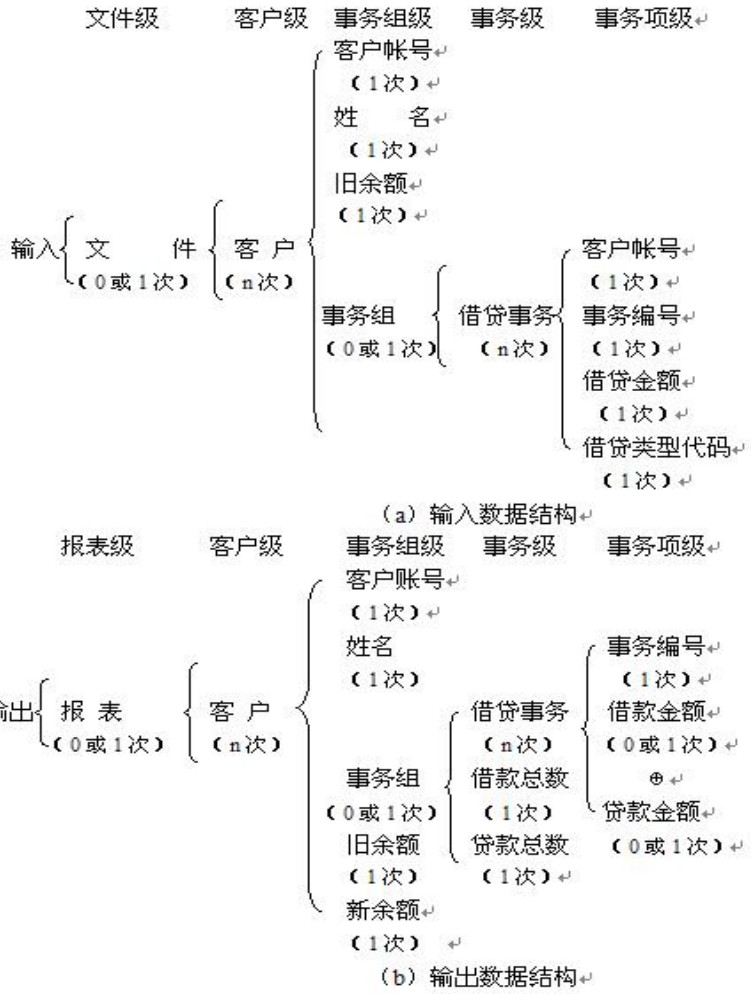
7.4.2 Warnier程序设计步骤

- 分析和确定问题的输入和输出数据结构，并用Warnier图来表示；
- 从数据结构（特别是输入数据结构）导出程序的处理结构，用Warnier图表示；
- 将程序结构改用程序流程图表示；
- 根据上一步得出的程序流程图，写出程序的详细过程性描述：

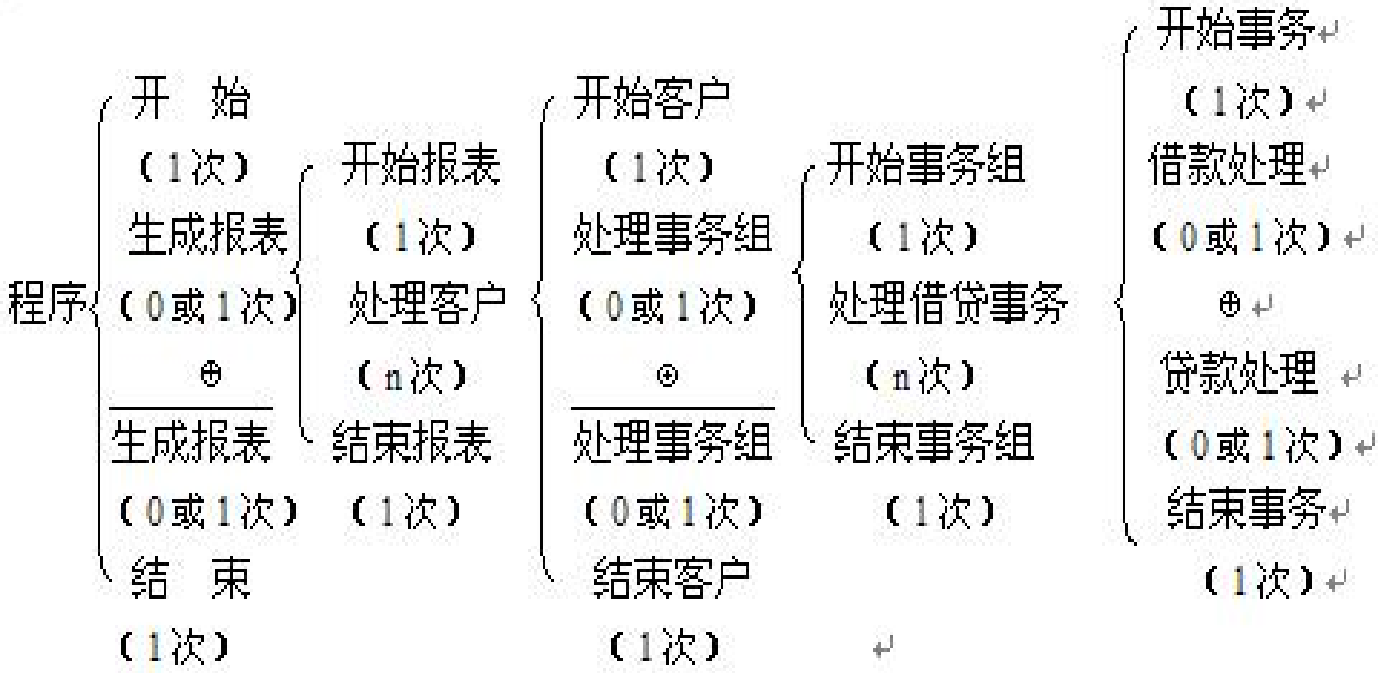
例子-----报表生成系统


有借贷事务
客户帐号 姓 名 事务编码 借款金额
事务编码 贷款金额
旧余额 新余额 借款总数 贷款总数
无借贷事务
客户帐号 姓 名
旧余额 新余额

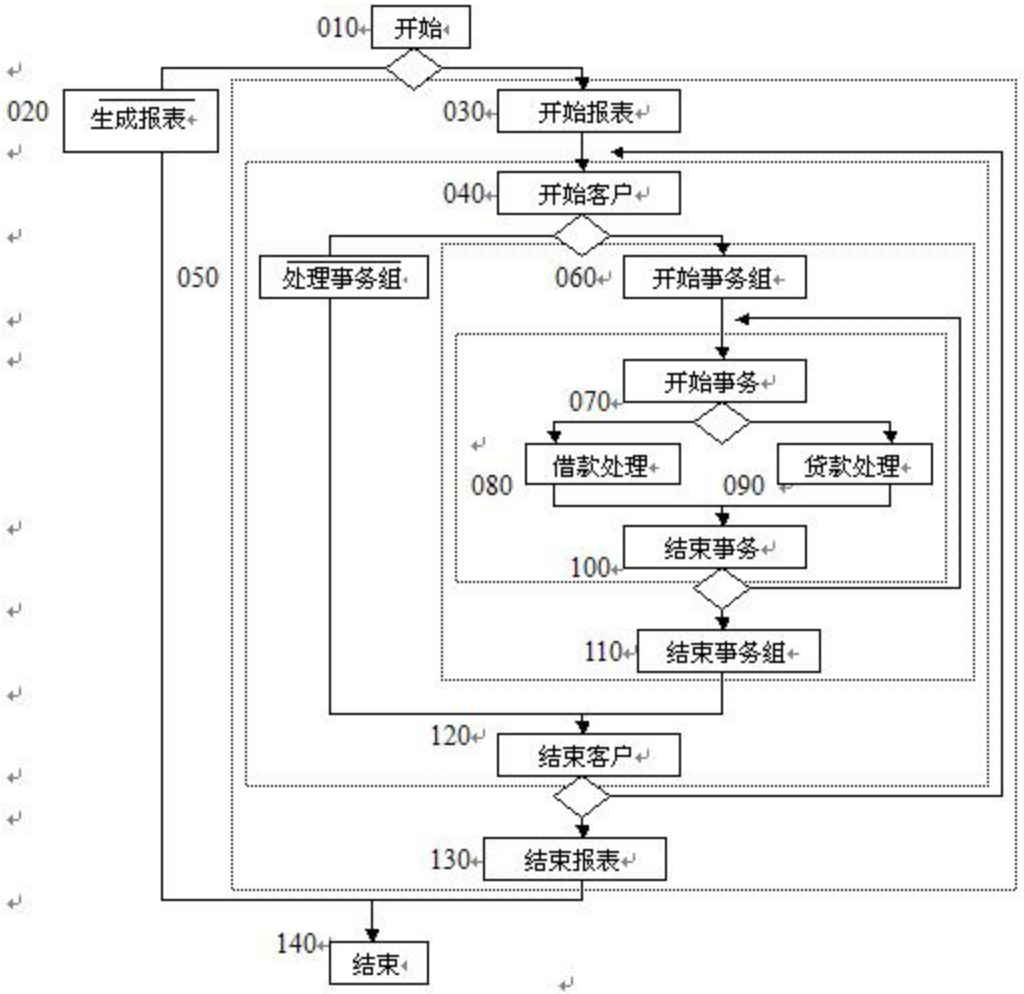
分析并确定问题的输入和输出数据结构



导出程序处理的层次结构



画出和程序处理的层次结构相对应的程序流程图



根据程序流程图写出程序的详细过程性描述

①自上而下给流程图每个处理框统一编号；

- ②列出每个处理框所需要的指令，冠以处理框的序号，并对指令分成如下几类：

  A、输入和输入准备指令；

  B、分支和分支准备指令；

  C、计算指令；

  D、子程序调用指令；

  E、输出和输出准备指令。

③将上述“分类指令表”中的指令全部按处理框序号重新排序，序号相同则基本按“输入/处理/输出”的顺序排列，从而得到了程序的详细过程性描述。

7.5 面向对象程序的详细设计

- 面向对象设计在详细设计阶段主要完成对象的属性和方法的设计，称之为面向对象程序的详细设计。
- 面向对象程序的详细设计与结构化程序的详细设计相似，但也有特殊之处，在设计时要认真考虑才能体现面向对象的优越性。

7.5.1面向对象程序的特性

- 封装性：类的封装性使得数据和操纵数据的算法（函数或过程）紧密地捆绑在一起，这样就可以使得操纵数据的函数或过程的作用域和可视性限制在软件系统的局部区域内。

```
//定义日期字符串类型string80
typedef  char  String80[80];
//定义日期类CDate
class CDate
{//类的实现
private:
    int year,month,day;
//类的接口
public:
    CDate(int month,int day,int year);
    CDate operator+(int days);
    void GetDateString(String80 &DateString);
    //…
};
//部分成员函数的实现
void CDate::GetDateString(String80 &DateString)
{    sprintf(DateString, "%d-%d-%d", month, day, year % 100);
}
```

继承性

- 在面向对象程序设计中，允许某个类继承其它类的成员函数或数据成员。被继承的类称为基类、父类或超类，继承的类称为派生类或子类。

```
class CBaseClass
{   …
        …
};
```

若要从CBaseClass派生一个新的类，其语法是：

```
class CDerivedClass：存取权限标识 CBaseClass
{   …
        …
};
```

多态性

- 多态性使得相关的类可有同名的函数，这个同名的函数根据不同类产生不同的结果。换言之，不同类的对象可以具有相同的接口，这些相同的接口自然会呈现出不同的行为。通过多态性程序设计，可以编写并编译代码以处理未知类型的对象。

在C++中，多态性是通过虚拟函数实现的。

```
//定义日期字符串类型string80
typedef  char  String80[80];

//定义日期类CDate
class CDate
{
protected:
    int year,month,day;
public:
    CDate(int month,int day,int year);
    CDate operator+(int days);
    virtual void GetDateString(String80 &DateString);
    void DisplayDateString(void); //将调用GetDateString
//…
};
```

如果不采用多态性处理GetDateString，则在CDate的子类中如果重写了GetDateString会影响DisplayDateString吗？

7.5.2 设计原则

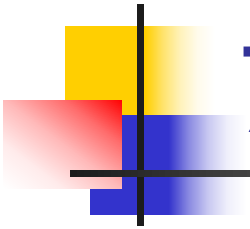
- 可复用性:保证方法的内聚性;减少一个方法的代码规模;保持方法对外接口的一致性;分离策略（控制）方法和实现方法;方法应均匀覆盖数据；加强封装性；减少方法的耦合性。
- 可扩展性：封装数据；封装方法内部的数据结构；避免情况分支语句；区分公有方法和私有方法。
- 健壮性：防止输入错误（围栏）；把握优化代码的时机；检查参数的合法性；选择适当的实现方法。
- 协作性：对类进行详细的文档化；把类打包成模块；尽量使得代码容易理解；等等

7.6 程序规格说明文档及复审

- 程序规格说明又称详细设计说明，与概要设计说明相比，程序规格说明着重描述模块的细节，包括算法过程和数据结构。
- 如果软件系统比较简单，则可将程序规格说明并入概要设计说明中。

7.6.2 程序规格说明复审

- 程序规格说明的复审类似于概要设计说明的复审，但重点在于各个模块的具体设计上。例如：模块的说细设计能否满足其功能与性能要求？详细设计描述是否简单、清晰？说细设计选择的算法与数据结构是否合理，符不符合程序实现语言的特点？等等。



7.7 小结

- 详细设计是进行逻辑系统开发的最后一个阶段，其质量的好坏将直接影响到系统的编码实现。
- 合理选择和正确使用有关工具、深入理解和掌握有关设计思想和方法，对搞好详细设计是非常重要的。
- 结构化程序的详细设计与面向对象程序的详细设计有许多共性。