

# Think in COBOL

# 目录

第一章 COBOL程序设计初步

第二章 匹配处理

第三章 表

第四章 数据编辑

第五章 表及表检索

第六章 子程序

第七章 排序

第八章 文件处理





## 程序案例业务介绍

## 微型银行储蓄业务核算系统

---

### ▶ 业务包括:

- ▶ 开户
- ▶ 活期储蓄存入、取出、周期性利息结算、销户利息结算
- ▶ 定期储蓄存入、取出、利息结算
- ▶ 销户





# 第一章 COBOL程序设计初步



- § 1.1 COBOL语言的历史与特点
- § 1.2 COBOL语言的编译方法
- § 1.3 COBOL程序的结构与书写格式
- § 1.4 COBOL字符、字、数据名与变量
- § 1.5 COBOL基本逻辑语句
- § 1.6 COBOL数据定义与编辑
- § 1.7 COBOL的标识部和环境部
- § 1.8
- § 1.9
- § 1.10 COBOL



# 第一章 COBOL程序设计初步

## § 1.1 COBOL语言的历史与特点

### ► COBOL语言的历史

#### ► COmmon Business Oriented Language

用于商业数据处理与管理，如：银行，会计业，人事管理，财会，统计报表，情报检索，证券，金融等等

- 1959年05月 美国国防部召开数据系统语言会议
- 1959年12月 第一个COBOL语言文本
- 1960年04月 正式发表COBOL 60，经过扩充完善COBOL 61
- ANSI COBOL 61 --- 最早的版本
- 1965年COBOL 65 --- ANSI COBOL 68 --- ISO COBOL 72
- 1972年COBOL 72 --- ANSI COBOL 74 --- ISO COBOL 78
- ANSI COBOL 85 --- 应用最广泛的版本
- ANSI COBOL 2002 --- 最新版本



# 第一章 COBOL程序设计初步

## § 1.1 COBOL语言的历史与特点

### ► COBOL语言的特点

- 描述性好，能根据需要描述各种形式的数据
- 适合大批量数据处理，能对数据进行严密的组织(算术运算简单但运算量大，逻辑运算多)
- 接近自然语言(英语)，成文自明
  - 例如：ADD A TO B GIVING C
- 遵循ISO标准，通用性强，移植方便
- 格式固定，结构严谨，层次分明
- 缺点是比较繁琐

事務処理向き

- 帳表出力やファイル操作が容易

わかりやすさ

- 英語に似ている

構造化が容易

- 構造化が容易
- 保守効率がよい





## § 1.1 COBOL语言的历史与特点

### ► COBOL所处理数据的特点

- ▶ **层次**: 数据间不是孤立的, 而是存在从属关系
- ▶ **记录**: 具有一定层次关系的一组数据项的最大集合
- ▶ **文件**: 记录在外部介质上的记录的集合
- ▶ **库**: 由若干个文件组成
- ▶ **初等项**: 数据的基本单位
- ▶ **组合项**: 由若干初等项和低一层组合项组成
- ▶ **层号**: 由两位整数组成, 用来表示层次, 层号约小则层次越高



# 第一章 COBOL程序设计初步

## § 1.2 COBOL程序的编译



### ▶ RDz编译方式



# 第一章 COBOL程序设计初步

## § 1.2 COBOL程序的编译



### ► NetCOBOL编译方式



## § 1.2 COBOL程序的编译

### ► zOS编译方式

- 创建分区数据集ST×××.COBOL.SOURCE(COB××) 存放源程序
- 创建分区数据集ST×××.COBOL.LOAD 存放可执行模块  
它的属性比较特殊：  
Record format = U  
Record length = 0  
Block size = 6144
- 创建顺序数据集ST×××.COBOL.COMLINK  
编写JCL用于编译链接源文件 [例1.3](#)
- 创建顺序数据集ST×××.COBOL.RUN  
编写JCL用于运行可执行模块

## § 1.2 COBOL程序的编译

### ▶ 示例

- ▶ [例程1.2.1](#) 了解COBOL程序的基本结构
- ▶ [例程1.2.2](#) 简单的变量定义，输入输出及运算



## § 1.3 COBOL程序的结构与书写格式

### 部

- IDENTIFICATION DIVISION (标识部)  
主要指定源程序的名称，也可记录备忘信息，如日期作者
- ENVIRONMENT DIVISION (环境部)  
指出程序中用到的数据文件名与系统设备的对应关系
- DATA DIVISION (数据部)  
说明程序中所有数据的类型和所占内存大小
- PROCEDURE DIVISION (过程部)  
定义程序要执行的指令，是程序的核心

見出し部

環境部

データ部

処理部

**注意：四个部缺一不可！即使部的内容为空也要写全！**



## § 1.3 COBOL程序的结构与书写格式

### ▶ 节，段，描述体

- ▶ 部下设置节(SECTION)，节下设段(PARAGRAPH)或描述体(DESCRIPTION ENTRY)
- ▶ 标识部下直接定义段
- ▶ 环境部下定义节，节下定义段
- ▶ 数据部下定义节，节下定义描述体
- ▶ 过程部下定义节，节下定义段(复杂程序)  
也可以直接定义段(一般程序)



## § 1.3 COBOL程序的结构与书写格式



### 句子，语句和子句

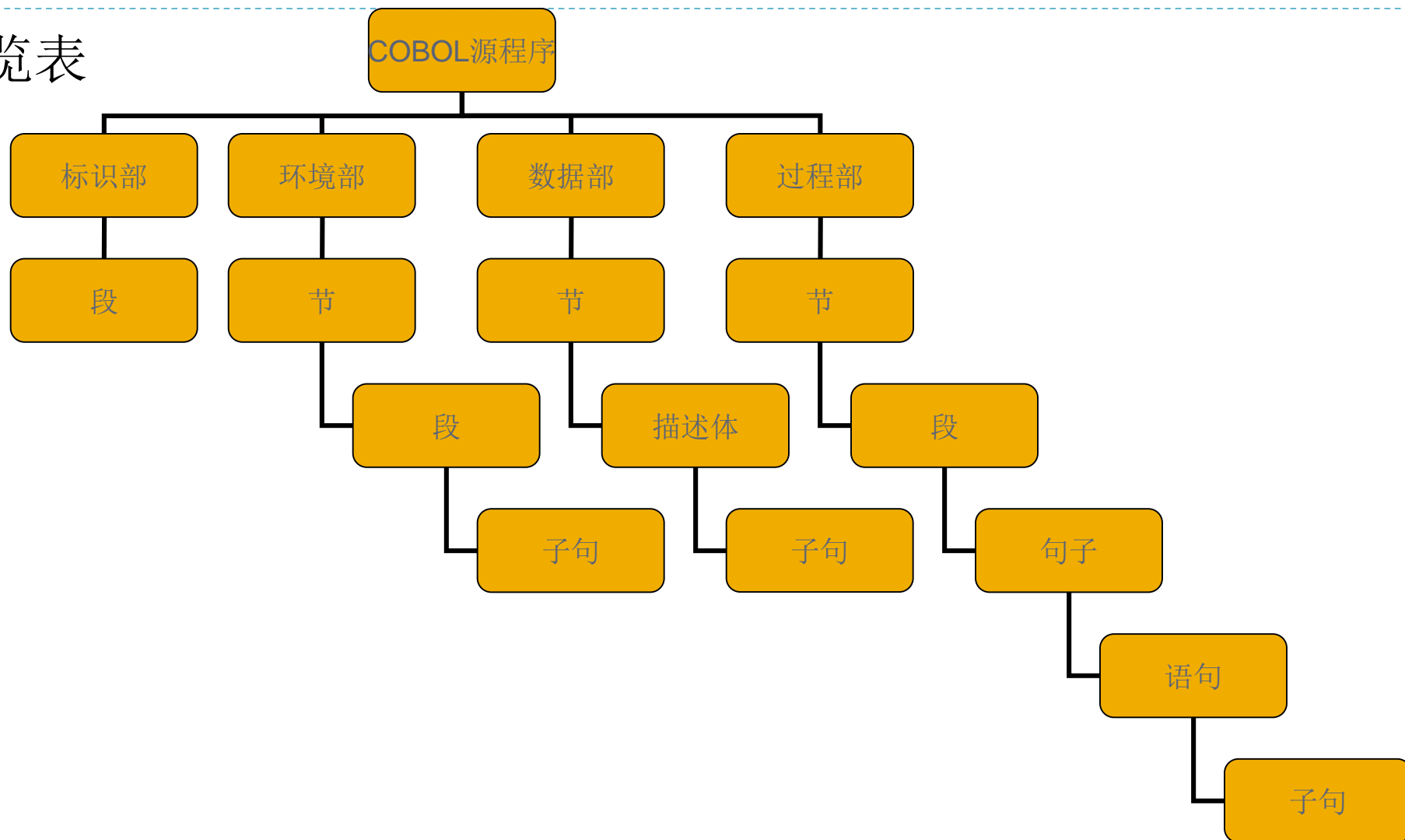
- ▶ 句子 (SENTENCE) 由语句 (STATEMENT) 组成，语句又由子句 (CLAUSE) 组成
- ▶ 句子以一个句号加一个以上的空格来结束
- ▶ 每个语句都是一条完整的指令，用相应的动词表示其操作
- ▶ 子句也有一个动词 (往往可省)，指定某一方面特定的功能





## § 1.3 COBOL程序的结构与书写格式

### 结构一览表



## § 1.3 COBOL程序的结构与书写格式

### ▶ 书写格式

#### ▶ ANSI格式

- ▶ 标准COBOL程序每行80列，被分为五个区域
- ▶ 第1—6列为 “标号区”
  - 标号是6位数字，应从小到大，但不一定连续
  - 标号只是方便查阅程序，可以不写
- ▶ 第7列为 “续行区”
  - 使用 “-” 表示本行是紧接在上一行后面
  - 续行要从第12列开始写
  - 使用 “\*” 表示本行是注释



## § 1.3 COBOL程序的结构与书写格式

### ▶ 书写格式

- ▶ 第8—11列为 “A区”
  - 程序中有些内容必须从该区开始书写，如：  
部头，节头，段头，层号01和77，文件描述符
- ▶ 第12—72列为 “B区”
  - 程序正文部分，过程部语句必须从该区开始书写
- ▶ 第73—80列为 “注释区”
  - 写入此区的内容为注释内容，编译时被舍去



## § 1.3 COBOL程序的结构与书写格式

### ► 书写格式

- ▶ 早期COBOL要求所有字母应大写，现在COBOL大小写等价，用引号括起来的字符串除外，如：

ADD A TO B = add A To b

DISPLAY 'HELLO' ≠ DISPLAY 'hello'

- ▶ 相邻的两个COBOL字之间有一个以上的空格
- ▶ 运算符和等号左右必须各有一个空格
- ▶ 圆括号外侧必须有一个空格，内侧不必，如：

A + (B + C) / D

- ▶ 逗号，句号，分号左边不能有空格，而右边应有

## § 1.4 COBOL字符、字、数据名与变量

### ► COBOL字符与字

- COBOL字符是指在程序中允许出现的字符  
包括**数字**，**大小写字母**及**15个专用符号**  
+, -, \*, /, =, 逗号, 句号, 分号, 引号, \$, (, ), <, >, 空格
- COBOL字是由上述字符组成的最小单位  
分为
  - 保留字**: 在COBOL已经规定专门用途的字
  - 用户字**: 用户自定义的名字  
如: 程序名, 文件名, 节名, 段名, 数据项名等

プログラム名は最初の8桁  
が識別される

## § 1.4 COBOL字符、字、数据名与变量

### ▶ 数据名

- ▶ 数据名相当于其他语言的变量名，代表一个具体的数据项
- ▶ 数据名长度为1—30个字符
- ▶ 只能由字母(至少一个)，数字和连字符“-”组成，连字符不能出现在两端，不能包含空格
- ▶ 不应用保留字作为数据名  
123, DECO. HENRY, OWEN-, 3R, DIVISION
- ▶ 尽量使用有意义的英文字或拼音，如：NAME, AGE, GONGZI
- ▶ 建议：多使用连字符，如：DEPTART-NUMBER



## § 1.4 COBOL字符、字、数据名与变量

### ▶ 常量

#### ▶ 数值常量

- 由正负号，小数点，数字0—9组成的序列

如：12300, 45.67, -89

- 小数点不能出现在常数右边，如：

MOVE 20. TO AGE

- 数值长度不超过18位
- 最少有一个数字
- 最多有一个正负号，且只能出现在最左边



## § 1.4 COBOL字符、字、数据名与变量

### ▶ 常量

#### ▶ 非数值常量

- 用引号括起来的字符串

如： ‘ABCD’ , ‘\$123’ , ‘HELLO WORLD’

- 由纯数字组成的非数值常量不能用于计算

‘123’ 和123不同

- 可以使用保留字, 如： ‘DATA’

- 可以写入引号, 如:

MOVE QUOTE ‘HENRY’ QUOTE TO NAME





## § 1.4 COBOL字符、字、数据名与变量

### ▶ 常量

#### ▶ 表意常量

- ZERO, ZEROS, ZEROES表示零字符
- SPACE, SPACES表示空格
- HIGH-VALUE, HIGH-VALUES表示具有最高值的字符(每个字符二进制为11111111)
- LOW-VALUE, 常LOW-VALUES表示具有最低值的字符(每个字符二进制为00000000)
- QUOTE, QUOTES表示引号
- ALL 常量 : 表示由该量组成的字符串



## § 1.5 COBOL基本逻辑语句

### ► 过程部的特点

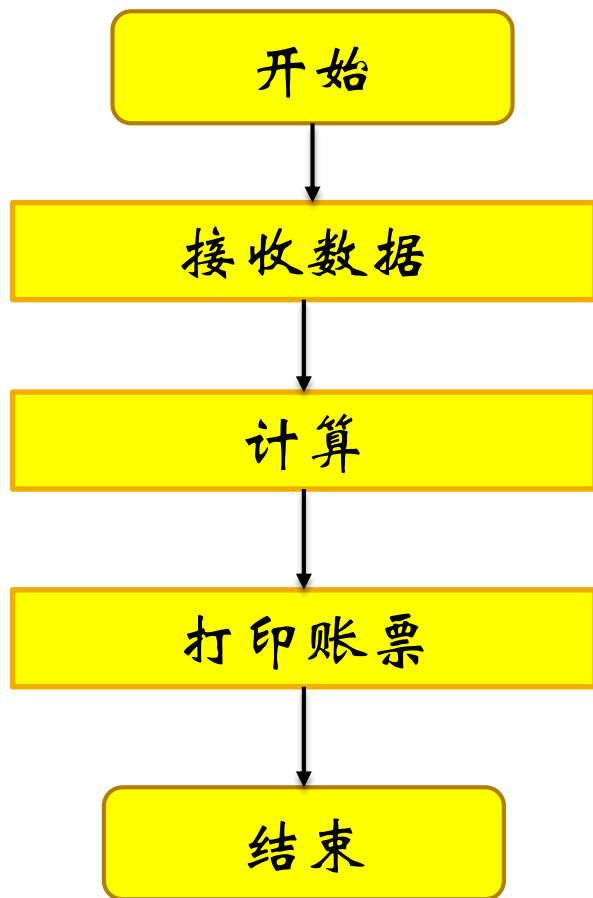
过程部是COBOL程序的**核心**，有以下四个特点

1. 过程部是程序的第四部分，以部头PROCEDURE DIVISION开头，必须从A区(8-11列)开始书写
2. 过程部的句子都以**动词**开始，如：DISPLAY, ADD，表示计算机应执行的操作
3. 动词后面一般要跟一个操作的对象，可以是数据名或文件名，如：MOVE **X** TO **Y**
4. 过程部的语句必须从B区(12列之后)开始书写



## § 1.5 COBOL基本逻辑语句

### ► 程序的流程



## § 1.5 COBOL基本逻辑语句

- ▶ 接收语句 ACCEPT
  - ▶ ACCEPT语句从键盘或指定设备获得少量的输入数据
  - ▶ 一般格式:  
ACCEPT **A** [ FROM B ]
  - ▶ ACCEPT 后只能有一个标识符! ACCEPT **A, B** 是错的
  - ▶ 标识符可以是组合项, 如: A包含A1, A2, A3  
可以ACCEPT A 统一输入 或者 ACCEPT A1... 分别输入



## § 1.5 COBOL基本逻辑语句

### ► 显示语句 DISPLAY

- DISPLAY语句将少量数据输出到指定的设备(显示器)上

- 一般格式:

DISPLAY 标识符 / 常量

- 每次执行DISPLAY都从新行开始, 如: DISPLAY A, B, C  
和分别执行DISPLAY A, DISPLAY B, DISPLAY C不同



## § 1.5 COBOL基本逻辑语句

▶ 停止语句 STOP



## § 1.5 COBOL基本逻辑语句

### ▶ 练一练

#### ▶ 例程1.5.1

力試し！頑張れ！

## § 1.5 COBOL基本逻辑语句

### ▶ 打开语句 OPEN

- ▶ 如果需要读写文件，必须先用OPEN语句打开该文件，系统在读写之前先检查该文件是否已经准备好

- ▶ 一般格式：

OPEN    INPUT   X1, X2    OUTPUT   Y1, Y2    EXTEND   Z1, Z2

- ▶ 可以用一个OPEN语句打开多个输入输出文件，X1, X2等是内部文件名





## § 1.5 COBOL基本逻辑语句

### ► 读语句 READ

- READ语句从外部文件读取数据到程序的数据项中

- 一般格式:

READ 内部文件名 [ RECORD ]

使用内部文件名便于记忆和程序移植，它与外部文件之间的关系应在环境部中预定义

- READ语句以记录为单位，必须在数据部单独定义存储单元存放从文件读入的信息，即输入文件记录区

### 例1.5.3

- 读入下一记录会将记录区内前一记录内容覆盖！

## § 1.5 COBOL基本逻辑语句

### ► 关闭语句 CLOSE

- 对一个文件的读写操作完成后应该用CLOSE语句关闭，使其不在涉及后续操作

- 一般格式：

CLOSE INPUT X1, X2 OUTPUT Y1, Y2 EXTEND Z1, Z2

- 可以简写：CLOSE X1, X2
- 文件CLOSE后如需再用，可以再次OPEN



## § 1.5 COBOL基本逻辑语句

### ► 读语句 READ

- READ语句从外部文件读取数据到程序的数据项中

- 一般格式:

READ 内部文件名 [ RECORD ]

使用内部文件名便于记忆和程序移植，它与外部文件之间的关系应在环境部中预定义

- READ语句以记录为单位，必须在数据部单独定义存储单元存放从文件读入的信息，即输入文件记录区 [例1.5.3](#)
- 读入下一记录会将记录区内前一记录内容覆盖！
- 使用 AT END 子句可以判断文件是否已经读取结束  
例如：READ IN-FILE AT END STOP RUN.
- 使用 INTO 子句可以将读取的记录保存到其它数据项  
例如：READ IN-FILE INTO TMP AT END STOP RUN.



## § 1.5 COBOL基本逻辑语句

### ▶ 练一练

#### ▶ 例程1.5.2

力試し！頑張れ！

## § 1.5 COBOL基本逻辑语句

### ► 写语句 WRITE

- WRITE语句将内存数据输出到外部设备(磁盘文件)

- 一般格式:

WRITE    **输出记录名**                      //注意!不是输出文件名!

- 与READ相似, WRITE语句以**记录**为单位, 必须在数据部单独定义存储单元存放要写入文件的信息, 即**输出文件记录区**
- 在WRITE输出之前, 应该先向记录区记录传送数据 [例1.5.4](#)
- 每次WRITE输出后不会自动换行, 而在下次输出前换行!
- 使用BEFORE和AFTER子句控制打印走纸

## § 1.5 COBOL基本逻辑语句

### ▶ 练一练

#### ▶ 例程1.5.3

力試し！頑張れ！

## § 1.5 COBOL基本逻辑语句

### ► 加法语句 ADD

- ADD A TO B  $B=A+B$ , A不变
- ADD B TO A
- ADD 10 TO A
- ADD A TO 10 ?
- ADD A, B TO C  $C=A+B+C$ , AB都不变
- ADD A, 10 TO C
- ADD A, B GIVING C  $C=A+B$ , AB都不变
- ADD A, 20 GIVING C
- ADD A, B TO C, D  $C=A+B+C$ ,  $D=A+B+D$
- ADD A, B GIVING C, D

## § 1.5 COBOL基本逻辑语句

### ▶ 减法语句 SUBTRACT

- ▶ SUBTRACT B FROM A                       $A=A-B$ , B不变
- ▶ SUBTRACT B, C FROM A
- ▶ SUBTRACT C, D FROM A, B
- ▶ SUBTRACT B, C FROM A GIVING D
- ▶ SUBTRACT A FROM 10                      ?
- ▶ SUBTRACT 5 FROM 10 GIVING A, B, C





## § 1.5 COBOL基本逻辑语句

### ▶ 乘法语句 MULTIPLY

- ▶ MULTIPLY A BY B  $B=A*B$ , A不变
- ▶ MULTIPLY 5 BY B
- ▶ MULTIPLY B BY 5 ?
- ▶ MULTIPLY A BY B GIVING C
- ▶ MULTIPLY B BY 5 GIVING C  $C=B*5$
- ▶ MULTIPLY A BY B, C  $B=A*B$ ,  $C=A*C$



## § 1.5 COBOL基本逻辑语句

### ▶ 除法语句 DIVIDE

▶ DIVIDE A INTO B  $B=B/A$ , A不变

▶ DIVIDE A INTO 5 ?

▶ DIVIDE A INTO 5 GIVING C

▶ DIVIDE A BY B GIVING C  $C=A/B$

▶ DIVIDE A BY B ?

▶ DIVIDE A BY 0 GIVING C ?



## § 1.5 COBOL基本逻辑语句

### ► 算术运算语句小结

四种算术运算的特点：

- ▶ 一个语句只能进行一种单一的运算
- ▶ 加法和减法可以进行两个以上数值量的计算，但乘法和除法只能在两个量之间进行
- ▶ 四种算术运算都有两种形式：带或者不带GIVING

练习1：从键盘获得一个整数N，逐步实现下列过程：

$N=N+3$ ；  $N=N/5$ ；  $N=N-2$ ；  $N=N*8$ ； 显示N的值

练习2：某工人每周工作五天，每天的工时累加得到一周总工时，乘以每小时工资得到一周总工资，再乘以应上交比例得到应扣除工资，最后从总工资中减去扣除工资得到实发工资



## § 1.5 COBOL基本逻辑语句

### ► 计算语句 COMPUTE

►  $Z = X^3 / (Y^3 - 1)$  需要几次运算？

► 使用COMPUTE语句可以简化四则运算：

COMPUTE Z = X \*\* 3 / ( Y \*\* 3 - 1 )

► 在该语句中可以使用：圆括号，正负号，乘方(\*\*)，乘除，加减；  
优先级自左向右递减

例：- 4 \*\* 2 / 2 - 1     结果是7

► COMPUTE语句运算速度慢

► **注意：**所有运算符两侧均至少留一个空格，圆括号内侧可以不留空格

## § 1.5 COBOL基本逻辑语句

### ▶ 练一练

#### ▶ 例程1.5.4

力試し！頑張れ！



## § 1.5 COBOL基本逻辑语句

### ▶ 传送语句 MOVE

- ▶ MOVE语句可以将数据从一个内存区传送到另一个内存区  
与WRITE语句不同！
- ▶ MOVE语句相当于赋值语句，可以将常量或一数据项的内容传送到另一数据项，例如：

```
MOVE  A   TO  B
```

```
MOVE  10   TO  COUNT
```

```
MOVE  ZERO TO  BLANKFIELD
```

```
MOVE  'IBM' TO  COMPANY
```

- ▶ 传送规则：A是发送项，B是接收项



## § 1.5 COBOL基本逻辑语句

### ▶ 传送语句 MOVE

- ▶ 如果A和B的数据类型及长度相同，则按字节相对应传送
- ▶ 如果A和B都是数值型，但长度不同，则按**小数点对齐**原则传送；如A比B长，则产生截断；如B比A长，则多余位补零。 [例2.4](#)
- ▶ 如果A和B的长度相同，且二者都是非数值型，则按**左侧对齐**原则；如A比B长，则从右端截断；如B比A长，则多余位补空格。
- ▶ A和B可以是初等项，也可以是组合项，或者两者混合，同样遵守以上原则



## § 1.5 COBOL基本逻辑语句

### ► 转移语句 GO TO

► GO TO 语句用来改变程序的执行顺序，程序执行到此将**无条件**转移到指定的标号（**段名**）

► 一般格式：

GO TO 标号

► 特殊格式：

GO TO 标号1，标号2... DEPENDING ON 标识符

如果标识符值等于1，则转移到GO TO语句之后的第一个标号，以此类推

例2.5

练习：运输公司规定，每次运送货物在50吨以下，按每吨10元收费；50至100吨按每吨9元；100至150吨按每吨8元；150至200按每吨7元。如果已知吨数不超过200吨，求运费。用GO TO写完整程序。





## § 1.5 COBOL基本逻辑语句

### ► 分支语句

- IF语句用来改变程序执行的顺序，是条件转移语句

- 例2.6.1

- 关系运算符：

IS GREATER THAN		>
IS LESS THAN		<
IS EQUAL TO		=
NOT GREATER THAN	≧	NOT >
NOT LESS THAN	≦	NOT <
NOT EQUAL THAN	≠	NOT =

## § 1.5 COBOL基本逻辑语句

### ► 分支语句

#### ► 关系预算规则:

a. 数值量之间按其**代数值**进行比较

$3 < 5$ ,  $10.5 = 10.50$ ,  $100 > -200$

b. 字母型数据之间**字典序**进行比较

$'x' < 'y'$        $'cat' > 'cap'$        $'c' < 'cobol'$

c. 字符型数据之间按其**编码规则**进行比较

ASCII编码:       $'9' < 'A'$

EBCDIC编码:     $'9' > 'A'$



## § 1.5 COBOL基本逻辑语句

### ► 分支语句

#### ► IF 条件 语句组

例1：某公司对顾客购买商品1000件以上给予3%优惠

IF QUANTITY IS NOT LESS THAN 1000

MULTIPLY 0.97 BY PRICE.

MULTIPLY QUANTITY BY PRICE GIVING TOTAL

#### ► IF 条件 语句组1 ELSE 语句组2

例2：某公司对顾客购买商品1000件以上给予3%优惠，1000件  
以下优惠1%

## § 1.5 COBOL基本逻辑语句

### ► 分支语句

#### ► IF中的句子和语句的区别

```
IF  A>0  
    DISPLAY A  
    ADD  A  TO  TOTAL.  
DISPLAY TOTAL.
```

```
IF  A>0  
    DISPLAY A.  
    ADD  A  TO  TOTAL.  
DISPLAY TOTAL.
```



## § 1.5 COBOL基本逻辑语句

### ► 分支语句

#### ► NEXT SENTENCE语句

例如：求 $AX^2+BX+C=0$ 的解

```
IF  B ** 2 - 4 * A * C NOT < 0
```

```
    NEXT SENTENCE
```

```
ELSE
```

```
    COMPUTE  X1 = .....
```

```
    COMPUTE  X2 = .....
```



## § 1.5 COBOL基本逻辑语句

### ► 停止语句 STOP

- ▶ 当程序实现预期要求后，应用STOP使其停止

- ▶ 一般格式：

STOP      RUN

- ▶ STOP    RUN应该是一个句子中**唯一的或最后一个**语句



### ▶ 数据部

- ▶ 数据部是COBOL源程序中的第三部分，也是唯一描述数据的部分，不可或缺，程序中涉及到的全部数据（输入，输出，中间）都要在此定义
- ▶ 数据有两类：
  - 孤立项**：两个相互独立，没有内在联系的数据项，各自占据内存区域
  - 组合项**：数据相互管理，它们之间在逻辑上存在联系（平等或从属），其各个数据项的数据类型可以不同
- ▶ 所有数据项都应在数据部中对其**属性**进行描述
  - a. 类型（数值/字符）和 存储形式（长度）
  - b. 数据项间的关系（层次和层号）
  - c. 记录与文件的关系
  - d. 文件的书写
- ▶ 用数据部将数据和数据“加工”过程分离，使任务单纯清晰，便于程序的书写，修改和阅读

### ► 数据的层次和层号

- COBOL中把有从属关系的数据用层次(level)关系来描述，数据的层次结构是：  
记录(record) -> 组合项(group item) -> 初等项(elementary item)

#### ► 例4.1.2

- 层次规定如下：
  - a. 描述层次结构的层号从01—49；01层最高，用来描述记录
  - b. 从属项的层号比其上属项的层号大，但层号不必连续
  - c. 如果多个数据项都从属于同一组合项但互不从属，则这几个数据项应具有相同的层号
  - d. 如果多个数据项都**不属于**同一组合项且互不从属，则这几个数据项可以有不同的层号
  - e. 一个层号为K的组合项包括它下面所有层号比它大的数据项，直到遇到小于或等于K的层次为止



### ► 数据部的结构

数据部通常用到的有以下几个节

- 文件节 (FILE SECTION)
- 工作单元节 (WORKING-STORAGE SECTION)
- 联接节 (LINKAGE SECTION)
- 报表节 (REPORT SECTION)



### ► 文件节的作用

- 程序中每个输入和输出文件都要在此描述，内容包括：
  - a. 文件名和文件属性
  - b. 文件中包括的记录的名字
  - c. 每个记录中数据的层次关系
  - d. 记录中各数据项的数据形式和占内存的大小

### ► 例4.2.1

### ► 文件节的作用

- 文件描述体用FD (FILE DESCRIPTION) 开始，而不是用层号开始，FD后面是在环境部中定义的内部文件名
- LABEL RECORD IS STANDARD  
只有磁盘(带)文件才有标号记录，且一律定义为‘标准的标号记录’；
- LABEL RECORD IS OMITTED  
打印文件是没有标号记录的，应定义为‘标号记录省略’
- DATA RECORD IS RECEIVABLE  
表示文件中包含的记录名是 RECEIVABLE，该项可省略



### ► 工作单元节的作用

- ▶ 程序中的数据项分为两部分：一部分用于输入或输出文件的，在数据部的文件节中加以描述；另一部分则用于非文件输入输出，如运算的中间结果等，则在工作单元节中描述。此外还可以在工作单元节为数据项赋初值。
- ▶ 工作单元中描述的数据项有两种形式
  - 初等项：以层号77开头
  - 组合项：以层号01开头

### ▶ 记录描述

- ▶ 记录描述体由01层号开头，后跟记录名

如： 01 RECEIVABLE.

- ▶ 例4.2.1中 RECEIVABLE记录包含四个初等项，每个初等项的数据类型和长度须单独定义

- ▶ 如果记录下面不再分项，即记录本身就是一个初等项，则可以定义成：

01 RECEIVABLE PIC X(80).



- ▶ 书写格式
  - ▶ 文件描述FD必须从A区开始书写
  - ▶ 层号01必须从A区开始书写
  - ▶ 其它层号可以从A或B区开始书写，为使层次清楚，最好从B区开始按层次关系写成锯齿形状，如：

```
01  A1.  
    02  B1.  
        03  C1  PIC .....  
        03  C2  PIC .....  
01  A2.  
    02  B2  PIC .....
```

### ► 数据项描述

- 在每个初等项的名字后使用 **PIC** 子句描述数据类型和长度
- PIC是PICTURE的缩写，用来描述初等数据项，它说明：
  - a. 数据是什么类型的，如果是数值型的，是否包含正负号和小数点？
  - b. 数据项占多大内存区
  - c. 是否需要准备有关特殊字符（ \$, +, -, \* 等 ）

### ► 例4.3.1

- 9999=9(4)                  AAAA=A(4)                  XXXX=X(4)



### ▶ 数值型数据的描述

- ▶ “9” 描述符：表示该位置可以放入一个0—9之间的数字

描述	数值	内存中表示
----	----	-------

77 X PIC 9	8	8
------------	---	---

77 Y PIC 9(5)	456	00456
---------------	-----	-------

- ▶ 注意：
  - 数值型数据项不能放入空格
  - 不能输入小数部分，如果输入则被舍弃
  - 不能输入负数，如果输入则负号被舍弃，即存入绝对值



### ▶ 数值型数据的描述

- ▶ ‘V’ 描述符：指出数据结构中隐含的小数点的位置，小数点不占内存单元，例如：

描述	数值	内存中表示
77 A PIC 9V9	7.7	77
77 B PIC 9(3)V9(2)	789	78900

### ▶ 注意：

- V在描述符最后等价于无小数点：99V = 99
- 传送数据时，按**小数点对齐**原则：多余数值位被截断，不足位补‘0’
- 运算时，按小数点位置对准进行运算
- 输出时，只显示内存中各字节的内容，不显示小数点



### ▶ 数值型数据的描述

#### ▶ ‘P’ 描述符

- a. 当数值很大，如:1000000000，需要用PIC 9999999999 来描述，共占用10个字节
- b. 使用PIC 9P(9)，P是隐含的不占内存单元，运算时按 $1 \times 10^9$ 进行只占用1个字节
- c. 对于很小的数，如：0.000012，可以描述为：PIC PPPP99

#### ▶ 注意：

- a. ‘P’ 必须出现在全部‘9’之前或之后，如：99P99 是错的
- b. ‘P’ 隐含指出小数点的位置：9PP=9PPV，VPP9=PP9，PPV9
- c. 对于很小的数，如：0.000012，可以描述为：PIC PPPP99’
- d. ‘9’ 后有n个‘P’，表示 $10^n$ ；‘9’ 前有n个‘P’，且有m个‘9’，表示 $10^{-(n+m)}$
- e. 输出时不显示零，只显示实际存放的数值



### ▶ 数值型数据的描述

- ▶ ‘S’ 描述符：指定带符号的数，正负号不占内存单元

描述	数值	内存中表示
77 A PIC S99V9	-7.7	077

- ▶ 注意：

- ‘S’ 必须是最左边的一个描述符
- ‘S’ 只用于运算，输出时不显示 ‘S’，输出的**最后一位数字特殊**

### ▶ 字母型数据的描述

- ▶ ‘A’ 描述符：指定字母型数据，只存放字母和空格
- | 描述           | 数值    | 内存中表示 |
|--------------|-------|-------|
| 77 T PIC AAA | ‘ABC’ | ABC   |

- ▶ 注意：

a. ‘A-B’ , ‘A. B’ , ‘C07’ 等都是非法的

- ▶ 练习：

文件FILEA中包含字符串 ‘THISISACOBOLPROGRAM’  
将其转换成 ‘THIS IS A COBOL PROGRAM’ 后输出到FILEB

## § 1.6 COBOL数据定义与编辑

### ▶ 字符型数据的描述

- ▶ ‘X’ 描述符：指定字符型数据  
描述

```
77 T PIC X(6) 'COBOL'
```

数值                      内存中表示  
COBOL+1个空格

- ▶ 注意：

- a. 可以用字母和数字描述符代替部分字符型数据

```
77 S PIC A(7)9(4)  
MOVE 'GERMENY2006' TO S
```

- b. ‘X’ 既可以存储字母型数据，也可以存储字符型数据

```
77 S PIC X(5).  
MOVE 'COBOL' TO S  
MOVE 123 TO S
```

- ▶ 注意：

- c. 传送字符型数据时，用引号括起来；传送数值型数据时不使用括号

```
MOVE '123' TO S  
MOVE 123 TO S
```



### ▶ 编辑型数据的描述

#### ▶ ‘.’ 描述符:

插入小数点，使数值型数据中隐含的小数点能在相应的位置上显示出来

```
77  A  PIC  99V99  
MOVE  12.34  TO  A  
DISPLAY  A  
//1234
```

```
77  B  PIC  99.99  
MOVE  12.34  TO  B  
DISPLAY  B  
//12.34
```

#### ▶ 前者占用内存区4个字节，后者占用5个字节

#### ▶ ‘.’ 描述符同样遵循小数点对其原则

## § 1.6 COBOL数据定义与编辑

### ▶ 编辑型数据的描述

- ▶ ‘,’ 描述符：  
插入逗号，用作分位符

```
77 A PIC 9999999  
1000000 TO B  
DISPLAY A  
//1000000
```

```
77 A PIC 9,999,999 MOVE 1000000 TO A MOVE  
DISPLAY B  
//1,000,000
```

- ▶ 前者占用内存区7个字节，后者占用9个字节
- ▶ ‘,’ 描述符：  
插入逗号，用作分位符

```
77 A PIC 9999999  
1000000 TO B  
DISPLAY A  
//1000000
```

```
77 A PIC 9,999,999 MOVE 1000000 TO A MOVE  
DISPLAY B  
//1,000,000
```

- ▶ 前者占用内存区7个字节，后者占用9个字节



### ▶ 编辑型数据的描述

- ▶ ‘0’ 描述符：在数值型数据后输出相应的 ‘0’

```
77  A  PIC  999PPP
```

```
MOVE 123000 TO A
```

```
DISPLAY A
```

```
//123
```

```
77  A  PIC  999000
```

```
MOVE 123000 TO B
```

```
DISPLAY B
```

```
//123000
```

- ▶ 前者占用内存区3个字节，后者占用6个字节



### ▶ 编辑型数据的描述

#### ▶ ‘B’ 描述符：插入空格

```
77  A  PIC  9(3)
```

```
77  B  PIC  B9(3)B
```

```
MOVE  789  TO  A
```

```
MOVE  A  TO  B
```

```
DISPLAY  A           //789
```

```
DISPLAY  B           // 789
```

#### ▶ 前者占用内存区3个字节，后者占用5个字节

### ▶ 编辑型数据的描述

- ▶ ‘+’ ‘-’ 描述符：在数值前后插入正负号

```
77 A PIC +99
```

```
77 B PIC -99
```

```
MOVE 10 TO A
```

```
MOVE -10 TO A
```

```
MOVE -10 TO B
```

```
MOVE 10 TO B
```

```
DISPLAY A //+10
```

```
// -10
```

```
DISPLAY B //-10
```

```
// 10
```

- ▶ 两者占用内存区3个字节
- ▶ 使用 ‘+’，不论数值为正或负，一律加符号
- ▶ 使用 ‘-’，数值为负时加负号，数值为正时前面加空格

### ▶ 编辑型数据的描述

#### ▶ ‘\$’ 描述符:

在数值前加\$符

```
77 A PIC 99
```

```
77 B PIC $99
```

```
MOVE 10 TO A
```

```
MOVE A TO B
```

```
DISPLAY A //10
```

```
DISPLAY B // $10
```

在数值前加正负号和\$符

```
77 A PIC +$99
```

```
77 B PIC -$99
```

#### ▶ 只能在数值数据(9)前加\$符

#### ▶ 正负号不能加到\$符后

### ► 编辑型数据的描述

#### ► 浮动插入 ‘\$’ 符:

```
77  A  PIC  $999.99
```

```
77  B  PIC  $$$9.99
```

```
MOVE  1.23  TO  A
```

```
MOVE  A  TO  B
```

```
DISPLAY  A      // $001.23
```

```
DISPLAY  B      //      $1.23
```

#### ► 练习1

#### ► 注意: 如果数值是一个小数, 则\$只能浮动到小数点的位置

### ► 编辑型数据的描述

- 浮动插入 ‘正负号’ :

```
77 A PIC +999.99
```

```
77 B PIC +++9.99
```

```
MOVE 1.23 TO A
```

```
MOVE A TO B
```

```
DISPLAY A //+001.23
```

```
DISPLAY B // +1.23
```

### ► 练习2

- 注意:

- 在编辑型数据的描述中, 指定浮动插入的字符个数应足够, 以免数据被截断
- 浮动字符前不能再出现其他符号, 如: **+\$\$.99** 是错的



### ► 编辑型数据的描述

- ‘Z’ 和 ‘\*’ 描述符：取消高位零，不加 \$, +, - 等符号

```
77 A PIC 999.99
```

```
77 B PIC ZZZ.99
```

```
77 C PIC ***.99
```

```
MOVE 1.23 TO A
```

```
MOVE A TO B
```

```
MOVE A TO C
```

```
DISPLAY A //001.23
```

```
DISPLAY B // 1.23
```

```
DISPLAY C //* *1.23
```

### ► 练习3

### ▶ 编辑型数据的描述

#### ▶ 注意:

a. 不能同时使用\$, +, - 浮动和 Z, \* 浮动  
如: **ZZ\$**9.9, **++\*\***9.9 都是错的

b. 单个的\$, +, - 可以和Z浮动  
如: +Z(3).99 , -\* (3).99 , \$ZZ99 都是正确的

c. 如果使' Z' 或' \*' 对应于所有字符, 如: ZZZ.ZZ或\*\*\*.\*\* , 当数值为0时, 则所有数值位全部由空格或\*代替, 小数点位也由空格代替或保留

77	A	PIC	ZZZ.ZZ	//六个空格
77	B	PIC	***.**	//***.**
77	C	PIC	***.99	//***.00



### ► 编辑型数据的描述

#### ► 注意:

- d. ‘Z’ , ‘\*’ 可以和 ‘,’ 一起使用, 但当插入的 ‘,’ 前面是被取消的无用零时, 该 ‘,’ 位置也被空格或 ‘\*’ 代替

```
77  A  PIC  *,***.**
```

```
77  B  PIC  *,***.**
```

```
77  C  PIC  Z,ZZZ.ZZ
```

```
MOVE 1234.56 TO A
```

```
MOVE 123.45 TO B
```

```
MOVE B TO C
```

```
DISPLAY A //1,234.56
```

```
DISPLAY B //**123.45
```

```
DISPLAY C // 123.45
```





### ▶ 编辑型数据的描述

#### ▶ ‘/’ 描述符：插入斜线

```
77  A  PIC  99/99/9999
```

```
MOVE  10012006  TO  A
```

```
DISPLAY  A           //10/01/2006
```

### ▶ 编辑型数据的描述

#### ▶ ‘DB’ 和 ‘CR’ 描述符:

在银行业务中，有时用到 DB (debit, 借方) 和 CR (credit, 贷方)

DB和CR 只能用作固定插入，而且只作为最后一个描述符；当数值为负时，将数据项最后两个字节置为DB或CR； 当数值为正时置为空格

```
77  A  PIC  $99.99DB
```

```
77  B  PIC  $99.99CR
```

```
MOVE  12.34  TO  A           //$12.34__ 两个空格
```

```
MOVE  -12.34  TO  A           //$12.34DB
```

```
MOVE  56.78  TO  B           //$ 56.78__
```

```
MOVE  - 56.78  TO  B           //$ 56.78 CR
```

### ► 编辑型数据的描述

- 编辑字符除了可用于数值型数据外，还可用于字符型数据；可用的字符只有' **B** ' 和 ' **0** ' 和 ' / '，插入空格和零字符

PIC	AAABAAAA	NEWYEAR
PIC	ABABABABAB	COBOL
PIC	X(4)BX(2)BX(2)	20022002
PIC	XX/XX/XXXX	06102006
PIC	X(5)B(3)	CHINAREN
PIC	00X(6)00	PEOPLE

### ► PIC子句小结

- COBOL中用编辑字符来描述数据，主要为打印输出服务。通过对数据进行编辑，使输出结果符合会计，银行，统计等行业习惯的形式，使打印报表清晰，灵活，易懂，这是COBOL语言比其他语言优越的地方

► 格式：      层号    数据项名    PICTURE    IS    描述符

► 数据类型                      可以使用的描述符

数值型                      9 V S P

字母型                      A

字符型                      9 A X

编辑数值型                      9 V P . , B Z + - \$ \* 0 / DB CR

编辑字符型                      9 A X B 0 /

### ► 数据的初始化

- ▶ 程序中某些数据项需要赋初值，例如用于循环的累计数据项，或事先置空的字符数据项

- ▶ 可以使用MOVE语句为变量赋值

```
MOVE 0 TO A
```

```
MOVE SPACE TO B
```

- ▶ COBOL允许在描述数据项的时候赋初值，如：

[例1.6.2](#)



### ► 数据的初始化

#### ► 注意:

- a. 只有工作单元节的数据项可以赋初值，而文件节的数据项不能
- b. 如果在组合项的描述体中使用VALUE子句，初值只能是标以常量

或非数值型常量，如：

```
01  A  VALUE  '1234' .
```

```
01  A  VALUE  1234.
```

```
02  A1  PIC  99.          //12
```

```
02  A2  PIC  99.          //34
```

A1, A2可进行数值运算

- c. 当用一个带符号的数值作初值时，相应的PIC子句中应有‘S’描述符  
否则初值无效

```
77  N  PIC  S99  VALUE -22.
```



### ► 数据的初始化

#### ► 注意:

- d. 赋初值应注意类型的一致性，表意常量既可作为数值常量又可作为非数值常量

```
77 A PIC X(4) VALUE 123 //错的, 123是数值型
```

```
77 B PIC X(4) VALUE '123'
```

```
77 C PIC 9(3) VALUE ZERO //数值0, 可用于计算
```

```
77 D PIC X(3) VALUE ZERO //000
```

- e. 初值应适合PIC子句描述的范围，否则会出现截断或产生错误

```
77 E PIC S99 VALUE 45.6 //E=45
```

```
77 F PIC 9(2) VALUE 345 //F=45
```

```
77 G PIC X(4) VALUE 'COBOL' //G= 'COBO'
```



### ▶ 内存中数据的存储

- ▶ 区域图用来形象的表示数据在内存中的存储情况，使用户对数据部中数据的描述和过程部中语句的执行有一个形象的概念

- ▶ 输入记录区

```
FD  IN-FILE  LABEL  RECORD  IS  STANDARD.
```

```
01  IN-RECORD.
```

```
    02  PRODUCT-NUM    PIC    9(6).
```

```
    02  PRODUCT-NAME   PIC    X(10).
```

```
    02  PRODUCT-PRICE  PIC    99V99.
```





### ▶ 内存中数据的存储

#### ▶ 输出记录区

```
FD  OUT-FILE  LABEL  RECORD  IS  STANDARD.
```

```
01  OUT-RECORD.
```

```
    02  PD-NUM    PIC    9(6).
```

```
    02  PD-NAME   PIC    X(10).
```

```
    02  PD-PRICE  PIC    99V99.
```

```
    02  PD-DISCOUNTED-PRICE PIC  99V99
```

#### ▶ 工作单元区

```
77  DISCOUNT  PIC  9V99  VALUES  0.75
```



## § 1.7 COBOL的标识部和环境部

### ► 标识部

标识部是COBOL程序的第一部分，用来为程序设定标志，以便识别，每个程序（包括主程序和子程序）都必须有一个名字，系统按名字对程序管理和调用

标识部还可以包含文档记录信息，作备望用。如：作者，写程序的日期，保密程度等

标识部下面不设置节，只设置段，**部头**和**段头**都从**A区**开始书写

#### ► 书写格式：

```
IDENTIFICATION    DIVISION.
```

```
ID    DIVISION.
```

```
PROGRAM-ID.      program-name.
```

- 应有意义或按照公司规定的命名规则起名
- 只能使用字母，数字和连字符
- 最少一个字母，不能以连字符开头或结尾
- 最长30个字符



## § 1.7 COBOL的标识部和环境部

### ► 标识部的任选部分

- ▶ AUTHOR.  
记录程序员姓名，可增加荣誉感
- ▶ INSTALLATION.  
指定设计该程序的公司或部门
- ▶ DATE-WRITTEN  
记录每次程序编写，修改日期
- ▶ DATE-COMPILED  
记录程序被编译的日期
- ▶ SECURITY  
列出谁有权访问该程序，该段只是记录并不实际保护代码



## § 1.7 COBOL的标识部和环境部

### ► 环境部

- 环境部用来说明程序运行的软硬件环境，是COBOL程序中唯一与设备相关的部分，将程序中用到的内部文件与外部设备建立起联系

- 环境部包括两个节：配置节和输入输出节

- 一般格式：

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
.....  
INPUT-OUTPUT SECTION.  
.....
```

- 配置节和输入输出节都是可选的，当使用外部文件时必写输入输出节

## § 1.7 COBOL的标识部和环境部

### ► 环境部—配置节

配置节包含三个段

- SOURCE-COMPUTER. computer-name.  
标识对该COBOL程序进行编译的计算机
- OBJECT-COMPUTER. computer-name.  
标识运行该COBOL程序的计算机
- SPECIAL-NAMES.  
用来通知系统把原来规定的设备名或符号改为用户自己指定的名字或符号

CURRENCY SIGN IS '\$'      \$是默认货币符号，可使用非数值常量  
DECIMAL-POINT IS COMMA      用逗号代替小数点

注1：SOURCE-COMPUTER和OBJECT-COMPUTER可以是不同的计算机，但必须兼容

注2：若主程序中已设置了配置节，则子程序不必再设置

## § 1.7 COBOL的标识部和环境部

### ► 环境部—输入输出节

- ▶ 程序中如果用到输入输出文件，就在该节把程序中的内部文件和外部设备联系起来

- ▶ 输入输出节包括两个段

FILE-CONTROL.                      文件控制段

I-O-CONTROL.                      输入输出控制段

- ▶ 只要用到INPUT-OUTPUT节，就必写FILE-CONTROL段
- ▶ I-O-CONTROL段指定目标程序运行时多个文件共用一个内存区以节省内存

## § 1.7 COBOL的标识部和环境部







- § 5.1 MOVE语句
- § 5.2 算术运算语句
- § 5.3 IF语句
- § 5.4 程序举例
- § 5.5 字符串连接语句—STRING
- § 5.6 字符串分解语句—UNSTRING
- § 5.7 检测语句—INSPECT
- § 5.8 转换语句—TRANSFORM



§ 5.1.1 各种数据类型之间的传送

§ 5.1.2 组合项的传送

§ 5.1.3 对应传送



- ▶ 数值型数据之间传送，按小数点对齐原则。如发送项长于接收项，则多余位截断；反之则接受项空位补零
- ▶ 字母或字符型数据之间传送，按左侧对齐原则。如发送项长于接收项，则右端多余位截断；反之则右端补空格
- ▶ 发送项是数值型，接收项是编辑数值型数据，则先将发送项中数据按接收项的描述要求进行编辑，然后再传送。

不能由编辑型向数值型数据传送！

例： 77 A PIC 9(4)V99  
77 B PIC \$(6)V99  
MOVE A TO B //正确  
MOVE B TO A //错误



## § 5.1.1 各种数据类型之间的传送



### ▶ 不同类型数据间非法的传送

- a. 数值编辑项，字符编辑项，SPACE，字母型数据项不能传送给  
数值型和数值编辑型数据项
- b. 数值常量，ZERO，数值数据项，数值编辑项不能传送给  
字母数据项
- c. 非整数的数值数据项或非整数的数值常量不能传送给字符型或  
字符编辑型数据项，如：  
77 A PIC 99V99  
77 B PIC X(4)  
MOVE A TO B //错误  
MOVE 1000 TO B //错误



### ▶ 不同类型数据间合法的传送

- a. 接收项为字符或字符编辑型，发送项长于接收项，按左侧对齐原则

```
77 A PIC 9(4)
```

```
77 B PIC X(4)
```

```
MOVE A TO B //正确
```

若发送项的描述符带符号，符号不予传送

```
77 A PIC S99 VALUE -123
```

```
77 B PIC X(4)
```

```
MOVE A TO B //B=123+空格
```

- b. 接收项是数值或数值编辑型，可以接收数值型以及内容全是数字的字符型数据项，[例5.1.1](#)
- c. 接收项是字母型，可以接收字母型以及内容全是字母和空格的字符型数据项



### ▶ 最常见的传送类型

- a. 同类型数据间传送
- b. 数值型向数值编辑型的传送，便于输出
- c. 各类型数据项(不包括非整型)向字符型数据项的传送



表 5.1

发 送 项 \ 接 收 项		数 值 型		数值编辑型	字母型	字符型	字符编辑型	组合项
		整 数	非整数					
数 值 型	整 数	✓	✓	✓	×	✓	✓	✓
	非整数	✓	✓	✓	×	×	×	✓
编辑数值型		×	×	×	×	✓	✓	✓
字 母 型		×	×	×	✓	✓	✓	✓
字 符 型		△	△	△	△	✓	✓	✓
编辑字符型		×	×	×	×	✓	✓	✓
数值常量		✓	✓	✓	×	×	×	✓
非数值常量		×	×	×	✓	✓	✓	✓
ZERO		✓	✓	✓	×	✓	✓	✓
SPACE		×	×	×	✓	✓	✓	✓
组 合 项		△	△	△	△	✓	✓	✓





- 发送项和接收项都是组合项，而且结构和描述均相同，则可以看作将各个初等项一一对应传送

01 A.

02 A1 PIC X(3) VALUE 'A07' .

02 A2 PIC 9(2)V9 VALUE 12.3.

02 A3 PIC A(3) VALUE 'CDE' .

01 B.

02 B1 PIC X(3).

02 B2 PIC 9(2)V9.

02 B3 PIC A(3).

MOVE A TO B                    //B=A07 123 CDE





▶ 发送项和接收项长度相同，但数据结构形式不同，则将发送项的内容原样不动地自左而右顺序地传送到接收项

01 A.

02 A1 PIC X(3) VALUE 'A07' .

02 A2 PIC 9(2)V9 VALUE 12.3.

02 A3 PIC A(3) VALUE 'CDE' .

01 B.

02 B1 PIC X(2).

02 B2.

03 B21 PIC X(2).

03 B22 PIC X(3).

02 B3 PIC A(2).

MOVE A TO B                    //B=A0 71 23C DE



- ▶ 组合项作为一个整体，不是数值型的，一般情况下可以当作字符型

```
01  A.  
    02  A1  PIC  99  VALUE  12.  
    02  A2  PIC  99  VALUE  34.  
    02  A3  PIC  99  VALUE  56.
```

```
ADD  1  TO  A1  
SUBTRACT 10  FROM  A2  
ADD  A1  TO  A3
```

```
ADD  10  TO  A      //错误
```



### ▶ 数据名的受限和受限名的传送

01 TODAY-DATE.

03 YEAR PIC 9(4).

03 MONTH PIC 99.

03 DAY PIC 99.

01 LAST-DATE.

03 YEAR PIC 9(4).

03 MONTH PIC 99.

03 DAY PIC 99.

MOVE 2006 TO YEAR. // ???

MOVE 2006 TO YEAR OF TODAY-DATE



## § 5.1.3 对应传送



- ▶ 如果限定一次还不能成为唯一，可以多次使用限定符

01 TODAY-DATE.

02 A.

03 YEAR PIC 9(4).

03 MONTH PIC 99.

03 DAY PIC 99.

01 LAST-DATE.

02 A.

03 YEAR PIC 9(4).

03 MONTH PIC 99.

03 DAY PIC 99.

MOVE 2006 TO YEAR OF A OF TODAY-DATE



► COBOL提供**对应传送**功能，即把一个组合项中若干项传送给另一组合项中同名的项

格式：MOVE [ CORRESPONDING ] 标识符1 TO 标识符2  
[ CORR ]

MOVE **CORR** TODAY-DATE TO LAST-DATE

01 TODAY-DATE.

03 YEAR PIC 9(4).

03 MONTH PIC 99.

03 DAY PIC 99.

01 LAST-DATE.

03 YEAR PIC 9(4).

03 MONTH PIC 99.

03 DAY PIC 99.



- 如果两个组合项中包括的项不同，则只传送同名的项

MOVE CORR PAY-RECORD TO TAK-RECORD

01 PAY-RECORD.		01 TAK-RECORD.
02 A1 PIC 9(4).		02 B1 PIC Z(4).
02 A2 PIC 9(4).	→	02 A2 PIC Z(4).
02 A3 PIC 9(4).	→	02 A1 PIC Z(4).

相当于：

```
MOVE  A1  OF  PAY-RECORD  TO  A1  OF  TAK-RECORD
MOVE  A2  OF  PAY-RECORD  TO  A2  OF  TAK-RECORD
```



## § 5.1.3 对应传送



- ▶ 传送的两者间必须有成对的同名数据项，而且这一对中必须至少有一个项是初等项，否则不能作为对应项传送

```
MOVE CORR A1 TO A2
```

```
01 A1.
```

```
02 B PIC X(2).
```

```
02 C.
```

```
03 C1 PIC X(4).
```

```
03 C2 PIC X(5).
```

```
01 A2.
```

```
03 B PIC X(2).
```

```
03 C.
```

```
05 C3 PIC X(4).
```

```
05 C4 PIC X(5).
```

相当于：

```
MOVE B OF A1 TO B OF A2.
```

C即使同名，记录也不传送



▶ 将A2记录的结构改变：

```
MOVE CORR A1 TO A2
```

```
01 A1.
```

```
    02 B PIC X(2).
```

```
    02 C.
```

```
        03 C1 PIC X(4).
```

```
        03 C2 PIC X(5).
```

```
01 A2.
```

```
    03 B PIC X(2).
```

```
    03 C PIC X(9).
```

相当于：

```
MOVE B OF A1 TO B OF A2.
```

```
MOVE C OF A1 TO C OF A2.
```





## § 5.1.3 对应传送

▶ 所谓同名，指的是它们有相同的全程受限（全程同名）

MOVE CORR A1 TO A2

01 A1.

02 X.

03 X1 .....

03 X2 .....

03 X3 .....

02 Z1.

04 B1 .....

04 B2 .....

01 A2.

02 X.

03 X1 .....

03 X3 .....

03 X4 .....

02 Z2.

04 B1 .....

04 B2 .....

▶ B1和B2不是全程同名，因此不是对应项，不会对应传达



- § 5.2.1 四舍五入处理
- § 5.2.2 长度溢出处理
- § 5.2.3 对应项间的运算
- § 5.2.4 除法中的余数子句



在四则运算中，计算结果可能比接收项允许的长度长，则发生截断

计算结果	接收项描述	接收项内容	说明
123.456	9V9	3.4	整个长度不够
1234.56	99v99	34.56	整数位不够
12.34	99V9	12.3	小数位不够

- 为提高精度，使用ROUNDED子句对截断后的一位按四舍五入处理  
例如：

ADD A, B TO C ROUNDED

A+B+C 值	C 描述	C 值
186.74	999	187
186.74	999V9	186.7

- 如果计算结果有多个，则应分别说明哪一个接收项要进行舍入处理  
例如：

ADD A, B, C TO D, E ROUNDED, F



- ▶ 计算结果的整数部分的长度如果比结果数据项描述所规定的整数部分长，则称**长度溢出**

例如：

```
77  A  PIC  9V9  VALUE  1.2
```

```
77  B  PIC  9V9  VALUE  9.0
```

```
77  C  PIC  9V9
```

```
MULTIPLY  A  BY  B  GIVING  C          //08
```

- ▶ 小数部分截断只影响数值近似程度，而整数部分超长会导致结果完全错误，不应继续运算



- ▶ ON SIZE ERROR子句提供溢出处理，当发生溢出时按程序设计者指定的操作进行处理

例如：

```
77  A  PIC  9V9  VALUE  1.2
77  B  PIC  9V9  VALUE  9.0
77  C  PIC  9V9
MULTIPLY  A  BY  B  GIVING  C  ON  SIZE  ERROR
      DISPLAY  'SIZE  ERROR'
      STOP  RUN.
```

- ▶ ON SIZE ERROR 子句相当于一个IF语句



▶ 使用ON SIZE ERROR子句时，如果发生溢出，错误的结果不存入数据项；若有多个计算结果，未发生溢出的正常保存，发生溢出的结果不保存

例如：

```
77  A  PIC  9      VALUE    4
```

```
77  B  PIC  9      VALUE    7
```

```
77  C  PIC  9      VALUE    8
```

```
77  D  PIC  99     VALUE   10
```

```
ADD  A,  B  GIVING  C,  D  ON  SIZE  ERROR
```

```
    DISPLAY  C                      //8
```

```
    DISPLAY  D                      //11
```

```
GO   TO  S-END.
```



▶ 当ROUNDED与ON SIZE ERROR同时使用时，先作四舍五入处理再判断是否溢出

例如：

```
77  A  PIC  9V9    VALUE    4.5
```

```
77  B  PIC  9V9    VALUE    5
```

```
77  C  PIC  9      VALUE    6
```

```
ADD  A,  B  GIVING  C
```

```
ADD  A,  B  GIVING  C  ROUNDED
```

```
ADD  A,  B  GIVING  C  ON  SIZE  ERROR  .....
```

```
ADD  A,  B  GIVING  C  ROUNDED  ON  SIZE  ERROR  .....
```

//9096





## § 5.2.3 对应项间的运算

- ▶ ADD和SUBTRACT语句可以用来对组合项中的对应项进行加减运算

格式：ADD [ CORRESPONDING ] 标识符1 TO 标识符2  
[ CORR ]

例如：

01 R1.	01 R2.
02 A1 PIC 9(2).	03 A1 PIC 9(2).
02 A2 PIC 9(2).	03 A2 PIC 9(2).
02 A3 PIC 9(2).	03 A3 PIX X(2).

ADD CORR R1 TO R2

相当于：

ADD A1 OF R1 TO A1 OF R2  
ADD A2 OF R1 TO A2 OF R2



## § 5.2.3 对应项间的运算



▶ 注意：

- a. 进行运算的各项必须是数值型初等项
- b. 只有加法和减法语句可以有CORR子句，而乘法，除法以及计算语句不能使用



## § 5.2.4 除法中的余数子句

▶ 使用REMAINDER子句可以将除法的余数保存到指定的数据项

例1:

```
DIVIDE 1.5 INTO 7 GIVING A REMAINDER B
```

例2:

```
77 A PIC 9V9 VALUE 6.0
```

```
77 B PIC 99V9 VALUE 16.3
```

```
77 C PIC 9.99
```

```
77 D PIC 9.99
```

```
DIVIDE A INTO B GIVING C REMAINDER D
```

```
DISPLAY C //2.71
```

```
DISPLAY D //0.04
```



## § 5.2.4 除法中的余数子句

- ▶ 商和余数的值不仅取决于被除数和除数，还取决于数据部中对商和余数的描述  
例如：

```
77  A  PIC  9V9    VALUE  6.0
```

```
77  B  PIC  99V9   VALUE  16.3
```

```
77  E  PIC  9.9
```

```
77  F  PIC  9.9
```

```
DIVIDE  A  INTO  B  GIVING  E  REMAINDER  F
```

```
DISPLAY  E                //2.7
```

```
DISPLAY  F                //0.1
```

## § 5.2.4 除法中的余数子句

- 如果有ROUNDED子句，则对商四舍五入，而余数不作处理  
例如：

```
77  A  PIC  9V9    VALUE  6.0
```

```
77  B  PIC  99V9   VALUE  16.3
```

```
77  C  PIC  9.99
```

```
77  D  PIC  9.99
```

```
DIVIDE  A  INTO  B  GIVING  C  ROUNDED  REMAINDER  D
```

```
DISPLAY  C                //2.72 (2.716四舍五入)
```

```
DISPLAY  D                //0.04
```

## § 5.2.4 除法中的余数子句

- ▶ 长度溢出只对商进行检查，而不检查余数。如果商值溢出，则执行ON SIZE ERROR子句，**商值不放入数据项，但余数放入数据项**

例如：

```
77  A  PIC  9      VALUE  8
77  B  PIC  9(4)   VALUE  1006
77  C  PIC  99.9
77  D  PIC  9. 9
```

```
DIVIDE  A  INTO  B  GIVING  C  REMAINDER  D
        ON  SIZE ERROR  GO  TO  S1.
```

```
DISPLAY  C           //结果125.7，C溢出仍为原值
DISPLAY  D           //0.4
```



- § 5.3.1 IF语句的嵌套
- § 5.3.2 关系表达式条件
- § 5.3.3 符号条件
- § 5.3.4 类型条件
- § 5.3.5 条件名条件
- § 5.3.6 复合条件



## § 5.3.1 IF语句的嵌套

- ▶ 例1：如果 $1000 < Q < 2000$ ，显示Q值

```
IF  Q>1000
    IF  Q<2000
        DISPLAY  Q.
```

- ▶ 例2：当金额AMOUNT $\leq 50$ 时，利率RATE=0.02；

当 $50 < \text{AMOUNT} < 100$ 时，RATE=0.03；

当 $\text{AMOUNT} \geq 100$ ，RATE=0.04

```
IF  AMOUNT <100
    IF  AMOUNT > 50
        MOVE  0.03  TO  RATE
    ELSE
        MOVE  0.02  TO  RATE
ELSE
    MOVE  0.04  TO  RATE
```





## § 5.3.1 IF语句的嵌套

- ▶ 例3：如果IF与ELSE个数不等，应由内向外逐层匹配

```
IF  X = Y  
    IF  A = B  
        DISPLAY  'YES'  
    ELSE DISPLAY  'NO' .
```

- ▶ 例4：嵌套式IF结构中，只能在最外层的选择结构中使用一个句点

IF  A > 100	IF  A > 100
DISPLAY  A.	DISPLAY  A
IF  B > 100	IF  B > 100
DISPLAY B.	DISPLAY B.





## § 5.3.2 关系表达式条件

客体 主体	数值型	数值常量	非数值常量	字母型	字符型	组合项
数值型	N	N	C	C	C	C
数值常量	N	×	×	C	C	C
非数值常量	C	×	×	C	C	C
字母型	C	C	C	C	C	C
字符型	C	C	C	C	C	C
组合项	C	C	C	C	C	C

N: 数值型

C: 字符型

×: 不能比较



▶ 用来检查某数据项的值的代数符号格式：

数据名 IS [NOT] { POSITIVE | NEGATIVE | ZERO }

IF X IS POSITIVE      等价于      IF X > 0  
    DISPLAY X.                      DISPLAY X.

▶ 符号条件用来作定性的测定，如：商品的库存量不应为负，库存为零表示脱销，银行信用卡业务中负值表示透支



- 检查数据项的类型是否符合指定的要求，即数据项的内容是否全为数字或字母格式：

数据名 IS [NOT] { NUMERIC | ALPHABETIC }

77 T PIC A(4) VALUE 'WANG'

77 Q PIC X(5) VALUE '0S390'

77 R PIC 9(3) VALUE 789

IF T IS ALPHABETIC DISPLAY T.

IF Q IS NOT NUMERIC DISPLAY Q.

IF R IS NUMERIC ADD 1 TO R.

- ▶ 用 ‘9’ 描述符定义的数据项不能用ALPHABETIC比较
- 用 ‘A’ 描述符定义的数据项不能用NUMERIC比较
- 用 ‘X’ 描述符定义的数据项既能用NUMERIC，也能用ALPHABETIC

```
77 X PIC 99 VALUE 88
```

```
77 Y PIC AA VALUE ‘QQ’
```

```
77 Z PIC XX VALUE ‘02’
```

```
IF X IS ALPHABETIC ..... //错误
```

```
IF Y IS NOT NUMERIC ..... //错误
```

```
IF Z IS NOT ALPHABETIC
```

```
    IF Z IS NOT NUMERIC
```

```
        DISPLAY ‘ERROR INPUT’ .
```



▶ 例如：

为鼓励存款，存款数小于1000元，利息为2%；

大于等于1000元但小于10000元，利息为3%；

大于等于10000元但小于50000元，利息为4%；

大于等于50000元但小于100000元，利息为5%

▶ 简单的说，条件名就是用一个数据名代表一个条件；使用四个名字代表四个区间

77 X PIC 9(5).

88 X1 VALUE 0 THRU 999.

88 X2 VALUE 1000 THRU 9999.

88 X3 VALUE 10000 THRU 49999.

88 X4 VALUE 50000 THRU 99999.



- ▶ X是条件变量，定义为数值变化的范围
- ▶ X1—X4是条件名，紧跟在条件变量之后，必须用层号88定义
- ▶ VALUE子句的作用不是赋初值，而是为条件变量的一个可能值命名
- ▶ 对于 “88 X1 VALUE 0 THRU 999” 应理解为：当X的值在0—999时，条件X1为“真”

- ▶ 在过程部可以直接使用条件名条件

IF X1 MOVE 0.02 TO RATE.

IF X2 MOVE 0.03 TO RATE.

IF X3 MOVE 0.04 TO RATE.

IF X4 MOVE 0.05 TO RATE.





- ▶ 复合条件是由若干个简单的“条件”组合而成的条件
- ▶ 使用逻辑运算符 AND(与), OR(或), NOT(非) 连接多个条件  
例如:

IF A>1000 AND A<10000

- ▶ 如果在一个IF语句中同时用到AND, OR, NOT, 其优先级是:

NOT > AND > OR

例如:

X=3.5 Y=4 T=3 W=2 C=5 D=4 G=8, 求:

IF X=Y OR NOT T=W AND C>D AND G IS POSITIVE



▶ 例1:

统计磁盘文件中学生记录男女生的总数。

学生记录包括学号，姓名，性别，年龄，班级



### ▶ 例2:

人事部门统计职工受教育程度。凡上学年数<12年的，表示高中以下；等于12年的为高中毕业；13—15年为大学肄业；16年为大学毕业；17—19年为研究生。职工数据存储于磁盘文件中，包括员工编号，姓名，受教育年数。

输出不同程度受教育的人数



- ▶ STRING语句用来将多个非数值型数据项的值连接后送到一个接收数据项中，在合并过程中可以删除某些指定的字符

例如：

```
77  A  PIC  X(4)  VALUE  'ABC'  
77  B  PIC  X(4)  VALUE  'JKL'  
77  C  PIC  X(4)  VALUE  'XYZ'  
77  D  PIC  X(16)
```

```
STRING  A, B, C  DELIMITED  BY  SIZE  INTO  D  
// D= 'ABC_JKL_XYZ_____'   (末尾五个空格)
```

DELIMITED BY SIZE 是按发送项的长度全部传送到接收项



## § 5.5 字符串连接语句—STRING



- ▶ 可以将发送项空格之前的字符串传送到接收项

例如：

```
STRING  A, B, C  DELIMITED  BY  SPACE  INTO  D
// D= 'ABCJKLXYZ_____'   (末尾七个空格)
```

- ▶ 可以使用其它字符作为定界符，各个发送项可以使用不同的定界符

例如：

```
STRING  A  DELIMITED  BY  'B'
        B  DELIMITED  BY  'L'
        C  DELIMITED  BY  'M'  INTO  D
// D= 'AJKXYZ_____'   (末尾十个空格)
```



▶ 定界符可以是字符串

```
STRING  A  DELIMITED  BY  'BC'  
        B  DELIMITED  BY  'LP'  
        C  DELIMITED  BY  'ZZ'  INTO  D  
// D= 'AJKL_XYZ_____' (末尾八个空格)
```

▶ 可以在传送中插入所需字符

例如：

```
STRING  A,  '*', B,  '*', C  DELIMITED  BY  SPACE  INTO  D  
// D= 'ABC*JKL*XYZ_____' (末尾五个空格)
```



- ▶ 如果不想从接收项的最左端开始接收字符，可以使用POINTER短语指定从某一字符位开始接收字符

```
MOVE 3 TO P.
```

```
STRING A, B, C DELIMITED BY SPACE
```

```
WITH POINTER P INTO D.
```

```
// D= ‘__ABCJKLXYZ_____’ (头部两个，末尾五个空格)
```

- ▶ 如果接收项的字符个数不足，则发生“溢出”，可以进行溢出处理  
例如：

```
77 D PIC X(10)
```

```
STRING A, B, C DELIMITED BY SIZE INTO D
```

```
ON OVERFLOW DISPLAY ‘OVERFLOW’ .
```

```
// D= ‘ABC_JKL_XY’
```



▶ 注意：

- a. 接收数据项必须是初等项
- b. 指针项必须是一个整型的初等项
- c. STRING语句结束后，接收项中未送入的字符位置上  
保持原有内容，而不是自动设置空格





- UNSTRING语句将一个发送字符串拆成若干个接收字符串，是STRING语句的逆操作  
例如：

```
77  A  PIC  X(23)  VALUE  'DATE PRODUCT QUANTITY'
```

```
77  B  PIC  X(5)
```

```
77  C  PIC  X(8)
```

```
77  D  PIC  X(8)
```

```
UNSTRING  A  INTO  B, C, D
```

```
DISPLAY  B           //B= 'DATE '           末尾一个空格
```

```
DISPLAY  C           //C= 'PRODUCT '        末尾一个空格
```

```
DISPLAY  D           //D= 'QUANTITY'
```



- ▶ 可以使用DELIMITED子句设置分解时的定界符。自左向右累计字符，直到遇见定界符，符号左面的内容按MOVE语句的规则传送到接收项

例如：

```
UNSTRING  A  DELIMITED  BY  'T'  INTO  B, C, D
DISPLAY  B                               //B= 'DA___'      末尾三个空格
DISPLAY  C                               //C= 'E_PRODU'     末尾一个空格
DISPLAY  D                               //D= '_QUAN___'    末尾三个空格
```

- ▶ 如果扫描到最右端仍未发现定界符，则以最右端字符为截止界线

例如：

```
UNSTRING  A  DELIMITED  BY  'W'  INTO  B, C, D
DISPLAY  B                               //B= 'DATE_'
DISPLAY  C                               //C全是空格
DISPLAY  D                               //D全是空格
```



- ▶ 用DELIMITED BY ALL 可以表示定界符是长度不固定的一个常量  
例如：

```
77  A  PIC  X(23)  VALUE  'DATE      PRODUCT QUANTITY'  
77  B  PIC  X(5)  
77  C  PIC  X(6)  
77  D  PIC  X(8)
```

```
UNSTRING  A  DELIMITED BY ALL SPACE  INTO  B, C, D  
DISPLAY  B                      //B= 'DATE_ '          末尾一个空格  
DISPLAY  C                      //C= 'PRODUC'  
DISPLAY  D                      //D= 'QUANTITY'
```

\*连续多个空格作为一个定界符



- ▶ 可以使用多个定界符，用“OR”连接  
例如：使用一个或多个连续的空格，句点或逗号作为定界符

```
UNSTRING A DELIMITED BY  
ALL SPACE OR ' ' OR ',' INTO B, C, D
```

- ▶ 可以使用COUNT子句统计已经传送的字符个数  
例如：

```
UNSTRING A DELIMITED BY 'T' INTO B COUNT IN W
```

- \* COUNT子句必须和DELIMITED子句连用
- \* 计数数据项必须是整型，且不能带编辑字符或P字符



- ▶ 如果有多个定界符，可以用DELIMITER子句促使接收项使用的定界符  
例如：

```
UNSTRING A DELIMITED BY ALL SPACE OR 'T' OR  
'W'  
          INTO B DELIMITER IN X  
              C DELIMITER IN Y  
              D DELIMITER IN Z
```

\* 如果DELIMITER和COUNT同时存在，DELIMITER应写在前面

- ▶ 使用WITH POINTER子句从发送项某一字符位开始传送  
例如：从发送项第五个字符开始传送

```
MOVE 5 TO P  
UNSTRING A INTO B, C, D WITH POINTER P
```



- ▶ 使用TALLYING子句统计实际接收传送的接收项项数

例如：

```
77  A  PIC  X(23)  VALUE  'DATE PRODUCT QUANTITY'  
77  B  PIC  X(5)  
77  C  PIC  X(6)  
77  D  PIC  X(8)  
77  N  PIC  9      VALUE  0  
77  P  PIC  99     VALUE  11
```

```
UNSTRING  A  INTO  B, C, D  
          WITH  POINTER  P  TALLYING  IN  N
```

//B= 'CT QU'      C= 'ANTITY'      D为空      N=2

- ▶ 如果全部的接收项都不足以接收发送项的内容时，发生溢出，可以使用ON OVERFLOW子句定义溢出操作，用法同STRING



- ▶ INSPECT语句可以检查一个字符串重的字符：
  - a. 累计一个指定字符出现的次数
  - b. 用指定的字符去代替另一个指定的字符
  - c. 通过指定某些字符来限制上述检查的区间

例1: INSPECT A TALLYING N FOR ALL SPACE  
检查数据项A中是否有空格，找到一个就将计数器N加1

例2: INSPECT A REPLACING ALL SPACE BY ‘,’  
将数据项A中所有空格替换成逗号

例3: INSPECT A REPLACING ALL ZERO BY SPACE  
BEFORE ‘.’  
将数据项A中句点之前的所有‘0’替换成空格



▶ TALLYING子句用来统计满足条件的字符的个数  
格式:

TALLYING 计数器 FOR { ALL | CHARACTERS | LEADING }

例1: 统计字符数

```
INSPECT A TALLYING N FOR CHARACTERS
```

例2: 统计句点之前连续全是的‘0’的个数

```
INSPECT A TALLYING N FOR LEADING ‘0’ BEFORE ‘.’
```

```
// A=123.45          N=0
```

```
// A=12300.45        N=2
```

```
// A=00012300.45     N=3
```





练习:

如果 A= 'AT\*\*F, \*\*\*, T' , 执行下面三个语句, 求N

- a. INSPECT A TALLYING N FOR ALL '\*' AFTER 'T'
- b. INSPECT A TALLYING N FOR LEADING '\*' AFTER 'T'
- c. INSPECT A TALLYING N FOR CHARACTERS  
AFTER 'T' BEFOR '\*'
- d. INSPECT A TALLYING N FOR ALL '\*' , ALL ',' , ALL 'T'
- e. INSPECT A TALLYING  
N1 FOR ALL '\*'  
N2 FOR ALL 'T' BEFORE 'F'

// 5 2 0 9 5 1



## § 5.7 检测语句—INSPECT



- ▶ REPLACING子句用指定的字符代替另一些字符

例1:

A= 'A00.1200B'

INSPECT A REPLACING ALL '0' BY '\$'

A= 'A\$\$\$.12\$\$B'

- ▶ REPLACING子句中可以使用FIRST短语

例2:

A= 'A00.1200B'

INSPECT A REPLACING FIRST '0' BY '\$' AFTER '.'

A= 'A00.12\$0B'



## § 5.7 检测语句—INSPECT



▶ TALLYING和REPLACING子句可以同时使用，TALLYING子句必须在前面

例1:

```
A= 'AT**F, ***, T'
INSPECT A TALLYING N FOR ALL '*'
      REPLACING ALL 'T' BY '$'
// N=5  A= 'A$**F, ***, $'
```

例2:

```
INSPECT A TALLYING N FOR ALL '*'
      REPLACING ALL 'T' BY '*'
// N=5  A= 'A***F, ***, *'
```



- ▶ 使用TRANSFOR语句将数据项中的内容作字符转换
- ▶ 格式:

TRANSFORM 数据项名 CHARACTERS  
FORM 被转换字符 TO 转换字符

例如: T= 'TOTAL AMOUNT'

TRANSFORM T FORM SPACE TO '—' //TOTAL—AMOUNT

TRANSFORM T FORM 'AO' TO '12' //T2T1L—1M2UNT

TRANSFORM T FORM 'TNU' TO '\*' // \*O\*AL AMO\*\*\*

TRANSFORM T FORM A TO B //A= 'TNU' B= '\*'

- ▶ 转换字符的个数不能大于被转换字符的个数, 被转换字符中不能有重复的字符

TRANSFORM T FORM 'AO' TO '123' //错误

TRANSFORM T FORM 'AOA' TO '123' //错误





- § 6.1 执行语句的作用
- § 6.2 执行语句的基本形式
- § 6.3 执行语句的使用规则
- § 6.4 用执行语句实现循环
- § 6.5 执行语句的复杂形式
- § 6.6 执行语句的多重循环形式
- § 6.7 执行语句小结
- § 6.8 EXIT语句
- § 6.8 练习



## § 6.1 执行语句的作用

COBOL程序中若有一部分语句需要多次执行，可以使用PERFORM简化

例如：计算存款的本利和，公式是  $P2=P*(1+R)$ , P为存款, R为利率

1) P=10000, R=0.03                      2) P=50000, R=0.04

```
      . . . . .  
A1.      MOVE  0.03  TO  R.  
          MOVE 10000 TO  P.  
          PERFORM C.  
  
A2.      MOVE  0.04  TO  R.  
          MOVE 50000 TO  P.  
          PERFORM C.  
  
      . . . . .  
C.        ADD  1  TO  R.  
          MULTIPLY P BY R GIVING P2.
```



程序的执行过程：

- ▶ 顺序执行A1段各语句，执行到PERFORM语句时，转到段名是C的段去执行
- ▶ 从C段的第一个语句开始执行，直到该段的最后一个语句为止，然后返回到调用它的PERFORM语句的下一个语句，本例为A2段
- ▶ 接着执行A2段各语句，遇到PERFORM时再转到C段，执行完C段后返回到PERFORM下一个语句





### ▶ 基本格式:

PERFORM 过程名

过程名即过程部下的节名或段名

▶ PERFORM只能转到指定的节或段的开头，执行完其中全部语句后返回；不能直接转去某段的某一个语句，也不能不执行完就返回

▶ 如果需要执行的是若个个段(节)，应指明从哪个段(节)开始，哪个段(段)结束，PERFORM的扩充格式：

PERFORM 过程名1 { THROUGH | THRU } 过程名2





▶ 在一个PERFORM语句所调用的语句序列中，又包括另一个执行语句，即**嵌套**，例如：

```
A.  DISPLAY    'A' .  
    MOVE    'B'    TO  T.  
    PERFORM  B.  
    DISPLAY  T.  
    STOP    RUN.  
B.  MOVE    'C'    TO  T.  
    PERFORM  C.  
C.  DISPLAY  T.           //ACC
```



## § 6.3 执行语句的使用规则

▶ 两个例子：

A.    DISPLAY    'A' .  
      MOVE    'B'    TO   T.  
      PERFORM  B.  
      DISPLAY  T.  
B.    MOVE    'C'    TO   T.  
      PERFORM    'C' .  
C.    DISPLAY  T.  
D.    STOP    RUN.

//ACCCC

A.    DISPLAY    'A' .  
      MOVE    'B'    TO   T.  
      PERFORM  B.  
      DISPLAY  T.  
B.    MOVE    'C'    TO   T.  
              PERFORM    'C' .  
C.    DISPLAY  T.  
      STOP    RUN.

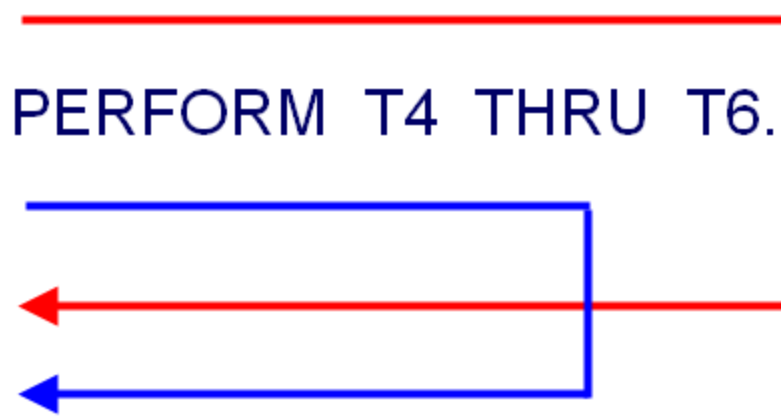
//AC



## § 6.3 执行语句的使用规则

- ▶ PERFORM语句不能交叉嵌套，后一个被调用的执行语句序列应全部在前一个调用的语句之中或之外

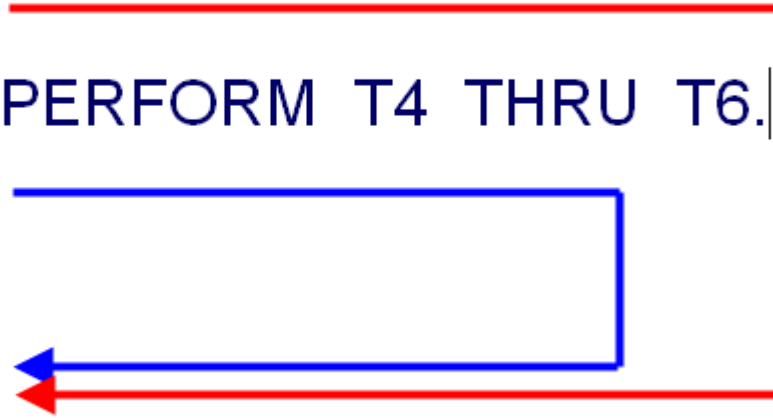
T1.	PERFORM T2 THRU T5.
T2.	_____
T3.	PERFORM T4 THRU T6.
T4.	_____
T5.	←_____
T6.	←_____
T7.	



## § 6.3 执行语句的使用规则

- ▶ 这个例子也非法，不能有重叠的段名

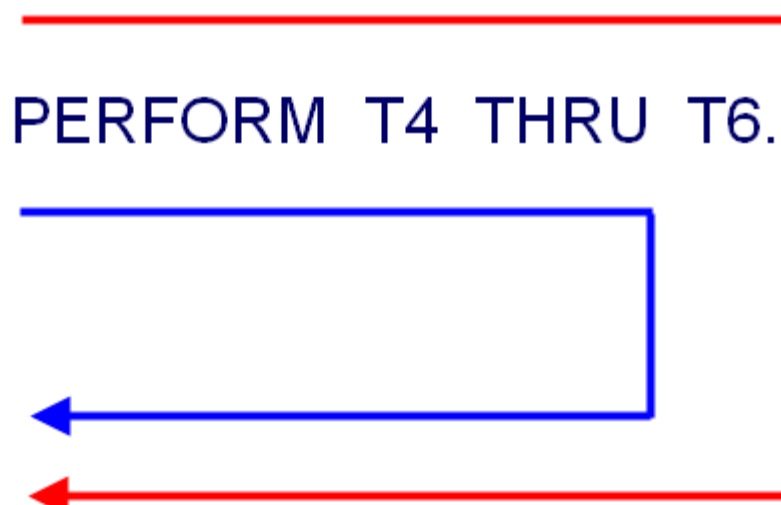
```
T1.      PERFORM T2 THRU T6.  
T2.        
T3.      PERFORM T4 THRU T6.  
T4.        
T5.        
T6.        
Y7.      
```



## § 6.3 执行语句的使用规则

▶ 这个流程是合法的

T1.	PERFORM T2 THRU T7.
T2.	
T3.	PERFORM T4 THRU T6.
T4.	
T5.	
T6.	
T7.	



## § 6.3 执行语句的使用规则

▶ 这个流程也是合法的

T1.       PERFORM T2 THRU T4.

T2.

T3.

PERFORM T5 THRU T7.

T4.

T5.

T6.

T7.






## § 6.3 执行语句的使用规则

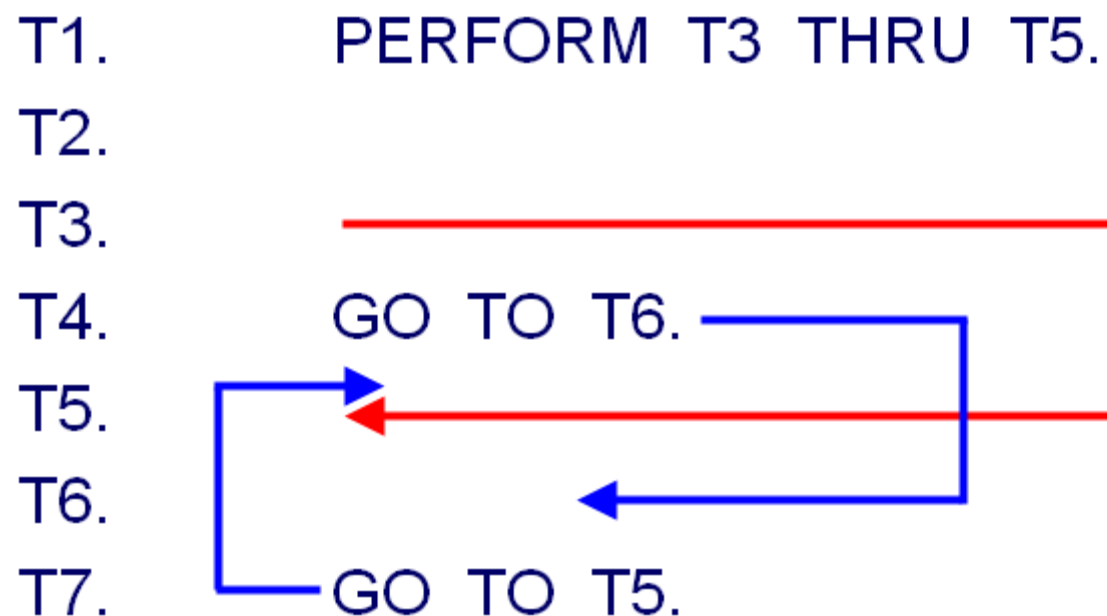
▶ 练习:

T1.	PERFORM T3 THRU T5.
T2.	PERFORM T4 THRU T6.
T3.	
T4.	
T5.	
T6.	



## § 6.3 执行语句的使用规则

▶ 可以在PERFORM语句中包含GO TO语句，使流程转到语句序列之外，但之后应转回到PERFORM语句序列，以便能结束PERFORM调用，让流程继续



- ▶ 可以用PERFORM语句多次执行同一个语句序列，即循环  
(不是在程序的不同地方分别利用几个PERFORM执行同一个语句序列)

格式： PERFORM 过程名1 [ THRU 过程名2 ] N TIMES

例如：

A. PERFORM B 5 TIMES.

.....

B. READ IN-FILE AT END STOP RUN.  
DISPLAY IN-RECORD.

- ▶ 练习6.4.1：求五年的本利合 $P2=P*(1+R)$



## § 6.4 用执行语句实现循环

▶ **N**值如果为零或负数，则语句无效

▶ 可以使用数据项保存**N**值

```
MOVE 5 TO N  
PERFOR A N TIMES
```

▶ 如果**N**值在PERFORM语句序列中有变化，不会影响执行的次数

```
A. ADD 3 TO N  
    DISPLAY N
```

练习6.4.2：计算N的阶乘，N=5



## § 6.5 执行语句的复杂形式

- ▶ 可以反复执行指定的语句序列，直到给定的条件满足为止  
格式：

PERFORM 过程名1 [ THRU 过程名2 ] UNTIL 条件

- ▶ 执行时首先判断条件，若为真则不运行语句序列，若为假则运行

- ▶ 例如：计算 $1+2+\dots+10$

```
A.      MOVE  0  TO  T.
          MOVE  1  TO  N.
          PERFORM B UNTIL  N>10.
          DISPLAY T.
          STOP  RUN.

B.      ADD  N  TO  T.
          ADD  1  TO  N.
```



## § 6.5 执行语句的复杂形式

▶ 可以反复执行指定的语句序列，直到给定的条件满足为止  
格式：

```
PERFORM 过程名1 THRU 过程名2  
VARYING 标识符1 FROM 标识符2 BY 标识符3 UNTIL 条件
```

例如：

```
PERFORM T1 THRU T2 VARYING X FROM A BY B  
UNTIL X>10
```

- 1) 把初值A赋给X
- 2) 检查条件X>10？若不满足，则执行T1~T2段一次
- 3) 将X增加步长B，即 $X+B \rightarrow X$
- 4) 再检查条件X>10
- 5) 若条件满足，则PERFORM执行完毕，继续执行下一语句



## § 6.5 执行语句的复杂形式



▶ 注意:

a. 循环变量X的初值A可以为正，负或零；  
步长B可以为正或负，但**不能为零**！

b. 循环变量X的值在每次循环中自动按步长增加，不必人为添加  
ADD B TO X 语句

c. UNTIL语句的条件，不一定直接用到循环变量，如：

```
A.      MOVE 1 TO N.  
        PERFORM B VARYING X FROM 1 TO 2  
        UNTIL N>10.
```

```
B.      ADD 1 TO N.          ( COMPUTE N = X * 2 )
```



## § 6.6 执行语句的多重循环形式

▶ 格式:

```
PERFORM 过程名1 THRU 过程名2
```

```
    VARYING 参数1 FROM 初值1 BY 步长1 UNTIL 条件1
```

```
    AFTER      参数2 FROM 初值2 BY 步长2 UNTIL 条件2
```

```
    AFTER      参数3 FROM 初值3 BY 步长3 UNTIL 条件3
```

例如:

```
PERFORM T VARYING I FROM 1 BY 1 UNTIL I>3
```

```
    AFTER J FROM 1 BY 1 UNTIL J>5
```

[练习6.6.1](#): 打印九九乘法表





### 五种形式的PERFORM语句

- ▶ `PERFORM T1`
- ▶ `PERFORM T1 THRU T2`
- ▶ `PERFORM T1 THRU T2 5 TIMES`
- ▶ `PERFORM T1 THRU T2 UNTIL N>10`
- ▶ `PERFORM T1 THRU T2  
VARYING X FROM 1 BY 2 UNTIL X>10`
- ▶ `PERFORM T1 THRU T2  
VARYING X FROM 1 BY 1 UNTIL X>10  
AFTER Y FROM 2 BY 2 UNTIL Y>8`



- EXIT语句提供一组过程的公共出口，或者说它指出了被调用过程的逻辑终点，一般用作PERFORM语句序列的出口

例1:

1)	PERFORM A THRU B.	2)	PERFORM A.
	.....		.....
A.	MOVE 1 TO X.	A.	MOVE 1 TO X.
	ADD X TO Y.		ADD X TO Y.
B.	EXIT.		

此例中EXIT可有可无

例2:

```
PERFORM  A  THRU  B.
```

```
.....
```

```
A.      IF  X>Y  GO  TO  B.
```

```
        MOVE  X  TO  T.
```

```
        MOVE  Y  TO  X.
```

```
        MOVE  T  TO  Y.
```

```
B.      EXIT.
```

在此例中，如果没有EXIT，则GO TO 语句无处可转

注意：EXIT语句必须是段中**唯一**的语句，前面必须有段名



学生成绩数据如下：

班级 姓名 语文 数学 英语

假设学校某年级有三个班，编写程序分班保存学生成绩单

除原始数据外，求出每班学生各课平均分，不及格人数，通过比率













Values = (*People+Knowledge*) sharing





Values = (*People+Knowledge*)<sup>sharing</sup>

---

