



# 大型计算机应用技术培训

## COBOL高级编程



# 课程介绍

## 目的：

本教材的目的是通过对教师的讲解和学员的具体操作，通过习题的练习，使学员能够达到能够对 COBOL 语言有初步的了解，依靠技术文档的帮助，独立地进行 COBOL 程序的开发工作，并能够对立地借助相关的技术文档不断提高自己的能力。

## 主要内容：

了解 COBOL 语言的由来和发展

COBOL 的数据

COBOL 的四部

COBOL 的表

COBOL 的排序和合并

## 预修课程：

IBM 大型计算机基本操作



大型计算机应用技术培训.....	1
COBOL高级编程.....	1
第一章 COBOL简介.....	5
1 什么是COBOL.....	6
2 COBOL的历史.....	6
3 COBOL的特点.....	6
3.1 简单的英语单词.....	7
3.2 易用性优先于功能.....	7
3.3 数据与过程分离.....	7
4 COBOL工作环境.....	7
第二章 基本规则.....	8
1 COBOL程序结构.....	9
1.1 部 DIVISION.....	10
1.2 节 SECTION.....	13
1.3 段 PARAGRAPH.....	13
1.4 语句 Sentence.....	13
1.5 语句 Statement.....	13
2 COBOL格式规则.....	14
2.1 格式规则.....	14
2.2 其他规则.....	15
第三章 标识部和环境部.....	18
1 本书语法规则.....	19
2 定义标识部.....	19
3 定义环境部.....	19
3.1 配置节.....	20
3.2 输入-输出节.....	20
第四章 数据部.....	22
1 数据部的格式.....	23
2 数据类型.....	24
2.1 数据项.....	24
2.2 数据组.....	24
3 如何在COBOL中定义数据.....	24
3.1 层次结构.....	26
3.2 数据名.....	27
3.3 COBOL数据定义.....	28
3.4 PICTURE从句.....	28
3.5 PICTURE 子句.....	31
3.6 缩写.....	32
3.7 数据类型.....	32
3.8 如何定义数据.....	33
3.9 66 层定义.....	42
3.10 88 层定义.....	43
第五章 Sequential File.....	44
1 一些基本概念.....	45



1.1	Files, Records, Fields.....	45
1.2	文件的类型.....	45
1.3	文件的访问方式.....	46
2	如何处理文件.....	46
2.1	记录缓存区.....	46
2.2	缓存区的使用.....	47
2.3	使用文件.....	47
第六章	过程部结构及一些基本语法.....	59
1	命令的基本分类.....	60
2	数据传输命令.....	60
2.1	MOVE .....	60
2.2	MOVE CORRESPONDING .....	61
3	人机通讯命令.....	62
3.1	ACCEPT .....	62
3.2	DISPLAY .....	64
3.3	INITIALIZE .....	64
4	字符串操作命令.....	65
4.1	INSPECT .....	65
4.2	STRING .....	71
4.3	UNSTRING.....	83
5	过程分支命令.....	90
5.1	GO TO.....	90
5.2	PERFORM 语句.....	91
5.3	CONTINUE .....	100
5.4	EXIT .....	100
5.5	STOP .....	101
第七章	算术语法.....	102
1.	ADD.....	103
1.1.	ADD TO.....	103
1.2.	ADD GIVING.....	104
1.3.	ADD CORRESPONDING.....	105
2.	SUBTRACT.....	105
2.1.	SUBTRACT FROM.....	105
2.2.	SUBTRACT GIVING.....	106
2.3.	SUBTRACT CORRESPONDING.....	107
3.	MULTIPLY .....	107
3.1.	MULTIPLY BY .....	108
3.2.	MULTIPLY GIVING .....	108
4.	DIVIDE.....	109
4.1.	DIVIDE INTO .....	110
4.2.	DIVIDE INTO GIVING .....	111
4.3.	DIVIDE BY GIVING .....	111
4.4.	DIVIDE INTO ...REMAINDER.....	112
4.5.	DIVIDE BY ...REMAINDER.....	113



5.	COMPUTE.....	114
6.	ROUNDED .....	114
7.	SIZE ERROR.....	115
8.	CORRESPONDING .....	116
第八章	逻辑与控制语法.....	117
1	IF THEN .....	118
1.1	关系型条件.....	118
1.2	类条件.....	119
1.3	符号条件.....	120
1.4	复合Condition.....	121
1.6	条件名.....	122
1.7	数据定义 88 层的定义.....	122
1.8	EVALUATE .....	124
第九章	排序与合并.....	127
1	基本定义.....	128
2	对环境部，数据部的要求.....	128
3	排序.....	128
4	合并.....	130
第十章	表格处理.....	132
1	基本定义.....	133
2	定义表格.....	133
2.1	索引.....	134
3	初始化表格.....	135
4	表格的引用.....	136
5	处理表格的 2 种语法.....	136



# 第一章 COBOL简介

- 什么是 COBOL
- COBOL 的历史
- COBOL 的特点
- COBOL 工作环境



# 1 什么是COBOL

COBOL 是 Common Business Oriented Language（通用商业的语言）的缩写。主要是专门为数据处理而设计的计算机高级程序设计语言。

现今，这种语言主要应用于政府、银行、运输系统中。尤其是针对文件系统地编程中。

COBOL 不是用来编写操作系统的语言。

在这些领域中，有大量的数据，数据处理，就是对这些数据进行记录、储存、传递、分类、计算、比较、排序、汇总等等系列活动的总和，其目的是从大量杂乱无章的数据中，抽取并导出对特定用户有价值的信息，作为决策的依据。它的特点是，在一般情况下，待处理的以及处理后的数据量都是非常大的。

## 2 COBOL的历史

1959 年 5 月，美国国防部(计算机的最大用户)出面召集了一个会议，**Conference on Data Systems Languages**，专门讨论创建一种用于数据处理的语言的问题。会议的结果是成立了一个九人的 CODASYL 委员会，由它负责起草语言文本，经过紧张的工作，委员会于 1959 年 12 月完成了 COBOL 语言初稿，并于 1960 年 4 月正式发表。这第一个 COBOL 文本称为“COBOL-60”。

COBOL 语言自诞生后，不断的改进完善。此后，新的 COBOL 标准由 American National Standards Institute (ANSI).美国国家标准组织确定。

此间，有 3 个 COBOL 标准分别于 1968，1974，1985 发布。

## 3 COBOL的特点

商业经济方面的事务处理是计算机的一个重要的应用领域，COBOL 语言受到广泛重视，据资料统计，世界上约一半的程序是用 COBOL 语言编写的，而在银行领域，约有 85% 的程序是用 COBOL 编写的。

最初设计 COBOL 语言的目的是让不懂计算机程序的用户等都可以读得懂 COBOL 编写的程序，这使得 COBOL 具有了以下的特点

- 简单的英语单词
- 易用性优先于功能



➤ 数据与过程分离

### 3.1 简单的英语单词

COBOL 非常接近于英语口语，一个 COBOL 程序就像一篇文章一样，章、节、段落分明，层次清楚，语句结构同人们日常说话相似，几乎没有特殊的、专用的表达方式，对于熟悉英语的人来说，学习、理解、掌握 COBOL 语言比较容易，对于用户之间的技术交流，程序移植也非常的方便。COBOL 语言所使用的英语都是非常简单的，即便不懂英语的人也很容易的学习，使用。

### 3.2 易用性优先于功能

COBOL 是面向商业经济应用的，最适合大量数据的处理，易于生成各种单据、报表。而在一般的数据处理中，复杂计算是极少，甚至没有的，因此，COBOL 语言的计算能力不强，比如，COBOL 没有三角、指数、对数等标准函数，也没有浮点计算。这是基于 COBOL 的商业应用考虑的，不是设计者的疏忽、不周到，也不是因为技术上的困难。此点既是优点也是缺点。易用性使得 COBOL 便于学习和使用，但也使得 COBOL 的功能比较薄弱。但是，COBOL 主要使用在商业系统中，这种系统中，复杂的是商业的逻辑，而不是功能。且其易用性使得 COBOL 语言编写的程序更易维护。

### 3.3 数据与过程分离

COBOL 对于数据的描述和对于过程的描述是并重的。用其他语言编写程序时，程序员的注意力集中于设计、描写一个解决问题的过程，对于解决问题过程中所用到的数据或者获得的结果，只在过程中顺便的、通常也是很简单的描写一下。

COBOL 语言认为，程序设计中过程描写固然重要，对过程中所要处理的对象的描写同样重要，把它提出来作为一个独立的部分，并且，必须在过程描述之前，完整地描述过程中所要处理的一切数据。

这种安排，有利于系统资源的合理利用和程序的调试、修改。在数据发生变化而处理过程不变时，只需要修改程序的数据部分，从而大大降低维护开销。

COBOL 不是一个十全十美的语言，但是在特定的应用领域，主要是商业领域，银行领域，它是一个性能较完善、功能较强的语言。

## 4 COBOL工作环境

只要有 IBM 机器的地方，就有 COBOL。包括 AS400，RS6000，ES9000。在银行界，主要机型是 ES9000，和 RS6000。





## 第二章 基本规则

- COBOL 程序结构
- COBOL 格式规则
- 其他



## 1 COBOL程序结构

COBOL 语言具有严谨的结构，每个 COBOL 程序，都有以下四个部分组成：

标识部（IDENTIFICATION DIVISION）

环境部（ENVIRONMENT DIVISION）

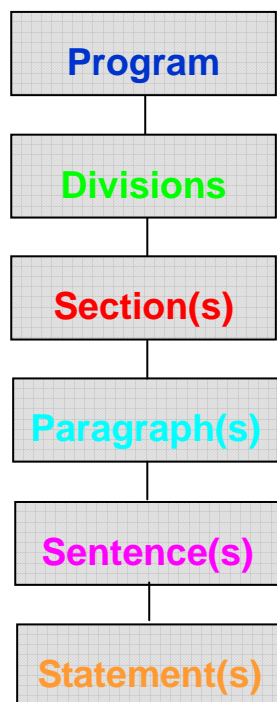
数据部（DATA DIVISION）

过程部（PROCEDURE DIVISION）

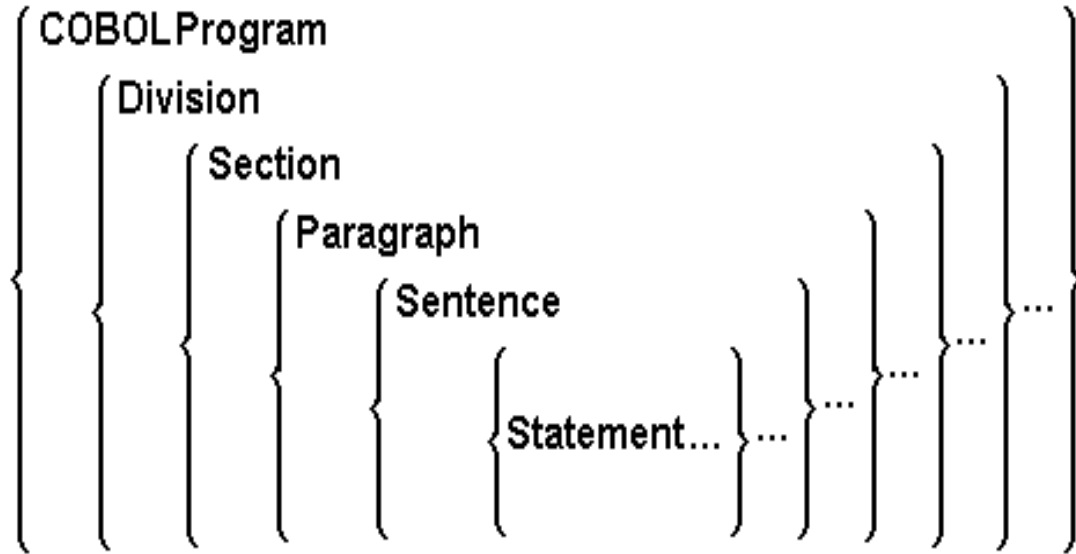
这四个部必须以上述次序出现，且每个部都必须有。根据情况，环境部和数据部的内容可能为空，但是部的标题（division header）仍然必须写出来。

COBOL 语言的四个部，在部下可以分节（section），节中可以分段（paragraph），段中有若干句（sentence），句中有语句（statement）。形成一个树形结构。

这些层次关系从上到下，如下图所示。



从外到内，如下图所示



## 1.1 部 DIVISION

每个 COBOL 程序都由四个部组成，也许部的内容为空，但是部的标题必须列出。

- 标识部 IDENTIFICATION DIVISION
- 环境部 ENVIRONMENT DIVISION
- 数据部 DATA DIVISION
- 程序部 PROCEDURE DIVISION

### 1.1.1 标识部 IDENTIFICATION DIVISION

提供有关程序的相关信息。提供程序的文档说明，如程序名和程序员名等。主要用于给出程序名。

PROGRAM-ID 段包含程序名称，是每个程序必须的，且必须跟在 IDENTIFICATION DIVISION 后，程序名称最长 30 字符。编译器通过 PROGRAM-ID 来识别不同的程序。

AUTHOR 段通常包含程序员名，DATE-WRITTEN 通常是程序编写的日期。这些都是可选项。

对编译器而言，标识部的信息，通常是说明性质的。

通常格式如下例

```
IDENTIFICATION DIVISION.  
PROGRAM-ID FIRSTP.  
AUTHOR LILLY.
```



DATE-WRITTEN 2004/12/15.

### 1.1.2 环境部 ENVIRONMENT DIVISION

描述程序运行的环境的相关信息。包括硬设备和软设备的环境条件。

提供与程序运行环境相关的信息，列出程序使用的文件。

主要有两个节：配置节（CONFIGURATION SECTION）和输入-输出节（INPUT-OUTPUT SECTION）

例如

```
ENVIRONMENT      DIVISION.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
          SELECT CZFCNBZ ASSIGN TO      DA-CZFCNBZ
              ORGANIZATION IS      INDEXED
              ACCESS      MODE      SEQUENTIAL
              RECORD KEY      IS      NBZHKEY
              FILE STATUS      IS      IN-FILE-STATU.
          SELECT NOSUB1 ASSIGN TO      DA-NOSUB1
              ORGANIZATION IS      INDEXED
              ACCESS      MODE      DYNAMIC
              RECORD KEY      IS      OLD-JG-ACCT
              FILE STATUS      IS      SUBJ-FILE-STATU.
          SELECT GLACA ASSIGN TO      SYS018-S-GLACA
              ORGANIZATION IS      SEQUENTIAL
              ACCESS      MODE      SEQUENTIAL
              FILE STATUS      IS      OUT-FILE-STATU.
```

### 1.1.3 数据部 DATA DIVISION

描述程序运行用到的所有数据对象，数据变量，包括输入数据，输出数据，中间数据以及控制变量等。程序中使用到的变量必须在这里定义。其中，最重要的是 File Section 和 Working-Storage Section。

**FILE SECTION** 描述程序处理数据的输入和输出。

**WORKING-STORAGE SECTION** 用来描述程序使用到数据变量。

使用如下例

```
DATA      DIVISION.
      FILE SECTION.
      FD CZFFKZX
```



LABEL RECORDS ARE STANDARD  
DATA RECORD IS CZFFKZX-RECORD.  
COPY CZFCKZX.

**WORKING-STORAGE SECTION.**

01 STARTMSG                    PIC X(8)    VALUE ' @@@@ @@@@' .  
01 FILLER                      PIC X(8)    VALUE ' INPUTMSG' .  
01 WK-DATE-99                PIC 9(9) .  
01 WK-DATE-99-RED REDEFINES WK-DATE-99.  
    05 WK-DATE-PRE          PIC 9(3) .  
    05 WK-DATE-96          PIC 9(6) .

**1.1.4 程序部 PROCEDURE DIVISION**

描述程序处理过程。主要用于处理数据。

由 Sections, Paragraphs, Sentences , Statements 组成。

Section 是可选的。每个 PROCEDURE DIVISION 至少包含一个 paragraph, sentence and statement 。

程序员自己定义 paragraph 和 section 的名称。此名称最好体现 paragraph 和 section 的功能。

各个部与部中的节和段的关系如下表

部 (DIVISION)	节 (SECTION)	段 (PAPAGRAPH)
标识部 (IDENTIFICATION DIVISION)	-	程序标识符段 (PROGRAM-ID)
环境部 (ENVIRONMENT DIVISION)	配置节 (CONFIGURATION SECTION)	源计算机段 (SOURCE-COMPUTER)
		目标计算机段 (OBJECT-COMPUTER)
		特殊名段 (SPECIAL-NAMES)
	输入-输出节 (INPUT-OUTPUT SECTION)	文件控制段 (FILE-CONTROL)
		输入-输出控制段 (I-O-CONTROL)
数据部 (DATA DIVISION)	文件节 (FILE SECTION)	文件描述体 ( File Description Entries)
	工作存储节 (WORKING-STORAGE SECTION)	记录描述体 ( Record Description Entries)



	连接节 (LINKAGE SECTION)	数据项描述体 (Data Item Description Entries)
	报表节 (REPORT SECTION)	
	通讯节 (COMMUNICATION SECTION)	
程序部 (PROCEDURE DIVISION)	根据需要安排	根据需要安排

## 1.2 节 SECTION

Section 由 1 个或多个 Paragraph 组成。

每个 Section 以 Section Name 开始，直到碰到下一个 Section Name，或者整个程序的结束，标识一个 Section 结束。

## 1.3 段 PARAGRAPH

1 个或多个 Paragraph 组成一个 Section。

Paragraph 由 1 个或多个 Sentence 组成。

每个 Paragraph 以 Paragraph Name 开始，知道碰到下一个 Paragraph Name，或者 Section Name，或者整个程序的结束，标识一个 Paragraph 结束。

## 1.4 语句 Sentence

1 个或多个 Sentence 组成一个 Paragraph。

Sentence 由 1 个或多个 Statements 组成，结束于英文的“.”符号。

## 1.5 语句 Statement

Statements 由 COBOL 的“动词”，“直接数”和“操作符”组成。

例 1:

```
READ GLACA2
  AT END
    MOVE 'T' TO ACATOGL-EOF-STATE
    DISPLAY 'TOTOL: ' HANDLED-LINES
    GO TO 1000-EXIT
  END-READ.
```



例 2:

ADD 1 TO HANDLED-LINES.

## 2 COBOL格式规则

### 2.1 格式规则

由于 COBOL 是一种古老的计算机语言，当时的计算机的输入设备还使用打孔卡输入。因此，COBOL 程序格式比较原始，要求比较严格

每行开头是 6 个字符的序号区

序号区后是一个字符的指示符区

指示符区后是 4 个字符的 A 区

A 区后是语句区，称作 B 区，通常到第 72 个或第 80 个字符。

#### 2.1.1 序号区 1-6列

用于放置序号

没有要求

#### 2.1.2 指示符区 7列

正常的程序源代码中，指示符区为空。

星号“\*”或者斜杠“/”，表示其为说明行，行中内容将被忽略。星号与斜杠的区别在于，斜杠在编译时强制程序清单另起一页

连接符“\_”表示该行为上一行的续行，B 区的第一个非空字符是上一行最后一个非空字符后面的字符。

任何 COBOL 的语句都可以跨行，此时不需要“\_”连接符。只要在任何允许插入空格的地方插入分行符，在下一行的 B 区续行。

只有要在单词中间分行时，需要使用连接符。

字母“D”表示该行为调试行。调试行只有在打开测试时才属于该程序，此字母逐渐已经不再使用，将来的 COBOL 标准也不支持



### 2.1.3A 区 8-11列

用于书写 Division、Section、Paragraph 的标题。

FD、SD、RD、CD 输入。

层号 01 和 77。

第一个字符必须写在 A 区的第 8、9 列或者第 11 列。为了程序的清晰易读，建议一律写在第 8 列。

### 2.1.4B 区 12-80列

程序语句所在的区域，过程部的语句必须从 B 区开始。

02-49 的各层数据也写在此区。可以从任意一例开始书写，同样为了程序的清晰易读，见一律从第 12 列开始书写。

最好每行语句不要超过第 72 位，由于 9000 终端的限制，有的时候，73-80 的字符看不到，需要右移，不方便。早期的系统中，73-80 列的内容不予编译，因此，为避免麻烦，最好也不要在这个范围内写入正文内容。

## 2.2 其他规则

### 2.2.1 COBOL 字符集

COBOL 字符是组成 COBOL 程序的最小单位。字符集是所有 COBOL 字符的总称。COBOL 字符集总共有 78 个字符，分别是：

字母：A-Z，a-z

数字：0-9

运算符号：+、-、\*、/

关系符号：=、>、<

标点符号：,、;、.、'、”、(、)

货币符号\$和空格

### 2.2.2 保留符号

COBOL 有众多的保留字，大约 300 个左右





保留字用大写字母来表示

### 2.2.3 用户定义字

程序中除了保留字外，可以有程序员定义的名称。这些名称不一定按照英文书写，可以使用拼音，或者随意编写。无论是变量名，段落名，节名等，必须遵守以下的规范：

至少为 1 个字符长，最长 30 字符。文件名通常限制为 8 个字符。

至少包含 1 个字母或者连接符，不能以连接符开头或结束。

可以使用的字符包括：字母 A 到 Z，数字 0 到 9，连接符。

段（Paragraph）名应在节（Section）中保持唯一性。

节（Section）名应在程序（Program）中保持唯一性。

段（Paragraph）名不能与节（Section）名相同。

段名可以使用数字开头。数字必须为正整数。

用户的选择，最好能使人在阅读时候能够望文生义，不必另加说明。

例 1：正确的用户字

NAME, Y, FILE-READ

例 2：不正确的用户字

-BEGIN      连接符为第一个字符

DATA          保留字

DAT NUMBER    中间有空格

“GOOD”      有使用范围外的字符

### 2.2.4 书写程序的规则

- 一个字符占据一格；
- 除了非数值的直接量外，正文一律用英文大写字母；
- 两个 COBOL 字之间至少留一个空格；



- 严格遵守分区的规定，不得越区书写；
- COBOL 允许一行中写几个语句，也允许一个语句写在几行中，但最好一句一行，避免续行。
- 部、节标题必须独占一行；
- 表达式中的运算符以及关系符前后都要留空，“(”后不许要留空，但是“)”后必须留空；
- 使用“,”、“.”、“;”时，其后必须至少跟一个空格，而前面不能有空格。



## 第三章 标识部和环境部

- 本书的语法规则
- 如何定义标识部
- 如何定义环境部



## 1 本书语法规则

Cobol 作为一门让人易读的语言，有很多的无用的词在语句中。因此语法中，有一些词是可选的，有一些是必选的，本书介绍时使用的语法规则如下：

- 用{ }括起的语句，如果有多条语句可选，必须选一句，如果只有一条语句，则必选
- 用[ ]括起的语句，为可选项，可有可无
- ...表示之前的语句重复
- || 表示其间的内容可以选择一个或几个元素，但每个元素只能用一次

## 2 定义标识部

标识部的格式如下：

<u>IDENTIFICATION DIVISION.</u>	标识部
<u>PROGRAM-ID.</u>	程序名
[ <u>AUTHOR.</u> ]	作者名
[ <u>INSTALLATION.</u> ]	设备名
[ <u>DATE-WRITTEN.</u> ]	程序编写日期
[ <u>DATE-COMPILED.</u> ]	程序编译日期
[ <u>SECURITY.</u> ]	程序保密性要求

这个部中只有一个必写的段，就是程序标识符段，在此段中给出程序名。其他的段，都是任选的段。可写，可不写。在书写的情况下，这些段的内容都是按照注解来处理的。段的顺序根据系统的不同，有的可以任意排列，有的系统不允许，为了避免麻烦，最好按照上述顺序填写。

根据通常的经验，最好写上 AUTHOR 段和 DATE-WRITTEN 段。

DATE-COMPILED 段，可以只给标题，不给内容，这时，由编译程序自动填补相应的日期。

在某些系统中 IDENTIFICATION 可以缩写为 ID。因此，ID 也是 COBOL 的保留字。

## 3 定义环境部



环境部又分为配置节（CONFIGURATION SECTION）和输入-输出节（INPUT-OUTPUT SECTION），每节又分为若干段。格式如下：

<u>ENVIRONMENT DIVISION.</u>	标识部
[ <u>CONFIGURATION SECTION.</u>	配置节
[ <u>SOURCE-COMPUTER. ....</u> ]	源计算机段
[ <u>OBJECT-COMPUTER. ....</u> ]	目标计算机段
[ <u>SPECIAL-NAMES. ....</u> ]]	特殊名段
[ <u>INPUT-OUTPUT SECTION.</u> ]	
[ <u>FILE-CONTROL. ....</u> ]	
[ <u>I-O-CONTROL. ....</u> ]	

### 3.1 配置节

源计算机段给出编译与调试本程序的计算机名，其内容被编译程序当作注解。

目标计算机段给出执行本程序的计算机名，其内容也是注释性的。

特殊名段主要定义助记忆名，并说明程序中若干符号的用法。

这三段都是可选的，不重要，如果三个段都不写，则配置节的标题也不写。

### 3.2 输入-输出节

输入-输出节用以描写程序中各输入输出文件的特性，以及它们与外部设备的联系，有关输入输出文件在缓存区方面的规定。以上两部分的内容分别在文件控制段和输入输出控制段。只要程序中用到了文件，就一定要有输入-输出节。

#### 3.2.1 文件控制段

文件控制段主要有以下作用

- 为文件规定文件名，说明该文件驻留于哪个外部设备；
- 规定文件输入输出缓冲区的个数；
- 文件的类型和对文件的存取方式；
- 为反映文件存取操作结果的状态码指定工作单元



具体格式根据不同的文件类型而不同，详见后章。

### **3.2.2 输入输出控制段**

输入输出控制段用以说明程序在输入输出方面的一些技巧。目前比较少使用。



## 第四章 数据部

- 了解数据部的格式
- 了解 COBOL 的数据类型
- 如何定义数据
- 掌握正确的使用层号定义数据
- 学会为数据赋初值



## 1 数据部的格式

COBOL 的特点是，数据描述与程序描述并重，过程中用到的所有数据，都要在数据部现有明确的定义，完整的描述。

数据部的结构如下：

<u>DATA DIVISION.</u>	
<u>FILE SECTION.</u>	文件节
<u>WORKING-STORAGE SECTION.</u>	工作存储节
<u>LINKAGE SECTION.</u>	连接节
<u>COMMUNICATION SECTION.</u>	通讯节
<u>REPORT SECTION.</u>	报表节

数据部有五节

文件节：用以描述文件中的数据

工作存储节：描写文件意外的独立的数据项和记录

连结节：调用自程序使用，描述主程序与被调程序之间的结合参数

通讯节：用于通讯模块

报表节：用于报表打印，描述与报表有关的数据

本章主要介绍工作存储节，其他节结合以后的内容讲述。

我们使用到的数据项分为两部分，一部分属于输入输出文件，这部分的内容存放在文件节，后章，我们结合具体的文件类型讲述；一部分属于非输入输出文件数据，例如，存放中间结果的数据项，用作计数器的数据相等，这些都要在工作存储节中描述。

工作存储节的格式基本如下：

WORKING-STORAGE SECTION.

$$\text{层号} \left\{ \begin{array}{c} \text{数据名} \\ \text{FILLER} \end{array} \right\} \left[ \left\{ \begin{array}{c} \text{PICTURE} \\ \text{PIC} \end{array} \right\} \text{IS 描述字符串} \right] [ \text{VALUE 常量} ]$$





## 2 数据类型

COBOL 语言中的变量的数据类型之间的差别不是非常明显。因为实际上 COBOL 不是定义数据项，而是通过向系统提供数据项的例子来描述（PICTURE）数据项。

我们主要介绍两种数据类型

- Numeric 数字型
- text / string 文本型

不同数据类型决定了使用什么样的操作符。

### 2.1 数据项

COBOL 中的基本数据项-elementary

COBOL 的数据类型成为数据类别-category

- Numeric
- Numeric-editd
- Alphabetic
- Alphanumeric
- Alphanumeric-edited

### 2.2 数据组

数据之间的关系，基本有两种，一种是孤立的，另一种是数据间存在逻辑上的依赖关系。将基本数据项组成层次结构，称为组数据项

组可以分成子组，最终分成基本数据项

## 3 如何在COBOL中定义数据

COBOL 定义数据的基本格式如下：



层号 { Dataname1 } [ REDEFINES Datanam2 ]  
       FILLER  
 [ { PICTURE } IS Data string  
    PIC  
 [ [ USAGE IS ] { COMPUTATIONAL  
                   COMPUTATIONAL-3  
                   DISPLAY  
                   INDEX }  
 [ [ SIGN IS ] { LEADING } SEPARATE CHARACTER  
                   TRAILING  
 [ OCCURS { N1 TO N2 TIMES DEPENDING ON Dataname3 }  
                   N3 TIMES  
 [ { ASCENDING } KEY IS Dataname4 [ Dataname5 ]...]  
    DESCENDING  
 [ INDEXED BY indexname1 [indexname2]...]  
 [ { SYNCHRONIZED } [ LEFT ]  
    SYNC                  RIGHT ]  
 [ { JUSTIFIED } RIGHT ]  
    JUST  
 [ BLANK WHEN ZERO ]  
 [ VALUE IS 直接量 ].

- 必须有层号
- 必须有数据名
- 使用 PICTURE 从句



- 可以通过 VALUE 从句为数据赋初值
- FILLER 的原意为填充物，在某些记录格式中，为了便于阅读或检查，在有用字段间需要留出一些空格，这些空格字段在以后的过程中都不会被单独引用，因此没有必要为其分别予以命名，一律命名为“FILLER”
- 最后以“.”结束

### 3.1 层次结构

在数据描述中，无论哪个层次上的数据，层号和数据名是必须的。数据名是数据的标识符，层号标志着该数据的层次，层号必须为整数。

01 层是最高层，49 层是描述数据的最低层，02-49 用于描述 01 层下的数据项。层号越大，层次越低，在一个记录中，同一层次上的数据用相同的层号。

77 层用于与其它数据项无从属关系的数据项，88 层为数据项目条件名保留，66 层为数据描述符项目保留。

01 和 77 层号必须在 A 区开始书写。其它层号从 B 区开始书写。

重要的是不同层次数据间的关系，而不是具体使用的某个行号。如下例

例 1：雇员个人信息

<b>01 employeedetail.</b>		<b>=</b>	<b>01 employeedetail.</b>	
<b>02 name</b>	<b>PIC X(8).</b>		<b>05 name</b>	<b>PIC X(8).</b>
<b>02 birthday.</b>			<b>05 birthday.</b>	
<b>03 year</b>	<b>PIC 9 (4).</b>		<b>10 year</b>	<b>PIC 9 (4).</b>
<b>03 month</b>	<b>PIC 9(2).</b>		<b>10 month</b>	<b>PIC 9(2).</b>
<b>03 day</b>	<b>PIC 9(2).</b>		<b>10 day</b>	<b>PIC 9(2).</b>
<b>02 sex</b>	<b>PIC A.</b>		<b>05 sex</b>	<b>PIC A.</b>

例 2：定义数据项



```
01 RECORD-X.  
  03 FIELD-1.  
    05 NAME          PIC X(20).  
    05 STREET-1      PIC X.  
  03 FIELD-2.  
    05 STREET-2      PIC X(20).  
    05 CITY          PIC X(20).  
66 STREETS RENAMES STREET-1 THRU STREET-2.
```

### 3.2 数据名

由于在 COBOL 语言中，数据都是按名引用的，因此，每个数据项都要有自己的名称。

数据项的名称称作数据名——data-name

数据名不要求唯一，可以对不同的数据项使用同一数据名

引用不唯一的数据名时，需要引用所属的组名

同一组中的同一层的数据名应该唯一

例 3:

```
01 CLASSMATES.  
  05 NAME          PIC X(8).  
  05 BIRTHDAY      PIC 9 (8)  
  05 SEX          PIC X.  
01 FRIENDS.  
  05 NAME          PIC X(8).  
  05 BIRTHDAY      PIC 9 (8)  
  05 SEX          PIC X.
```

引用时

NAME OF CLASSMATES

NAME OF FRIENDS



### 3.3 COBOL数据定义

COBOL 使用特殊的机制描述程序中使用到的数据项。

COBOL 通过范例定义数据项。

程序员给系统提供数据项的例子或者说是模版，有或者称作描述（ PICTURE ）数据项。

系统通过对数据项的描述（picture）分配数据。

### 3.4 PICTURE从句

定义基本项的具体长度、格式、和数据类型。

数据组不允许使用 PICTURE 从句。

#### 3.4.1 Picture从句经常使用到的符号

- 9 表示数字 0-9。
- X 表示任何字符。
- V 表示数字型变量中小数点的位置。
- S 表示代数符号，通常表示符号位。
- A 表示字母。

#### 3.4.2 格式字符（一）

- , 插入“ , ”。
- B 表示空字符。
- 0 插入“0”。
- / 插入“/”。
- . 插入“.”。

Result A		Result B	
Picture	data	Picture	data



PIC 999999	123456	PIC 999,999	123,456
PIC 9(6)	000023	PIC 9(3),999	000,023
PIC 9(6)	123456	PIC 9(3)BB9(3)	123 456
PIC 9(6)	001234	PIC 9(3)009(3)	00100234
PIC 9(6)	040913	PIC 99/99/99	04/09/13
PIC 999V99	12345	PIC 999.99	123.45
PIC 999V99	01245	PIC 999.9	012.4
PIC 999V99	12345	PIC 99.99	23.45
PIC 999	123	PIC 999.99	123.00

### 3.4.3 格式字符（二）

- + 正值插入“+”，负值插入“-”。
- 正值插入空格，负值插入“-”。
- \$ 插入美元符号。
- CR 正值插入 2 个空格，负值插入“CR”。
- DB 正值插入 2 个空格，负值插入“DB”。
- “+”和“-”，“CR”和“DB”是相互排斥的，这两对编辑字符也是互相排斥的。
- 如果货币号和正负符号同时使用，则正负号在前，货币号在后。
- “+”，“-”，“\$”可以连写两个以上，使输出的符号和币号浮动在最高有效数位之前。

### 3.4.4 格式字符（二）

- \* 表示抑制前面的 0，并换成\*。
- Z 表示抑制前面的 0，并换成空字符。
- 两种抑制字符不能同时出现在一个 PICTURE 子句中。



Result A		Result B	
Picture	data	Picture	data
PIC S999	-123	PIC -999	-123
PIC S999	-123	PIC 999-	123-
PIC S999	123	PIC -9(3)	123
PIC S999	123	PIC +9(3)	+123
PIC S999	-123	PIC +9(3)	-123
PIC S999	-123	PIC 9(3)+	123-
PIC S9(4)	-0003	PIC ++++9	-3
PIC S9(4)	+0080	PIC ++++9	+80
PIC S9(5)	+12345	PIC ----9	2345
PIC 9(4)	1234	PIC \$9(5).9	\$01234.0
PIC 9(4)	0020	PIC \$,\$,\$9.99	\$20.00
PIC 9(4)	0123	PIC \$,\$,\$9.99	\$123.00
PIC 9(5)	12345	PIC \$,\$,\$9	\$2,345
PIC 9(3)	123	PIC 9(3)CR	123
PIC 9(3)	-123	PIC 9(3)CR	123CR
PIC 9(3)	123	PIC 9(3)DB	123
PIC 9(3)	-123	PIC 9(3)DB	123DB
PIC 9(6)	040913	PIC 99/99/99	04/09/13
PIC 999V99	12345	PIC 999.99	123.45
PIC 999V99	01245	PIC 999.9	012.4
PIC 999V99	12345	PIC 99.99	23.45
PIC 999	123	PIC 999.99	123.00
PIC 9(5)	12345	PIC ZZ,999	12,345
PIC 9(5)	01234	PIC ZZ,999	1,234
PIC 9(5)	00123	PIC ZZ,999	123
PIC 9(5)	00012	PIC ZZ,999	012



PIC 9(5)	01234	PIC **, **9	*1,234
PIC 9(5)	00123	PIC **, **9	**123
PIC 9(5)	00000	PIC **, ***	*****

### 3.5 PICTURE 子句

PICTURE 子句对每个数据项进行详细描述，包括类型，大小等。PICTURE 只对基本项进行描述，而不能描述数据组。

书写 PICTURE 时，应先写层号、数据名，然后是 PICTURE 保留字，最后是数据项的描述字符串和句号。

如下例

PICTURE 999                    3 位整型数据，仅表现正数

PICTURE S999                3 位整型数据，包括正负数

PICTURE XXX                3 位字符型

PICTURE 99V99              浮动型，仅指正数，从 0 到 99.99

PICTURE S99V99            浮动型，包括正负数，从-99.99 到 99.99

可以使用缩写 PIC。

数字型变量最长 18 位。

从这里例子可以看出，cobol 对变量的定义就是对变量的形态的描述。

COBOL 语言中，对数据项的字形描述确定了该数据项的类型和长度。在描述数据时，必须仔细斟酌，根据使用的需要和数据的可能变化范围写出恰当的描述。尤其是对数值型数据的描述，要仔细考虑以下几点：

- 数是否有正有负，如果是，描述中一定要有 S，否则，所有的数都将被当作正数处理。
- 数值的变化范围，最小几位，最大几位，要根据应用的情况确定。从而决定描述中的 9 的个数，如果 9 的个数多了，浪费空间，各数少了，数据将被掐头去尾，数据失真。
- 数的类型，是整数还是小数，小数点的位置在哪里？





### 3.6 缩写

在 PICTURE 子句中，描述时，可以使用一些缩写方式，例如，使用 PIC 替代 PICTURE，使用 ( ) 替代重复的元素

PIC 9(6) 等同于 PICTURE 999999

PIC 9(6)V99 等同于 PIC 999999V99

PICTURE X(10) 等同于 PIC XXXXXXXXXXXX

PIC S9(4)V9(4) 等同于 PIC S9999V9999

PIC 9(18) is 等同于 PIC 999999999999999999

### 3.7 数据类型

#### 3.7.1 字母型数据

其表达的类型仅限于字母表中的字符和空格符。

定义数据项时，数据项中的每一个字符用一个 A 来表示。

数据项的长度是构成该数据项的字母 A 的个数。

- 05 SEX PICTURE A.
- 05 PHONE PICTURE A(10).

#### 3.7.2 字母数字型数据

表达的数据可以包含任何字符。

格式定义只能包含字母 A、X 和 9。

至少要包含一个 X。

若格式字符串包含其他的字符，均当作 X 处理。

- 05 FILLER PIC X(10).
- 05 PHONE PIC AAA99999XX.



### 3.7.3 字母数字编辑型数据

把字母数字数据格式化成特定形式。

使用特定字符表示格式。

➤ 05 EDITED-DATE PICTURE 99/XXX/9999

### 3.7.4 数字型数据

由数字组成

包含隐含的小数点和操作符

只能包含 9、V、S、P 的组合

是少有一个 9，最多 18 个 9

字符 9 表示数字存放的位置，计入数据长度

字符 V 表示假设的小数点，不计入数据长度

字符 S 表示数据项的代数符号，不计入数据长度

字符 P 表示数据项的比例因子，不计入数据长度

### 3.7.5 编辑数字型数据

插入格式字符的数字型。

## 3.8 如何定义数据

使用 PICTURE 从句定义数据

组数据的定义不用 PICTURE 从句，因为下层的数据项可能有不同的数据类型。系统默认为 X 型。



```

01 ZIPCODE PIC 9(6).
01 SALARY PIC 9(5)V99.
01 AMOUNT PIC S9(7)V9(2).
01 DATE.
    05 YEAR PIC 9(4).
    05 MONTH PIC 99.
    05 DAY PIC 9(2).

```

### 3.8.1 USAGE

表示该数据项的用途，同时也指定计算机内存中的数据项格式。

充分利用计算机内存空间和算术功能

可以用于基本数据项和组数据项，基本格式如下：

$$[ \text{U S A G E I S} ] \left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{B I N A R Y} \\ \text{C O M P U T A T I O N A L} \end{array} \right\} \\ \text{C O M P} \\ \text{I N D E X} \\ \text{P A C K E D - D E C I M A L} \\ \text{D I S P L A Y} \end{array} \right\}$$

却省为 DISPLAY，每个字符占 8 位字节。

DISPLAY，表示数据项是用以输出打印的。计算机内部表示方法同其他所有类型的数据项一样，一个数位占一个字节（byte），右半个字节，用来表示数位的实际值，左半个字节叫区码部分，由系统填入适当的值，对于主机系统上面使用的 EBCDIC 码，填写的是 F，即 1111。上述情况是在正数的情况下，如果是负数，系统将把该数的最右一个数位的区码改成另一数 D，即 1101，作为负数的标志，其余保持不变。

BINARY、COMPUTATIONAL、PACKED-DECIMAL 只用于数字数据项

BINARY，项目存放成二进制

COMPUTATIONAL，表示和用法由厂家定义，通常定义为浮点数。机器内部表示为一个定点二进制数，最高的二进制位上的 0 或 1 用来区分正数和负数，其余的二进制位表示数



值。一般用两个字节表示一个定点二进制，其值范围在-32768 到+32767 之间。如果我们要定义的数据项是一个整数，其变化范围不超过二进制的数值范围，经常需要参与运算，那么，定义成这种方式最为合适。

COMP 是 COMPUTATIONAL 的缩写。

### Number of Digits Storage Required.

**1 TO 4 2 Bytes**

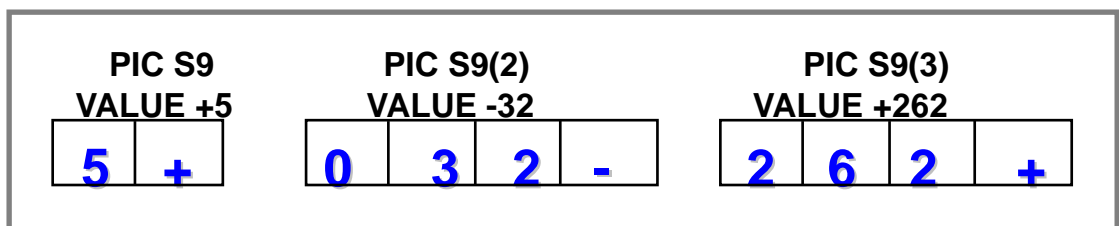
**5 TO 9 4 Bytes**

**10 TO 18 8 Bytes**

**01 TotalCount PIC 9(7) USAGE IS COMP.**

PACKED-DECIMAL，数据按 10 进制存放，减少每个数字的存储量。

COMPUTATIONAL-3，表示紧凑十进制形式的数据项。在计算机内部是去掉不紧凑十进制各位数上的区码部分，只保留数码部分，而在最低数位后用半个字节表示数的正负。记录的长度减小，节省了内外存空间；减少了数在内存中传送的时间。一个数据块中所包含的逻辑记录数量增多，对顺序文件的操作加快。



INDEX，制定该项目为表格索引，采用 INDEX 的项目只能用于过程部的某些语句中，不能有 picture 从句。

### 3.8.2 OCCURS

指定数据项作为固定的项目数组或可变项目数组。用以说明数据项重复出现的次数



$$\begin{array}{l}
 \text{OCCURS} \left\{ \begin{array}{l} \text{整数 1 TIMES} \\ \text{整数 2 TO 整数 3 TIMES } \underline{\text{DEPENDING ON dataname1}} \end{array} \right\} \\
 [ \left\{ \begin{array}{l} \underline{\text{ASCENDING}} \\ \underline{\text{DESCENDING}} \end{array} \right\} \text{ KEY IS dataname2 [ dataname3]...}] \\
 [ \text{INDEXED BY 指标名 1 [指标名 2]...}]
 \end{array}$$

不能用于 01 层和 77 层。

引用时，下标从 1 开始。

采用 **DEPENDING ON** 形式时，其后的数据项名必须是正整数类型，且满足

整数 2 ≤ 数据项名取值 ≤ 整数 3

含有 **OCCURS** 子句或从属于含有 **OCCURS** 子句的数据不能赋初值。

重复的次数有固定不变和可变两种方式，在可变的情况下，重复次数取决于记录中的另一个数据项的内容，即 **DEPENDING ON** 后的数据项名。

对于表示相同类型，相同长度的一组数据，使用 **OCCURS** 可以免去对各数组元素的重复描述。



01 year-salary.

05 month-salary PIC 99999V99 OCCURS 12 TIMES.

组项目	基本项目	数值类型
year-salary	month-salary(1)	99999V99
	month-salary(2)	99999V99
	month-salary(3)	99999V99
	month-salary(4)	99999V99
	month-salary(5)	99999V99
	month-salary(6)	99999V99
	month-salary(7)	99999V99
	month-salary(8)	99999V99
	month-salary(9)	99999V99
	month-salary(10)	99999V99
	month-salary(11)	99999V99
	month-salary(12)	99999V99

项目组总是当作字母数字类型，下例是一个有两个数据项的数据组

01 year-salary.

05 month-salary OCCURS 12 TIMES.

10 salary PIC 9(5)V9(2).

10 bonus PIC 9(5)V9(2).

组项目	组项目	基本项目
year-salary	month-salary(1)	salary(1)
		bonus(1)
	month-salary(2)	salary(2)
		bonus(2)
	month-salary(3)	salary(3)
		bonus(4)
	.....	
	.....	
	.....	



### 3.8.3 SIGN从句

在计算机内部对数据项的符号位的安排作变动，现在已经很少使用。

**SIGN IS TRAILING**，标示符号永远在数的末尾，且和最后一个数位结合起来，没有独立的符号字节。

**SIGN IS LEADING**，标示符号在最前，和第一个数位结合起来，没有独立的符号字节。

**SIGN IS TRAILING SEPARATE** 和 **SIGN IS LEADING SEPARATE**，标示符号从数位中独立出来，单独占有一个字节。

### 3.8.4 SYNCHRONIZED从句

为了使数据在内存用户工作区中的安排更有利于处理而设置的。指定基本数据项在计算机内存自然边界上的对齐。没有使用此从句时，数据项在字节边界对齐。

此从句用来优化计算速度，浪费存储空间

在没有此从句的情况下，数据在内存中的安排采用紧密形式，即某一文件的记录区中是一个数据项之后紧接着下一个数据项。这样，会出现一个字的几个字节分属于两个或多个数据项的情况，在处理时要加以分解，增加额外的操作开销。此从句是用来解决这个问题，提高运行效率。它自动在计算机内部增加一个“空闲字节”使得每个数据项都从字的边界开始。

**SYNC** 后面的 **LEFT**，和 **RIGHT** 实际是不起作用的，只为文档的需要设置。

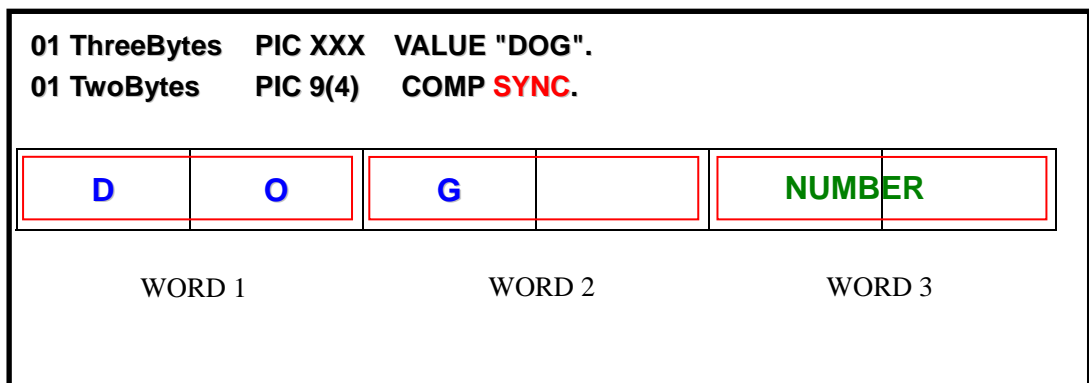
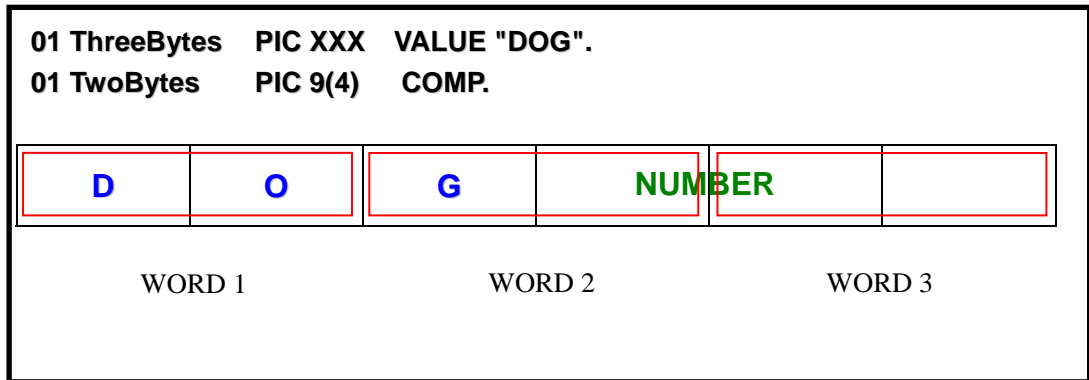
**COMP SYNC (1 TO 4 Digits)** = 2 Byte boundary

**COMP SYNC (5 TO 9 Digits)** = 4 Byte boundary

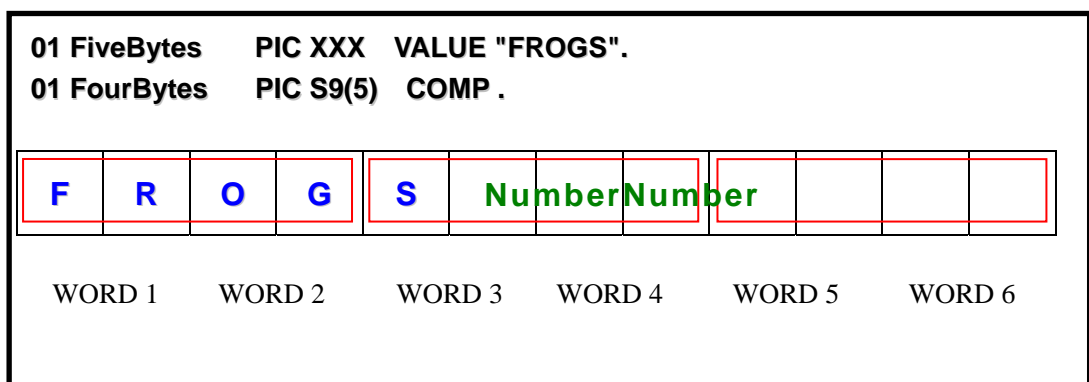
**COMP SYNC (10 TO 18 Digits)** = 8 Byte boundary

**INDEX** = 4 Byte boundary

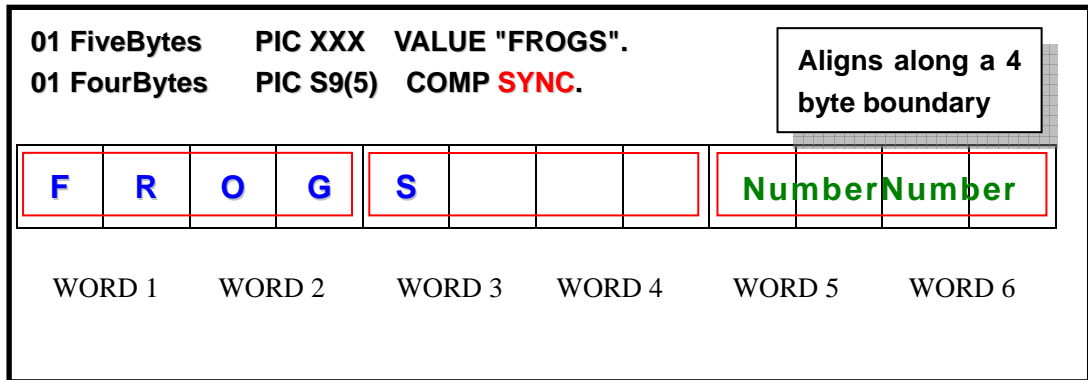
例 1:



例 2:







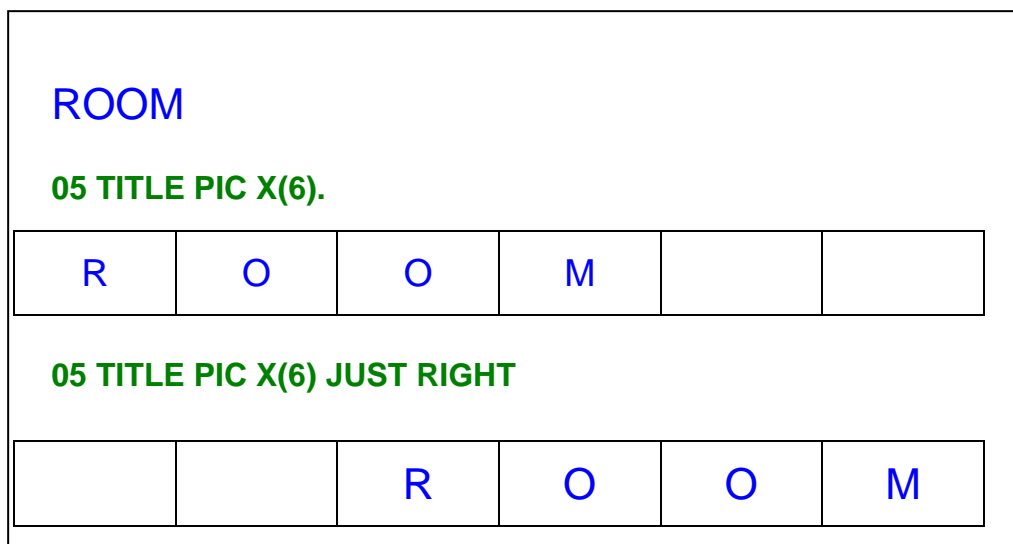
### 3.8.5 JUSTIFIED

只能用于非编辑的字母数字基本数据项

表示数据右对齐

数据太长时，左边截断，数据太短时，左边填充。

例 3:



例 4:



PARADISE					
05 TITLE PIC X(6).					
P	A	R	A	D	I
05 TITLE PIC X(6) JUST RIGHT					
R	A	D	I	S	E

### 3.8.6 REDEFINE从句

用不同的数据描述同一计算机内存。使同一工作区有不同的名和不同的组成结构。

适用于基本数据项及组数据项。

定义的区域长度小于或等于原区域的长度。

不能有 VALUE 从句

文件节或报表节的 01 层不能用此从句。

REDEFINE 定义的数据项不能包含 OCCURS 从句

```
01 RECORD-1.  
    05 WK-DATE      PIC X(8).  
    05 WK-DATE-RED REDEFINES WK-DATE-99.  
        10 WK-DATE-YEAR      PIC 9(4).  
        10 WK-DATE-MONTH     PIC 9(2).  
        10 WK-DATE-DAY       PIC 9(2).
```

### 3.8.7 VALUE从句

用来指定数据项的初始值，使用时格式如下：



VALUE IS literal

符合数据项的格式定义

对于数据项组项目，数值应该是非数字值。

VALUE 从句下不能再有 VALUE 从句。

```
01 DATE VALUE 20040908.  
    05 YEAR PIC 9(4).  
    05 MONTH PIC 99.  
    05 DAY PIC 9(2).  
  
01 CUSTNAME PIC X(8) VALUE ALL SPACES.  
  
77 COUNT PIC 999 VALUE 1.
```

### 3.9 66层定义

其格式如下：

66 dataname1 RENAMES dataname2 { THROUGH } Dataname3  
  THRU }

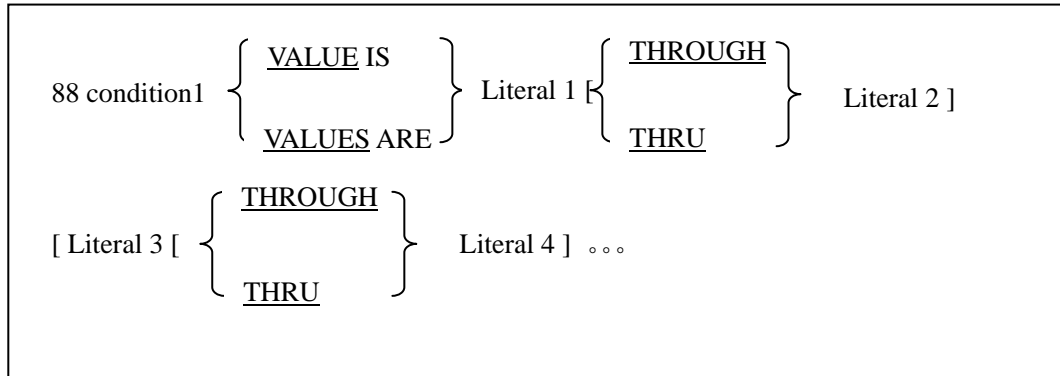
其意义和 REDEFINES 类似，也是为了节省内存，不同数据使用同一工作区。与 REDEFINES 的区别在于

- 重命名从句必须用于有特殊层号 66 的数据
- 才用重命名从句时，只起换名的作用，工作区的结构和字形不变。



### 3.10 88层定义

88 层数据不是独立的数据，它从属于另一个称作条件变量的初等数据项，用来表示它所根据条件取定的值（或在预定的域中取值）。其格式基本如下：



其范例如下：

```
01 MONTH PIC XXX.  
88 FIRST-KIND          VALUES ARE  
"JAN","MAR","MAY","JUY","AUG","OCT","DEC".  
88 SEC-KIND  VALUES ARE "APR","JUN","SEP","NOV".  
88 TRD-KIND  VALUE  "FEB".  
  
01 InputChar  PIC X.  
88 Vowel      VALUE  "A","E","I","O","U".  
88 Consonant  VALUE  "B" THRU "D", "F","G","H"  
                  "J" THRU "N", "P" THRU "T"  
                  "V" THRU "Z".  
88 Digit      VALUE  "0" THRU "9".  
88 LowerCase  VALUE  "a" THRU "z".  
88 ValidChar  VALUE  "A" THRU "Z", "0" THRU "9".
```



## 第五章 Sequential File

- 了解基本概念
- 掌握如何读写文件的记录
- 掌握如何读、写、打开及关闭文件



## 1 一些基本概念

### 1.1 Files, Records, Fields.

FIELD —— 文件中的数据结构

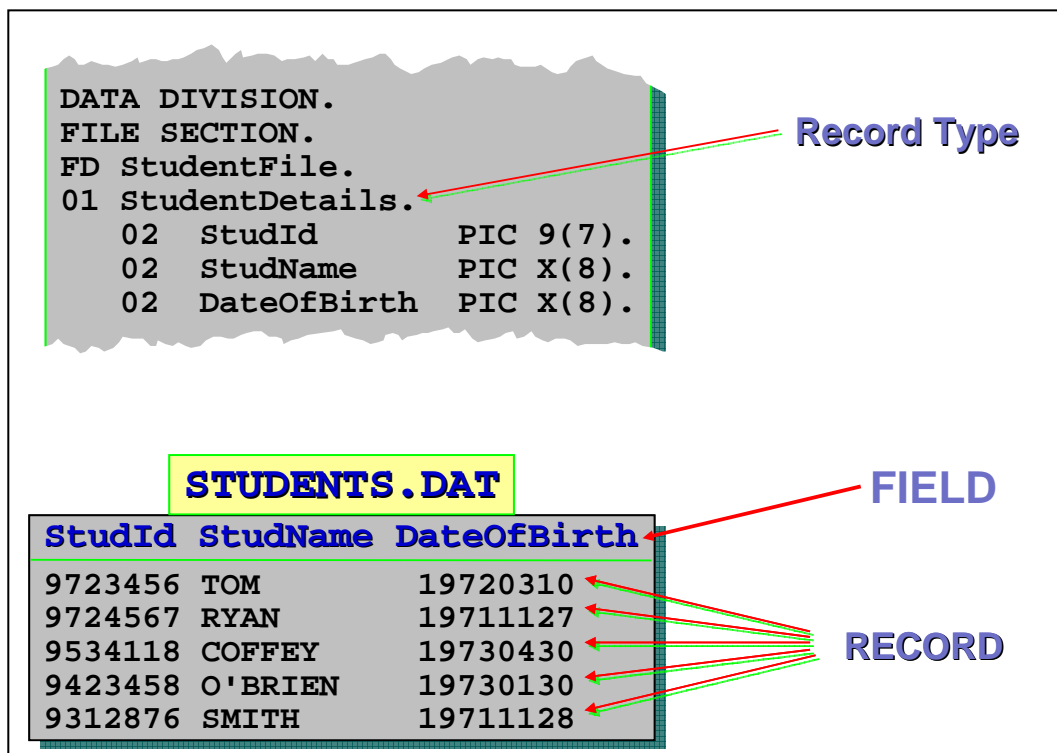
(例如: StudentName, DateOfBirth, CourseCode).

RECORD —— 文件的数据记录

(例如: StudentRecord 是有关学生的所有属性的一条记录信息).

FILE —— 一组记录的集合

Files, Records, Fields.



### 1.2 文件的类型

#### 1.2.1 顺序文件

记录从头到尾顺序列出，只能按顺序访问其中的记录。现有记录间不能插入，删除记录。在末尾增加新记录。



### 1.2.2 相对文件

记录按逻辑记录号引用，称为相对关键字或相对记录号。通过指定相对关键字按任意顺序处理。

### 1.2.3 索引文件

记录按字母数字关键字引用，系统保持一个索引。可以随机访问，可以随时改写和删除。索引文件是现在数据库系统的基础。

## 1.3 文件的访问方式

顺序访问——从头到尾访问记录。

顺序文件、相对文件、索引文件

随机访问——按随机顺序处理记录

相对文件、索引文件 先指定关键字

动态访问——同时利用顺序和随机处理

相对文件、索引文件 先指定关键字

## 2 如何处理文件

文件是程序内存外的一组记录，保存在硬盘或磁带上。因此，文件是存放在外部存储介质上，并按照一定方式组织起来的记录集合。

文件的内容可能很多，可能有上百万或千万条记录。处理时不可能将这么多的记录都读入内存。因此，处理文件时，是一次读入一条记录（RECORD）来处理。

### 2.1 记录缓存区

处理文件时，每次将文件的一条记录读入计算机内存来处理。在内存中找不到与文件对应的存储区，内存中最大的存取项不是文件而是记录。对文件的处理本质上可以看作是对记录的处理。

计算机通过程序员描述的记录格式，设置内存空间存放读入的记录。

用来存放记录的内存空间叫做“记录缓存区”。即计算机为文件中的记录，分配的一



块公用区域。被处理的记录临时占用这块区域，处理完后被放回文件，为其他待处理的记录让出这块公用区。

## 2.2 缓存区的使用

程序处理文件时，每次只读入一条记录。

在处理输入文件的时候，要处理的记录必须读入到缓存区。

在建立输出文件时，文件中的记录应该先放置在缓存区，然后写入文件。

将一条记录从输入文件写到输出文件中

- 讲述入文件的记录读入到输入缓存区
- 将记录从输入缓存区读出放到输出缓存区
- 将记录从输出缓存区写入输出文件

## 2.3 使用文件

文件不是拿过来就可以读写的，需要事前定义。

环境部——定义文件名，将文件名与程序外的实际文件相联系，同时定义文件的组织和访问方式。

数据部——定义文件的描述符，将文件名与其结构相联系。定义 record 的格式。

过程部——对文件进行处理。

### 2.3.1 环境部定义

通知计算机，文件具体的存放位置，说明文件和某一外设之间的联系。





```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT StudentFile
        ASSIGN TO "STUDENTS.DAT".
    FILE STATUS IS record-status.
```

除非程序不使用文件，环境部中可以省略这一节，否则，这节必须描述。

### 2.3.1.1 输入输出节

输入输出节由两段组成，完成我们所需要的功能的是文件控制段。

```
FILE CONTROL.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT [OPTIONAL] FileName ASSIGN TO ExternalFileReference
    [RESERVE Integer [ AREA/AREAS ] ] [FILE STATUS IS DataName]
```

SELECT FileName 是数据部文件节定义的内部文件名

ASSIGN TO ExternalFileReference 将内部文件名连接到外部文件

RESERVE AREAS 指定文件缓存区以提高效率，对程序逻辑没有影响

FILE STATUS 指定文件进行 I-O 操作后，存放操作状态的数据项

SELECT 语句是把指定的文件名和设备名联系起来，以通知计算机，文件放在该设备上。

一条 SELECGT 语句只能描述一个文件，若程序中涉及多个文件时，应该一个一个用 SELECT 语句描述。

### 2.3.2 数据部定义

程序中用到的每一个文件，其记录格式必须在环境部的文件节（ FILE SECTION ）



通过 FD 语句来定义。

基本格式如下：

```
DATA DIVISION.  
FILE SECTION.  
FD filename  
    文件描述项.  
    记录描述项.
```

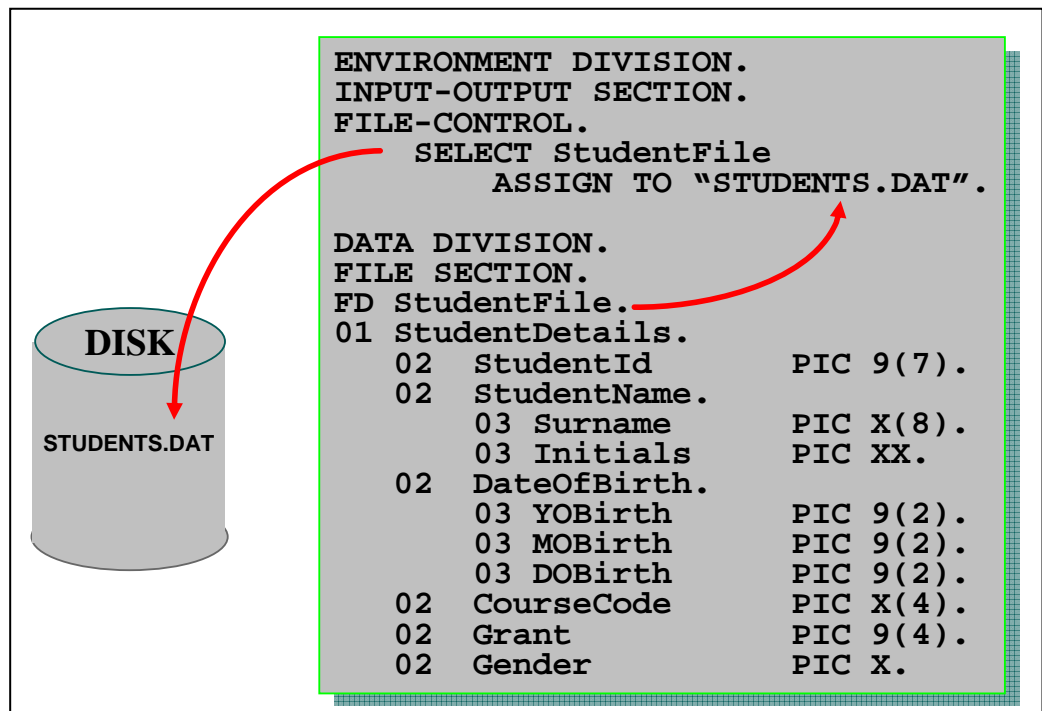
文件的记录描述项如下例：

```
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
01 StudentDetails.  
    02 StudentId          PIC 9(7).  
    02 StudentName.  
        03 Surname          PIC X(8).  
        03 Initials        PIC XX.  
    02 DateOfBirth.  
        03 YOBirth          PIC 9(2).  
        03 MOBirth          PIC 9(2).  
        03 DOBirth          PIC 9(2).  
    02 CourseCode          PIC X(4).  
    02 Grant                PIC 9(4).  
    02 Gender               PIC X.
```

FD 语句由 FD 和文件的内部文件名组成。

### 2.3.3 两者的关系

在环境部通过 FD 定义的内部文件名，通过 Select 和 Assign 从句与外部实际文件相连。



## 2.3.4 如何定义文件

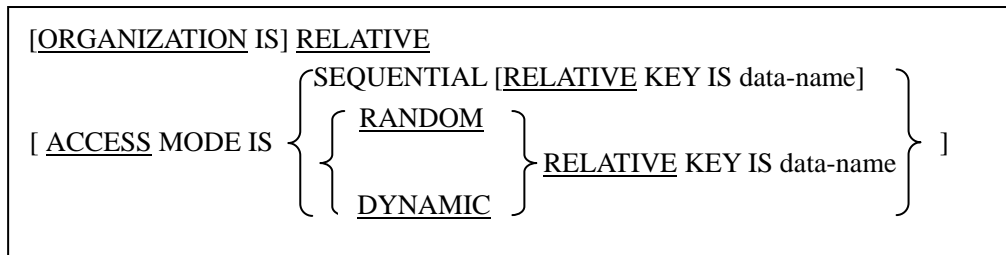
### 2.3.4.1 顺序文件

```
[ORGANIZATION IS SEQUENTIAL]  
[ACCESS MODE IS SEQUENTIAL]  
[PADDING CHARACTER IS id-lit]  
[ RECORD DELIMITER IS { STANDARD  
                        Char-name } ]
```

ORGANIZATION	可选，默认为顺序文件
ACCESS MODE	可选，顺序文件只允许顺序访问
PADDING CHARACTER	指定文件在固定块长的设备上所用的字符
RECORD DELIMITER	指定用什么字符区分变长记录

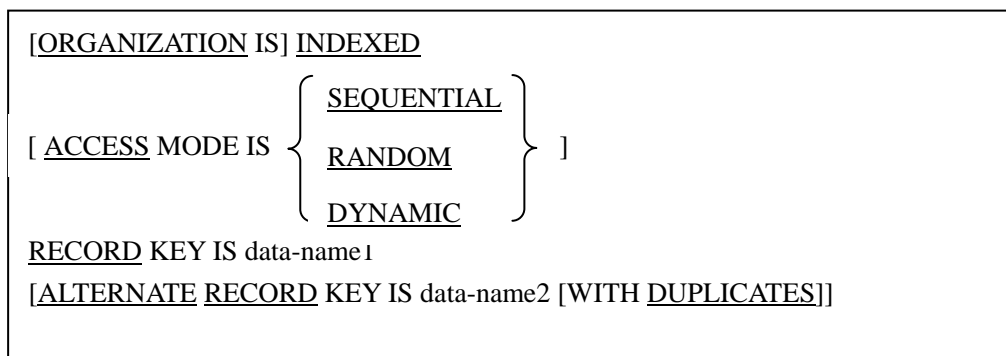


#### 2.3.4.2 相对文件



ORGANIZATION IS RELATIVE	必须项
ACCESS MODE	省略方式为顺序访问
RELATIVE KEY	指对顺序访问是可选，存在时，文件读取操作完成后，data-name 的值更新

#### 2.3.4.3 索引文件



ORGANIZATION IS INDEXED	必须项
ACCESS MODE	省略方式为顺序访问
RECORD KEY	指定文件的主记录关键字
ALTERNATE RECORD KEY	如果文件有替换关键字，用此句指定，允许重复关键字时，使用

#### 2.3.5 文件的处理

对文件的操作，基本分为两类，即读和写。要实现这两类功能，有以下操作：



## OPEN

当程序在处理输入的文件，或输出的文件前，必须通过 **OPEN** 语句打开文件。

## READ

**READ** 语句将文件中的一条记录（record）放到记录缓存区中

## WRITE

**WRITE** 语句将记录缓存区中的记录内容写入到文件中。

## CLOSE

在程序结束时需要用 **CLOSE** 语句将所有打开的文件关闭。否则会导致数据无法写入文件，或者其他用户无法使用文件。

### 2.3.5.1 OPEN 语法

由于文件与计算机系统的硬件紧密相连，因此，若使用外部介质，需要对其做有关的准备工作和结束工作，准备工作是指，使该设备处于可运行状态，在 **COBOL** 中，是通过打开文件实现的，只有先打开文件，使其处于可读或可写状态，我们才能对文件进行读或写的工作。

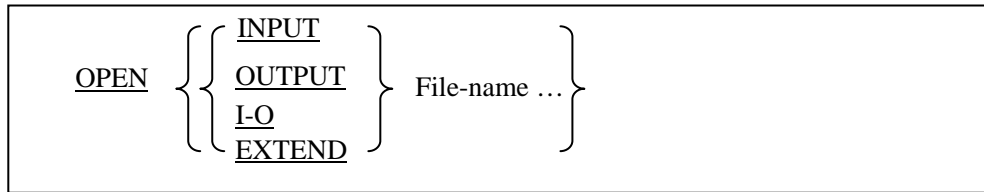
根据环境部文件控制段的 **SELECT** 语句和 **ASSIGN** 语句检查，指明该文件所在的外部设备是否联机，是否处于可用状态，否则会给出出错信息，退出程序。

检查在指定的外部设备上是否有指定的文件，如果没有，给出出错信息，退出程序。

上述两步通过了，则为该文件动态的分配一输入输出缓冲区。

将文件头（File Header）调入内存，检查其中的管理信息，看文件处于何种使用状态，若文件已经处于打开状态，则给出出错信息。因为，一个文件在一个时候，只能由一个用户打开使用，以免出现“并发操作错误”。如果文件原来处于关闭状态，则本次 **OPEN** 命令有效，在文件头中写入有关本次使用的信息，并将系统指针置于文件的第一个记录。

其基本格式如下：



- INPUT**      输入文件，文件只能读取。写入、修改、删除记录会发生错误。若打开的文件不存在，则产生错误，除非，在环境部 **FILE-CONTROL** 声明文件为 **OPTIONAL**，则产生没有记录的新文件。
- OUTPUT**    输出文件，生成要写入记录的新文件。读取文件会产生错误。若打开的文件已存在，则删除所有现存的纪录。
- I-O**        文件可以读取也可以写入。若打开的文件不存在，则产生错误，除非，在环境部 **FILE-CONTROL** 声明文件为 **OPTIONAL**，则产生没有记录的新文件。
- EXTEND**    打开顺序文件，    只能写入文件。将记录插入到现有文件的末尾。除非在环境部 **FILE-CONTROL** 声明文件为 **OPTIONAL**，产生没有记录的新文件。否则，打开不存在的文件，操作不成功。

在开始处理文件之前，必须执行 **OPEN** 语句，以保证对其进行 **READ** 和 **WRITE** 操作的正确，否则会出现系统错误。

一条 **OPEN** 语句可以打开一个或多个文件，而且可以选择不同的打开方式。例如：

<pre>OPEN INPUT STUDENT1 STUDENT2. OPEN OUTPUT STUDENT3. 等同于 OPEN INPUT STUDENT1       INPUT STUDENT2       OUTPUT STUDENT3.</pre>
--

### 2.3.5.2      **CLOSE** 语法

当一切操作结束后，应当关闭文件，只有关闭了文件，才能保证文件中的信息不被意外的错误操作而毁坏，同时，也放弃了对内存的占用，为处理其他文件提供更多的自由空间。



将文件输入输出缓冲区的内容写入文件，以便保留在此之前对文件操作的结果。

将文件头调入内存，修改其中的管理信息，注销用户的本次使用，然后写入文件。

将文件使用的输入输出缓冲区释放，交还给系统。

其格式如下：

**CLOSE** { file-name [ **WITH LOCK**] } ...

终止对文件的处理。

已打开的文件，需要 **CLOSE** 才能在同一程序中再次打开。

可以用一句 **CLOSE** 语句关闭多个文件，视同每个文件用一个 **close** 语句关闭。文件的组织方式和存取方式可以是互不相同的。

文件关闭的顺序，按照 **close** 语句中的顺序进行。

程序在结束运行之前必须用 **CLOSE** 命令关闭所有打开的文件。在程序中，文件的打开和关闭必须成对出现。

下例是一个打开关闭文件的例子



```
PROGRAM-ID.      BZJCACCP.
      ENVIRONMENT      DIVISION.
      INPUT-OUTPUT    SECTION.
      FILE-CONTROL.
SELECT CZFFBZJ ASSIGN TO  AS-CZFFBZJ
                        ORGANIZATION  IS    SEQUENTIAL
                        ACCESS MODE    IS    SEQUENTIAL
                        FILE  STATUS    IS    IN-FILE-STATU.

DATA      DIVISION.
      FILE SECTION.
      FD CZFFBZJ
          LABEL RECORDS ARE STANDARD
          DATA RECORD IS CZFFBZJ-RECORD.
          COPY CZFCBZJ.
WORKING-STORAGE SECTION.
01 SW-CZFCBZJ-EOF PIC X    VALUE 'F'.
      88 SW-END-CZFCBZJ      VALUE 'T'.
PROCEDURE  DIVISION.
000-OPEN-INPUT-FILES.
      OPEN INPUT CZFFBZJ.
      IF IN-FILE-STATU NOT = '00' THEN
          MOVE 'CZFFBZJ' TO FILE-NAME
          MOVE IN-FILE-STATU TO ERR-FILE-STATU
          PERFORM 810-OPEN-ERROR
      END-IF.
```

### 2.3.5.3 READ 语法

打开文件只是为真正读写文件做准备工作，而不是一打开文件就可以把记录真正的写入记录区。打开文件只是使文件处于可处理状态，这时，可以使用 READ 语句将当前的记录读入内存，作各种处理工作。

READ 语句的功能是从文件中读取一条记录，放置到文件缓存区中。

读取前，文件必须先用 OPEN 语句打开。文件必须以 INPUT 或 I-O 方式打开，才能用于读取。

文件的读取方式有两种：对顺序文件以顺序访问方式或动态访问方式读取；对随机访问方式文件读取。

#### 顺序读取文件





这里访问的文件，其记录都是顺序的，在文件中，系统安置了指针。文件打开后，指针处于第一个记录的第一个字节，读取一条记录后，指针自动移到第二条记录的第一个字节。我们把指针所指的记录称为当前记录。执行 **READ** 语句，就是把当前记录拷贝到文件对应的记录区中。

格式如下：

```
READ internal-file-name [NEXT] RECORD [ INTO ident]  
[AT END statements]  
[NOT AT END statements]  
[END-READ]
```

若文件声明为顺序访问方式，则必须有 **NEXT** 语句。

**INTO** 将数据缓存区中的内容拷贝到 **ident** 中。

如果 **READ** 成功，执行 **NOT AT END** 语句。

如果文件中不再有记录，无法执行 **READ** 语句，则执行 **AT END** 后的语句。当指针已经移到最后一条记录的最后一个字节之后，再执行 **READ** 语句，已经无记录可以处理，此时，程序将执行 **AT END** 后面的语句串，其作用是防止在文件记录结束后，再读取无效的记录。

动态访问方式，需要先建立文件的位置，然后再执行顺序 **READ** 语句。

### 随机读取文件

格式如下

```
READ internal-file-name RECORD [ INTO ident]  
[INVALID KEY statements]  
[NOT INVALID KEY statements]  
[END-READ]
```



- 如果是相对文件，则要先设置 FILE-CONTROL 中的 RELATIVE KEY 指定的整型关键字的值。
- 如果是索引文件，则要先设置索引的关键字值中不再有记录，无法执行 READ 语句，则执行 AT END 后的语句
- 动态访问方式，需要先建立文件的位置，然后再执行顺序 READ 语句。

#### 2.3.5.4 WRITE 语法

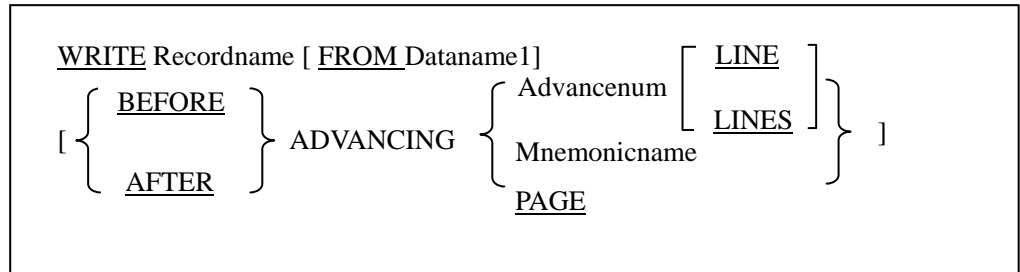
WRITE 语句将数据从缓存区写入文件中。即在文件中添加一个新的记录。在添加新记录的过程中，要注意以下两点：

必须提供要写入的记录名

记录名必须属于数据部文件节中定义的文件。

写文件，根据文件的种类不同，有两种格式

顺序文件



顺序文件的 WRITE 语句将记录放在文件末尾。

若指定 FROM 短语，则将 Dataname1 中的数据复制到 RecordName 中，然后再写入文件。

BEFORE 和 AFTER 支队支持行和页结构的文件有效。

相对文件/索引文件



```
WRITE RecordName [ FROM identifier]  
[ INVALID KEY statements ]  
[ NOT INVALID KEY statements ]  
[ END-WRITE ]
```

WRITE 语句使用的是记录名而不是文件名，跟其他三条语句不一样。因为 WRITE 语句是将内存中的记录而不是文件内容输出。内存记录区是以记录为单位存放数据的。

每执行一次 WRITE 语句，只能数出一条记录，不能用一条 WRITE 语句输出多条记录。

### 相对文件

要随机方式或动态方式写入文件，文件应以 OUTPUT 或 I-O 方式打开

要顺序方式写入文件，文件应以 OUTPUT 或 EXTEND 方式打开

OUTPUT 方式，记录写入文件的开头，并赋予从 1 开始的相对关键字的顺序。

EXTEND 方式，记录写入文件的末尾，并制定从现有最高关键字后一个整数开始的相对关键字。

顺序访问，若 FILE-CONTROL 中指定了 RECORD KEY 数据项，则执行完 WRITE 后， RECORD KEY 数据项更新为刚刚写入的记录相对记录号。

### 索引文件

要随机方式或动态方式写入文件，文件应以 OUTPUT 或 I-O 方式打开

要顺序方式写入文件，文件应以 OUTPUT 或 EXTEND 方式打开，记录按主关键字升序写入。

写入前，必须先提供记录的关键字值

OUTPUT 方式，记录写入文件的开头。

EXTEND 方式，记录写入文件的末尾，主关键字要大于文件中现有的关键字。

执行完 WRITE 后，与文件相关联的所有索引更新。



## 第六章 过程部结构及一些基本语法

- 介绍过程部的命令分类
- 命令的基本语法
- 学会使用这些语法



从本章开始，我们学习 COBOL 过程部。在过程部中，我们要利用允许的命令语句写出为实现制定的数据处理目标所取的步骤。

首先，我们会介绍过程部的结构，然后介绍 COBOL 语言的基本命令，学习了这些命令之后，我们就可以编写一般常用功能的程序了

## 1 命令的基本分类

过程部的命令，基本可以分为以下几类：数据传输命令、人机通讯命令、算术运算命令、文件操作命令、过程分支命令，字符串操作命令、条件语句、动态调试命令、编译程序引导命令。

文件操作命令，我们在上一章已经介绍过了，本章主要介绍数据传输命令，人机通讯命令，过程分支命令，字符串操作命令，下章介绍算术运算命令。

## 2 数据传输命令

### 2.1 MOVE

数据传输命令 MOVE 的功能是将数据从发送数据项复制到一个或多个数据项中。MOVE 语句并不把一个数据项移动到另一个地方，原先的数据项保持不变。

COBOL 语言中的 MOVE 相当于其他语言中的赋值命令，但是 MOVE 命令允许将一个值同时赋给多个变量，或者允许对多个变量同时赋值，远较其他语言中赋值命令方便，灵活。

其基本格式如下：

MOVE Ident1 TO { Ident2 } ...

其说明事项如下：

#### 2.1.1 移动数字或数字编辑型数据项

若接收项为数字型或数字编辑型，则发送项也应为数字型或数字编辑型。

若发送项是字母数字项，则作为无符号整数，若字段包含非数字字符，则结果不定

若发送项是数字编辑型，则取出数字值，用作发送项，若接收项长度小于发送项，



则在接受项的两端根据需要截断；若接收项大于发送项，则在接受项的两端根据需要加上 0

若接收项是数字编辑型，则按照格式进行

若接收项是数字型，发送项转化为按照接收项的 USAGE 从句指定的格式

### 2.1.2 移动字母数字、字母数字编辑型、字母项目

若接收项为字母数字或字母数字编辑型，发送项可为任何类别，若发送项是数字，则必须为整数

若接收项为字母，则发送项不能是数字或数字编辑型

接收项为字母数字，字母数字编辑型或者字母型，没有指定数据对齐方式时，默认左对齐，若接收项小于发送项，截断发送项尾部

若发送项是带符号的数字，则部移动符号，符号也不计入发送项目的长度

若发送项是数字编辑项，则不删除编辑，作为字母数字项

若发送项形如“ALL non-numeric literl”，则非数字直接数本身结合，直到接收项目的长度

例如

```
05 TEMP-WORD PIC X(11)
...
MOVE ALL" ABC" TO TEMP-WORD
```

#### TEMP-WORD

A	B	C	A	B	C	A	B	C	A	B
---	---	---	---	---	---	---	---	---	---	---

## 2.2 MOVE CORRESPONDING



其基本格式如下：

```
MOVE CORRESPONDING ident1 TO ident2  
MOVE CORR ident1 TO ident2
```

ident1 和 ident2 通常指组合数据项或记录，即组项目，他们所属的某些或全部数据项有相同的名称，执行这一格式的 MOVE 命令时，同名数据项内容对应传送，所以是成组传送。

表示移动多个同名数据项。

CORR 是 CORRESPONDING 的缩写。

语句的功效等同于每个所选项目的 MOVE 语句。

两个标识符不应代表同一数据区域，即不应是同一文件的两个 01 层的记录名，不应该由 66 层重命名的两个数据名，不应是同一记录中由 REDEFINES 重订一的两个数据名。

数据传送一般在同类数据项之间进行，虽然大多数的系统有条件的允许不同类型的数据项之间也可以传送，但是为了避免数据传送后发生畸变，应该尽量避免。

### 3 人机通讯命令

这类命令允许在程序的执行过程中，程序员向某一工作单元传送数据（ACCEPT），程序向程序员展示某一工作单元当前内容或者向程序员展示某种信息（DISPLAY），这类命令也可以认为是少量数据的输入输出传送。

#### 3.1 ACCEPT

用来接收数据的命令，格式如下：

```
ACCEPT ident [ FROM mnemonic-name ]
```

从键盘或其他设备读取数据



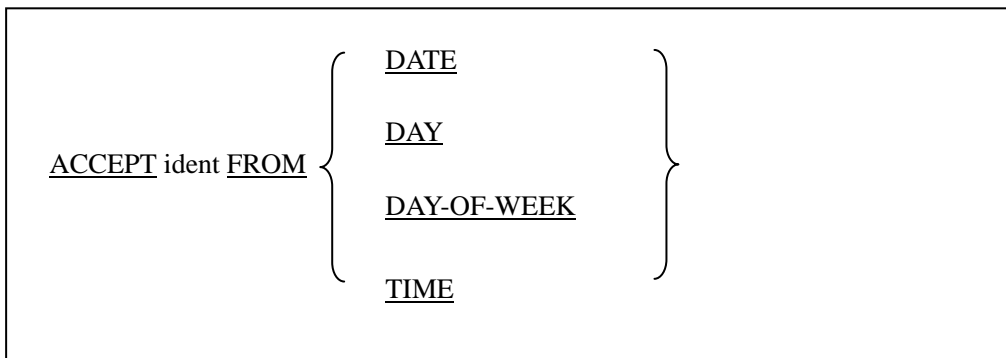
可以用 from 指定输入设备

用来将数据设备中的少量数据移到程序中

可以取得系统的日期和时间

## 取得时间和日期

其格式如下：



DATE，取系统当前日期，6 个字符长，按照年月日排列

DAY，取系统当前日期，6 个字符长，前 2 个字符是本世纪的年份，后 3 位是本日在当年的天数

DAY-OF-WEEK，取系统当前星期几，1 个字符长

TIME，取系统当前时间，8 个字符长，1-2 位表示小时，3-4 位表示分，5-6 位表示秒，最后两位表示百分之几

例：

系统时间

2004 年 2 月 1 日星期天下午 2： 41





使用不同的语句得到不同的结果

语句	结果
ACCEPT WORK-DATE FROM DATE	040201
ACCEPT WORK-DAY FROM DAY	04032
ACCEPT WORK-WEEK FROM DAY-OF-WEEK	7
ACCEPT WORK-TIME FROM TIME	14410000

### 3.2 DISPLAY

显示命令的格式如下：

```
DISPLAY { id-lit } ... [ UPON mnemonic-name ] [ WITH NO ADVANCING ]
```

此命令的功能是将数据写入输出设备，如监视器、系统输出流等。每个 id-lit 的内容，按照顺序写入输入设备。

若使用 NO ADVANCING，则输出最后一个字符后，输出设备仍保持原位，下一条 display 语句在该位置输出数据。

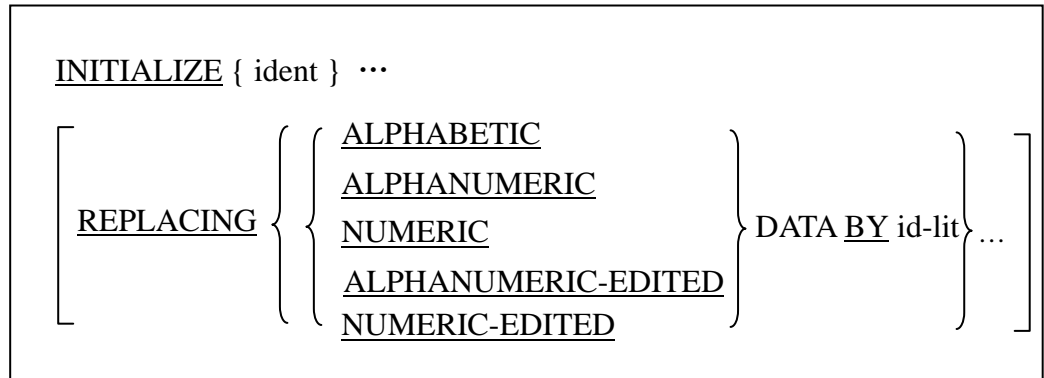
此命令主要用于以下两种情况：

- 在程序运行的关键点上显示某些数据内容，以供检查，方便调试
- 在对话式程序中用以向用户提供菜单或作出某些提示。

### 3.3 INITIALIZE

将数据项的值设为初始值

其格式如下：



若不指定要设置的数值，则按照数据项的类型分别设置为 0 或空格。作用于组项目时，相当于将组中所有的基本项目都初始化

## 4 字符串操作命令

### 4.1 INSPECT

检查数据项，计算该项目中某个字符串出现的次数，或者用另一字符串替换此字符串。

此语句有四种形式：

INSPECT TALLYING

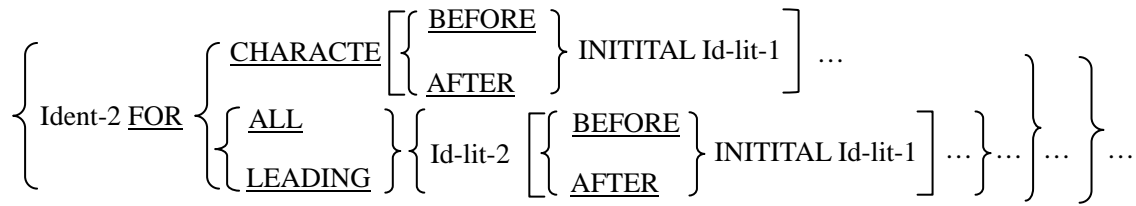
INSPECT REPLACING

INSPECT TALLYING REPLACING

INSPECT CONVERTING

#### 4.1.1 INSPECT TALLYING

其基本格式如下：

**INSPECT Ident-1 TALLYING**

其功能为计算 **ident-1** 中字符串出现的次数，并将其加入到 **ident-2** 中。

**ident-2** 为数字型数据项，不初始化为 0，每次匹配时，**ident-2** 的值递增

**CHARACTERS**，**ident-1** 中的每一个字符都算匹配，相当于计算字符数

**ALL**，将 **id-lit-2** 与当前位置相同个数字符比较，若匹配，则 **ident-2** 递增 1，从匹配项后一个字符开始再次检查。

**LEADING**，将 **id-lit-2** 与当前位置相同个数字符比较，只计算 **ident-1** 开头的，其他都不算作匹配项。

一旦找到匹配项，**ident-1** 的当前位置向前移动这个字符串的长度

**ALL**, **LEADING**, **CHARACTERS** 后面只能跟一个 **BEFORE** 或 **AFTER** 短语。

**BEFORE** 设置 **ident-1** 中停止比较点。

**AFTER** 设置 **ident-1** 种开始比较点。

若在一行 **INSPECT TALLYING** 语句中组和 **CHARACTERS**、**ALL** 或 **LEADING**，则按照语句指定的顺序，连续采用每个标准检查，找到匹配后，**ident-1** 的当前位置移动，再次从第一个操作数开始。

## 例 4.1.1.1

**STRINGDATA**

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

各种统计结果



语句	结果
INSPECT STRINGDATA TALLYING COUNTER FOR ALL “EE”	2
INSPECT STRINGDATA TALLYING COUNTER FOR CHARACTERS BEFORE “L”	3
INSPECT STRINGDATA TALLYING COUNTER FOR LEADING “E”	0
INSPECT STRINGDATA TALLYING COUNTER FOR LEADING “E” AFTER “H”	2
INSPECT STRINGDATA TALLYING COUNTER FOR CHARACTERS AFTER “E” BEFORE “W”	2

例 4.1.1.2

#### STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

各种统计结果

语句	结果
INSPECT STRINGDATA TALLYING	
COUNTER1 FOR ALL “E”	4
COUNTER2 FOR LEADING “W” AFTER “L”	1
COUNTER3 FOR CHARACTERS	4

#### 4.1.2 INSPECT REPLACING

其格式如下：



INSPECT Ident-1 REPLACING

$\left\{ \begin{array}{l} \text{CHARACTERS BY id-lit-3} \left[ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL Id-lit-1} \dots \\ \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{FIRST} \end{array} \right\} \left\{ \text{Id-lit-4 BY id-lit-3} \left[ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL Id-lit-1} \dots \right\} \dots \end{array} \right\}$

其功能是：找到 ident-1 中与字符串匹配，并将匹配字符串替换成指定标识符或直接数的内容。

#### 例 4.1.2.1

##### STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

INSPECT StringData REPLACING ALL “E” BY “Y”.

结果 STRINGDATA:

R	Y	Y	L	W	H	Y	Y	L
---	---	---	---	---	---	---	---	---

#### 例 4.1.2.2

##### STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

INSPECT StringData REPLACING CHARACTERS BY “Z” AFTER INITIAL “L”.

结果 STRINGDATA:

R	E	E	L	Z	Z	Z	Z	Z
---	---	---	---	---	---	---	---	---



例 4.1.2.3

STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

INSPECT StringData REPLACING LEADING “E” BY “Z”.

结果 STRINGDATA:

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

例 4.1.2.4

STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

INSPECT StringData REPLACING LEADING “E” BY “Z” AFTER “H”.

结果 STRINGDATA:

R	E	E	L	W	H	Z	Z	L
---	---	---	---	---	---	---	---	---

例 4.1.2.5

STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

INSPECT StringData REPLACING FIRST “E” BY “Z”.

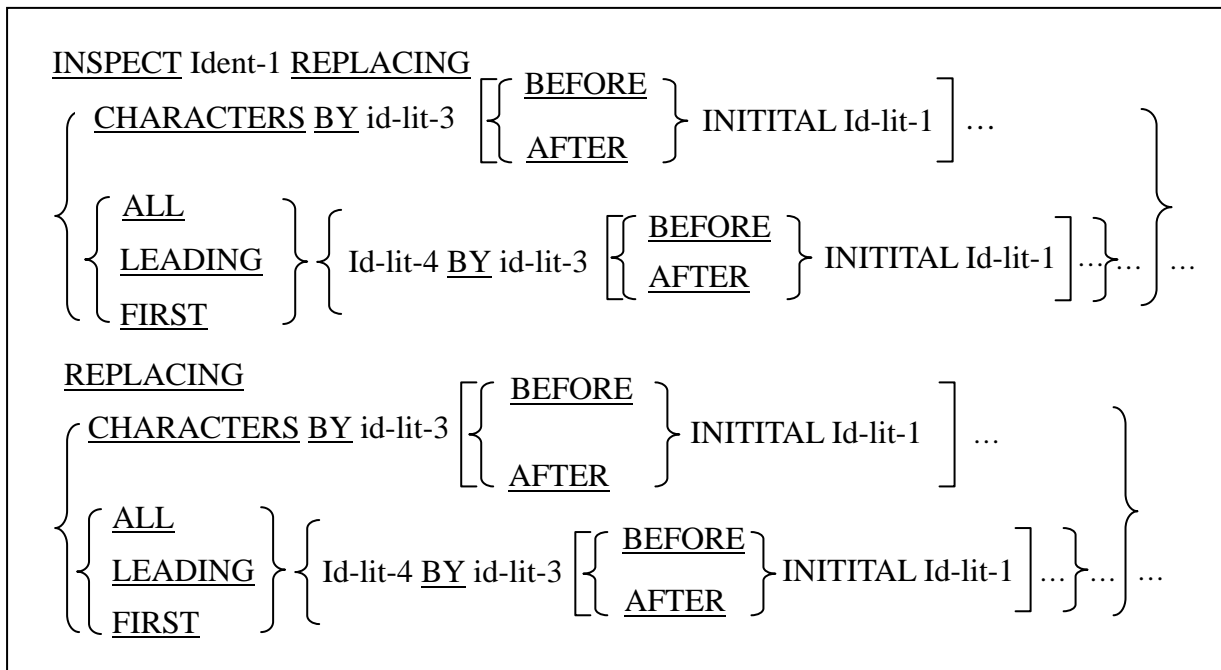
结果 STRINGDATA:

R	Z	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

### 4.1.3 INSPECT TALLYING REPLACING

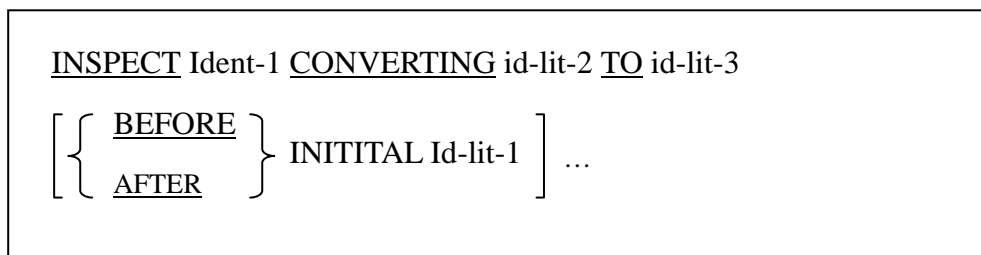


格式如下：



#### 4.1.4 INSPECT CONVERTING

格式如下：



其功能相当于 INSPECT REPLACING 的缩写,采用 ALL 选项,用 id-lit-3 替换 id-lit-2

id-lit-2 与 id-lit-3 长度相同, id-lit-2 中的字符不能重复

##### 例 4.1.4.1

STRINGDATA

R	E	E	L	W	H	E	E	L
---	---	---	---	---	---	---	---	---

**INSPECT StringData CONVERTING “EL” TO “ZY”.**

结果 STRINGDATA:

<b>R</b>	<b>Z</b>	<b>Z</b>	<b>Y</b>	<b>W</b>	<b>H</b>	<b>Z</b>	<b>Z</b>	<b>Y</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------

**4.1.5 INSPECT 如何工作**

INSPECT 从左到右检查源字符串，然后依据使用的 TALLYING, REPLACING 或 CONVERTING 短语 统计或替换相应的字符。

INSPECT 依据 LEADING, FIRST, BEFORE 和 AFTER 决定相应的操作。

An ALL, LEADING, CHARACTERS, FIRST or CONVERTING phrase may only be followed by one BEFORE and one AFTER phrase.

**4.2 STRING**

其格式如下:

$$\text{STRING} \left\{ \left\{ \text{id-lit-1} \right\} \dots \text{DELIMITED BY} \left\{ \begin{array}{c} \text{SIZE} \\ \text{Id-lit-2} \end{array} \right\} \right\} \dots \text{INTO ident-1}$$

[ WITH POINTER ident-2 ]  
 [ ON OVERFLOW statements-1 ]  
 [ NOT ON OVERFLOW statements-2 ]  
 [ END-STRING ]

从一个或多个发送数据项收集字符，并将其合成一个接收数据项

从 id-lit-1 开始，从左到右复制到 ident-1 中

DELIMITED BY id-lit-2，则复制进行到发送项中发现 id-lit-2 字符串为止

DELIMITED BY SIZE，则复制进行到发送项复制完毕或者 ident-1 填满为止

POINTER，用 ident-2 制定开始复制的具体字符位置。

Ident-2 应为数字型整数项，大于等于 1，小于等于 ident-1 的长度。语句结束时，ident-2 包含最后一个复制字符后面的一个位置。





Ident-2 小于 1 或者大于 ident-1 的长度，则语句溢出

ON OVERFLOW 指定溢出时的操作

NOT ON OVERFLOW 指定非溢出时的操作

#### 例 4.2.1

01 DayStr	PIC XX.	5												
01 MonthStr	PIC X(9).	J	U	N	E									
01 YearStr	PIC X(4).	2	0	0	4									
01 DateStr	PIC X(15) VALUE ALL "-".													
		-	-	-	-	-	-	-	-	-	-	-	-	-
STRING DayStr DELIMITED BY SPACES ", " DELIMITED BY SIZE MonthStr DELIMITED BY SPACES ", " DELIMITED BY SIZE YearStr DELIMITED BY SIZE INTO DateStr END-STRING.														

01 DayStr	PIC XX.	5												
01 MonthStr	PIC X(9).	J	U	N	E									
01 YearStr	PIC X(4).	2	0	0	4									
01 DateStr	PIC X(15) VALUE ALL "-".													
		5	-	-	-	-	-	-	-	-	-	-	-	-
STRING DayStr DELIMITED BY SPACES ", " DELIMITED BY SIZE MonthStr DELIMITED BY SPACES ", " DELIMITED BY SIZE YearStr DELIMITED BY SIZE INTO DateStr END-STRING.														



01 DayStr	PIC XX.	5												
01 MonthStr	PIC X(9).	J	U	N	E									
01 YearStr	PIC X(4).	2	0	0	4									
01 DateStr	PIC X(15) VALUE ALL "-".													
5	,	-	-	-	-	-	-	-	-	-	-	-	-	-
STRING DayStr DELIMITED BY SPACES ", " DELIMITED BY SIZE MonthStr DELIMITED BY SPACES ", " DELIMITED BY SIZE YearStr DELIMITED BY SIZE INTO DateStr END-STRING.														

01 DayStr	PIC XX.	5												
01 MonthStr	PIC X(9).	J	U	N	E									
01 YearStr	PIC X(4).	2	0	0	4									
01 DateStr	PIC X(15) VALUE ALL "-".													
5	,	J	U	N	E	-	-	-	-	-	-	-	-	-
STRING DayStr DELIMITED BY SPACES ", " DELIMITED BY SIZE MonthStr DELIMITED BY SPACES ", " DELIMITED BY SIZE YearStr DELIMITED BY SIZE INTO DateStr END-STRING.														



01 DayStr	PIC XX.	5												
01 MonthStr	PIC X(9).	J	U	N	E									
01 YearStr	PIC X(4).	2	0	0	4									
01 DateStr	PIC X(15) VALUE ALL "-".													
5, J U N E, 2 0 0 4 - - - -														
STRING DayStr DELIMITED BY SPACES ", " DELIMITED BY SIZE MonthStr DELIMITED BY SPACES ", " DELIMITED BY SIZE YearStr DELIMITED BY SIZE INTO DateStr END-STRING.														

例 4.2.2

01 StrPtr	PIC 99.													
01 DayStr	PIC XX.	5												
01 MonthStr	PIC X(9).	J	U	N	E									
01 YearStr	PIC X(4).	2	0	0	4									
01 DateStr	PIC X(15) VALUE ALL "-".													
- - - - - - - - - - - - - - -														
MOVE 1 TO StrPtr. STRING DayStr DELIMITED BY SPACES ", " DELIMITED BY SIZE INTO DateStr WITH POINTER StrPtr END-STRING. STRING MonthStr DELIMITED BY SPACES ", " DELIMITED BY SIZE INTO DateStr WITH POINTER StrPtr END-STRING. STRING YearStr DELIMITED BY SIZE INTO DateStr WITH POINTER StrPtr END-STRING.														



01 StrPtr PIC 99.

01 DayStr PIC XX.

01 MonthStr PIC X(9).

01 YearStr PIC X(4).

01 DateStr PIC X(15) VALUE ALL "-".

5														
J	U	N	E											
2	0	0	4											

5	,	-	-	-	-	-	-	-	-	-	-	-	-	-
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

MOVE 1 TO StrPtr.

**STRING DayStr DELIMITED BY SPACES**

**"," DELIMITED BY SIZE**

**INTO DateStr WITH POINTER StrPtr**

**END-STRING.**

**STRING MonthStr DELIMITED BY SPACES**

**"," DELIMITED BY SIZE**

**INTO DateStr WITH POINTER StrPtr**

**END-STRING.**

**STRING YearStr DELIMITED BY SIZE**

**INTO DateStr WITH POINTER StrPtr**

**END-STRING.**





01 StrPtr	PIC 99.														
01 DayStr	PIC XX.	5													
01 MonthStr	PIC X(9).	J	U	N	E										
01 YearStr	PIC X(4).	2	0	0	4										
01 DateStr	PIC X(15) VALUE ALL "-".														

01 DayStr	PIC XX.	5													
01 MonthStr	PIC X(9).	J	U	N	E										
01 YearStr	PIC X(4).	2	0	0	4										
01 DateStr	PIC X(15) VALUE ALL "-".														

5	,	J	U	N	E	,	2	0	0	4	-	-	-	-	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

STRING DayStr    DELIMITED BY SPACES  
                   ", "        DELIMITED BY SIZE  
                   MonthStr DELIMITED BY SPACES  
                   ", "        DELIMITED BY SIZE  
                   YearStr    DELIMITED BY SIZE  
                   INTO DateStr  
 END-STRING.

## 例 4.2.3

01 StringFields.
02 Field1    PIC X(18) VALUE "Where does this go".
02 Field2    PIC X(30)
VALUE "This is the destination string".
02 Field3    PIC X(15) VALUE "Here is another".
01 StrPointers.
02 StrPtr    PIC 99.
02 NewPtr    PIC 9.

**This is the destination string**

STRING Field1 DELIMITED BY SPACES  
                   INTO Field2  
 END-STRING.  
 DISPLAY Field2.



**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**Where is the destination string**

**STRING Field1 DELIMITED BY SPACES**

**INTO Field2**

**END-STRING.**

**DISPLAY Field2.**

**例 4.2.4**

**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**This is the destination string**

**STRING Field1 DELIMITED BY SIZE**

**INTO Field2**

**END-STRING.**

**DISPLAY Field2.**



**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**Where does this goation string**

**STRING Field1 DELIMITED BY SIZE**

**INTO Field2**

**END-STRING.**

**DISPLAY Field2.**

**例 4.2.5**

**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**This is the destination string**

**MOVE 6 TO StrPtr.**

**STRING Field1, Field3 DELIMITED BY SPACE**

**INTO Field2 WITH POINTER StrPtr**

**ON OVERFLOW DISPLAY "String Error"**

**NOT ON OVERFLOW DISPLAY Field2**

**END-STRING.**





**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**This WhereHere destination string**

**MOVE 6 TO StrPtr.**

**STRING Field1, Field3 DELIMITED BY SPACE**

**INTO Field2 WITH POINTER StrPtr**

**ON OVERFLOW DISPLAY "String Error"**

**NOT ON OVERFLOW DISPLAY Field2**

**END-STRING.**

**例 4.2.6**

**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**STRING Field1, Field2, Field3**

**DELIMITED BY SPACES**

**INTO Field4**

**END-STRING.**

**DISPLAY Field4.**



**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**WhereThisHere**

**STRING Field1, Field2, Field3**

**DELIMITED BY SPACES**

**INTO Field4**

**END-STRING.**

**DISPLAY Field4.**

**例 4.2.7**

**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**This is the destination string**

**MOVE 4 TO NewPtr.**

**STRING Field1 DELIMITED BY "this"**

**Field3 DELIMITED BY SPACE**

**"END" DELIMITED BY SIZE**

**INTO Field2**

**END-STRING.**



**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**Where does HereENDation string**

**MOVE 4 TO NewPtr.**

**STRING Field1 DELIMITED BY "this"**

**Field3 DELIMITED BY SPACE**

**"END" DELIMITED BY SIZE**

**INTO Field2**

**END-STRING.**

**例 4.2.8**

**01 StringFields.**

**02 Field1 PIC X(18) VALUE "Where does this go".**

**02 Field2 PIC X(30)**

**VALUE "This is the destination string".**

**02 Field3 PIC X(15) VALUE "Here is another".**

**01 StrPointers.**

**02 StrPtr PIC 99.**

**02 NewPtr PIC 9.**

**This is the destination string**

**MOVE 4 TO NewPtr.**

**STRING Field1 DELIMITED BY "this"**

**Field3 DELIMITED BY SPACE**

**"Tom" DELIMITED BY SIZE**

**INTO Field2 WITH POINTER NewPtr**

**ON OVERFLOW DISPLAY "String Error"**

**NOT ON OVERFLOW DISPLAY Field2**

**END-STRING.**

**01 StringFields.****02 Field1 PIC X(18) VALUE "Where does this go".****02 Field2 PIC X(30)****VALUE "This is the destination string".****02 Field3 PIC X(15) VALUE "Here is another".****01 StrPointers.****02 StrPtr PIC 99.****02 NewPtr PIC 9.****ThiWhere does Here is another****MOVE 4 TO NewPtr.****STRING Field1 DELIMITED BY "this"****Field3 DELIMITED BY SPACE****"Tom" DELIMITED BY SIZE****INTO Field2 WITH POINTER NewPtr****ON OVERFLOW DISPLAY "String Error"****NOT ON OVERFLOW DISPLAY Field2****END-STRING.**

### 4.3 UNSTRING

取一个发送数据项中的字符，并将起放入多个接收数据项中。其格式如下：

**UNSTRING SourceString****[ DELIMITED BY [ ALL ] Delim1 [ OR [ ALL ] Delim2 ]...****INTO {DestString [ DELIMITER IN Delimi3] [ COUNT IN Counter]} ...****[ WITH POINTER Pointer ]****[ TALLYING IN DestCounter ]****[ ON OVERFLOW statements1]****[ NOT ON OVERFLOW statements2 ]****[ END-UNSTRING ]**

UNSTRING 将发送数据项的字符拷贝到接收数据项中，直到碰到终止字符或者目的字符串满了。



这时，下一接收数据项开始接收数据，直到终止字符或着这个数据项也满了。

字符从发送数据项拷贝到接收数据项依据数字字母的移动规则进行。

**ON OVERFLOW:** 当下述情况发生时，产生溢出：

- 当 UNSTRING 执行时，指针没有在发送数据项的范围内
- 发送数据项中还有没有处理的字符，而所有的接收数据项都满了时。

**COUNT IN :** Counter 中保留了发送到接收数据项的字符数目。

**TALLYING IN:** 每个 UNSTRING 只能有一个 TALLYING 短语。Destcounter 中保留了被 UNSTRING 从句处理的接收数据项的数目。

**WITH POINTER:** Pointer 保留开始复制字符的具体位置。定义 Pointer 时，其大小至少要比发送数据项的长度大一。


**DELIMITER IN:** Delim3 中保留判断发送数据项终止的分隔符。

**ALL:** 如果使用了 ALL，则两个或多个连续出现的分隔符当作一个分隔符处理。

#### 例 4.3.1

01 DayStr PIC XX.								
01 MonthStr PIC XX.								
01 YearStr PIC XX.								
01 DateStr PIC X(8).	1	9	-	0	5	-	8	0

**ACCEPT DateStr.**  
**UNSTRING DateStr**  
**INTO DayStr, MonthStr, YearStr**  
**ON OVERFLOW DISPLAY "Chars Left"**  
**END-UNSTRING.**





01 DayStr PIC XX.  
01 MonthStr PIC XX.  
01 YearStr PIC XX.  
01 DateStr PIC X(8).

1	9								
1	9	-	0	5	-	8	0		

ACCEPT DateStr.  
**UNSTRING DateStr**  
**INTO DayStr, MonthStr, YearStr**  
**ON OVERFLOW DISPLAY "Chars Left"**  
END-UNSTRING.



01 DayStr PIC XX.  
01 MonthStr PIC XX.  
01 YearStr PIC XX.  
01 DateStr PIC X(8).

1	9								
-	0								
1	9	-	0	5	-	8	0		

ACCEPT DateStr.  
**UNSTRING DateStr**  
**INTO DayStr, MonthStr, YearStr**  
**ON OVERFLOW DISPLAY "Chars Left"**  
END-UNSTRING.



01 DayStr PIC XX.  
01 MonthStr PIC XX.  
01 YearStr PIC XX.  
01 DateStr PIC X(8).

1	9								
-	0								
5	-								
1	9	-	0	5	-	8	0		


ACCEPT DateStr.  
**UNSTRING DateStr**  
**INTO DayStr, MonthStr, YearStr**  
**ON OVERFLOW DISPLAY "Chars Left"**  
END-UNSTRING.





01 DayStr	PIC XX.	1	9									
01 MonthStr	PIC XX.	-	0									
01 YearStr	PIC XX.	5	-									
01 DateStr	PIC X(8).	1	9	-	0	5	-	8	0			


ACCEPT DateStr.  
UNSTRING DateStr  
    INTO DayStr, MonthStr, YearStr  
    ON OVERFLOW DISPLAY "Chars Left"  
END-UNSTRING.



例 4.3.2

01 DayStr	PIC XX.	1	9									
01 MonthStr	PIC XX.											
01 YearStr	PIC XX.											
01 DateStr	PIC X(8).	1	9	-	0	5	/	8	0			

ACCEPT DateStr.  
UNSTRING DateStr  
    DELIMITED BY "/" OR "-"  
    INTO DayStr DELIMITER IN Hold1  
        MonthStr DELIMITER IN Hold2  
        YearStr  
END-UNSTRING.  
DISPLAY DayStr SPACE MonthStr SPACE YearStr.  
DISPLAY Hold1 SPACE Hold2





01 DayStr PIC XX.  
01 MonthStr PIC XX.  
01 YearStr PIC XX.  
01 DateStr PIC X(8).

1	9							
0	5							
1	9	-	0	5	/	8	0	

ACCEPT DateStr.  
**UNSTRING DateStr**  
**DELIMITED BY "/" OR "-"**  
**INTO DayStr DELIMITER IN Hold1**  
**MonthStr DELIMITER IN Hold2**  
**YearStr**  
END-UNSTRING.  
DISPLAY DayStr SPACE MonthStr SPACE YearStr.  
DISPLAY Hold1 SPACE Hold2



01 DayStr PIC XX.  
01 MonthStr PIC XX.  
01 YearStr PIC XX.  
01 DateStr PIC X(8).

1	9							
0	5							
8	0							
1	9	-	0	5	/	8	0	

ACCEPT DateStr.  
**UNSTRING DateStr**  
**DELIMITED BY "/" OR "-"**  
**INTO DayStr DELIMITER IN Hold1**  
**MonthStr DELIMITER IN Hold2**  
**YearStr**  
**END-UNSTRING.**  
DISPLAY DayStr SPACE MonthStr SPACE YearStr.  
DISPLAY Hold1 SPACE Hold2









01 DayStr PIC XX.

01 MonthStr PIC XX.

01 YearStr PIC XX.

01 DateStr PIC X(8).

1	5										
0	7										
0	4										
1	5	-	-	-	0	7	-	-	0	4	

ACCEPT DateStr.

UNSTRING DateStr

DELIMITED BY ALL "-"

INTO DayStr, MonthStr, YearStr

ON OVERFLOW DISPLAY "Chars Left"

END-UNSTRING.



例 4.3.4

01 DayStr PIC XX.

01 MonthStr PIC XX.

01 YearStr PIC XX.

01 DateStr PIC X(8).

1	5	-	-	-	0	7	-	-	0	4	

ACCEPT DateStr.

UNSTRING DateStr

DELIMITED BY "-"

INTO DayStr, MonthStr, YearStr

ON OVERFLOW DISPLAY "Chars Left"

END-UNSTRING.



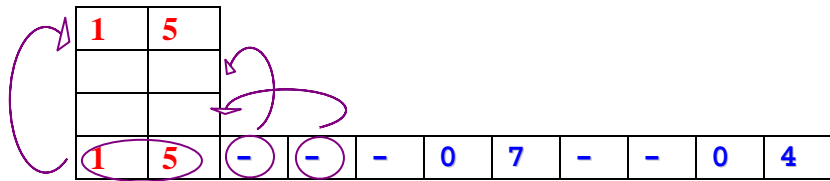


01 DayStr PIC XX.

01 MonthStr PIC XX.

01 YearStr PIC XX.

01 DateStr PIC X(8).



ACCEPT DateStr.

UNSTRING DateStr

DELIMITED BY "-."

INTO DayStr, MonthStr, YearStr

ON OVERFLOW DISPLAY "Chars Left"

END-UNSTRING.

Chars Left

## 5 过程分支命令

### 5.1 GO TO

将控制转入过程部的另一位值，指定下一个要执行的过程的名字，从而无条件的改变语句执行顺序。该过程可以位于程序的任何部分。

有两种形式

```
GO TO [ procedure-name ]
```

```
GO TO { procedure-name } ...DEPENDING ON ident
```

#### GO TO

无条件的将控制转入指定过程，常用于程序循环中，此循环将继续执行，直到到达一个出口条件为止。

#### GO TO DEPENDING



依据标识符的值将控制转入一列于标识符的值相应的过程名。

Ident 为整型数字数据对象，取值在 1-n 之间

若数值为 1，则控制转入清单中第一个 procedure-name；若数值为 2，则控制转入清单中第二个 procedure-name，以此类推。

若数值在 1-n 之外，则控制不转移，直接执行下一条语句。n 不能超过 2031。

## 5.2 PERFORM 语句

PERFORM 命令中断程序的正常执行序列，使之转移到其他节段，功能类似于 GO TO 语句，但是有不同点。GO TO 语句实现的转移是一去不回的，而 PERFORM 命令转出去执行其他节段后，在正常情况下将要返回原处继续执行。

通过 PERFORM 语句调用的程序段可以根据需要或者根据某种情况，重复执行。

循环执行是程序的重要组成部分。当需要重复操作相同的指令时，我们使用循环方式。

大多数的语言都有多种的循环语句，例如 WHILE, FOR, REPEAT 等。我们可以根据不同的情形采用不同的循环语句。

COBOL 只有一种循环语句： PERFORM。

但是 PERFORM 语句有不同的变化形式。

每种变化形式就相当于别的语言的不同的循环语句。

PERFORM 语句有四种形式

- PERFORM
- PERFORM TIMES
- PERFORM UNTIL
- PERFORM VARYING

### 5.2.1 PERFORM

这是最基本的 PERFORM 语句，格式如下：



PERFORM procedure-name-1  
[ { THROUGH } Procedure-name-2 ]  
  THRU ]

控制传递到第一条语句，并在完成后一个过程的最后一条语句后返回。

这是唯一一种不循环执行的语句形式

procedure-name-1 和 procedure-name-2 是要执行的段或节的名称。

在这条语句中，从 procedure-name-1 到 procedure-name-2 之间的所有过程作为一大段程序运行。

运行后，控制转移到直接跟在 PERFORM 语句后的语句。

例 1:

**PROCEDURE DIVISION.**

**TopLevel.**

**DISPLAY “ Starting to run program.”.**

**PERFORM OneLevel.**

**DISPLAY "Back in TopLevel.”.**

**STOP RUN.**

**TwoLevel.**

**DISPLAY “----- Now in TwoLevel. ”.**

**OneLevel.**

**DISPLAY “----- Now in OneLevel. ”.**

**PERFORM TwoLevel.**

**DISPLAY “----- Back in OneLevel.”.**

运行后的结果如下：



```
Starting to run program.  
Now in OneLevel.  
----- Now in TwoLevel.  
----- Back in OneLevelDown.  
Back in TopLevel.
```

程序运行的顺序如下：

```
PROCEDURE DIVISION.  
TopLevel.  
1      DISPLAY " Starting to run program."  
2      PERFORM OneLevel.  
7      DISPLAY "Back in TopLevel."  
      STOP RUN.  
  
4      TwoLevel.  
5      DISPLAY "----- Now in TwoLevel. "  
  
2      OneLevel.  
3      DISPLAY "----- Now in OneLevel. "  
4      PERFORM TwoLevel.  
6      DISPLAY "----- Back in OneLevel."
```

### 5.2.2 PERFORM TIMES

格式如下::



```
PERFORM procedure-name-1  
[ { THROUGH } Procedure-name-2 ]  
  THRU ]  
Counter TIMES
```

类似 PERFORM 语句，但不是执行一次

指定执行的次数 counter

counter 为整数直接数或者数字数据项

counter 小于 0，则指定的过程不执行

在 PERFORM 语句执行已经开始之后和在它结束之前，若 counter 的值有了变动，并不会改变该指定的过程要执行的次数。

例 2:

```
PROCEDURE DIVISION.  
Begin.  
  Statements  
  PERFORM  DisplayName 4 TIMES  
  Statements  
  STOP RUN.  
  
  DisplayName.  
    DISPLAY "Tom Ryan".
```

其结果如下：



**Tom Ryan**

**Tom Ryan**

**Tom Ryan**

**Tom Ryan**

例 3:

**IDENTIFICATION DIVISION.**

**PROGRAM-ID.** PerformExample2.

**AUTHOR.** Lilly.

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**01 NumofTimes**      **PIC 9 VALUE 5.**

**PROCEDURE DIVISION.**

**Begin.**

**DISPLAY "Starting to run program"**

**PERFORM 3 TIMES**

**DISPLAY ">>>>This is an in line Perform"**

**END-PERFORM**

**DISPLAY "Finished in line Perform"**

**PERFORM OutOfLineEG NumOfTimes TIMES**

**DISPLAY "Back in Begin. About to Stop".**

**STOP RUN.**

**OutOfLineEG.**

**DISPLAY ">>>> This is an Paragraph Perform".**

其结果如下:





```
Starting to run program
>>>>This is an in line Perform
>>>>This is an in line Perform
>>>>This is an in line Perform
Finished in line Perform
>>>> This is an Paragraph Perform
>>>> This is an Paragraph Perform
>>>> This is an Paragraph Perform
>>>> This is an Paragraph Perform
>>>> This is an Paragraph Perform
Back in Begin. About to Stop
```

### 5.2.3 PERFORM UNTIL

其格式如下：

```
PERFORM procedure-name-1
[ { THROUGH } Procedure-name-2 ]
  THRU
[ WITH TEST { BEFORE } ]
               AFTER
UNTIL condition
```

重复执行指定的过程组，直到条件变为 TRUE。如果在 PERFORM 开始之前，条件为真，则过程不执行，控制转移到 PERFORM 后面的语句。

WITH TEST 选择在第一次执行前或执行后求值 condition，默认为 BEFORE。

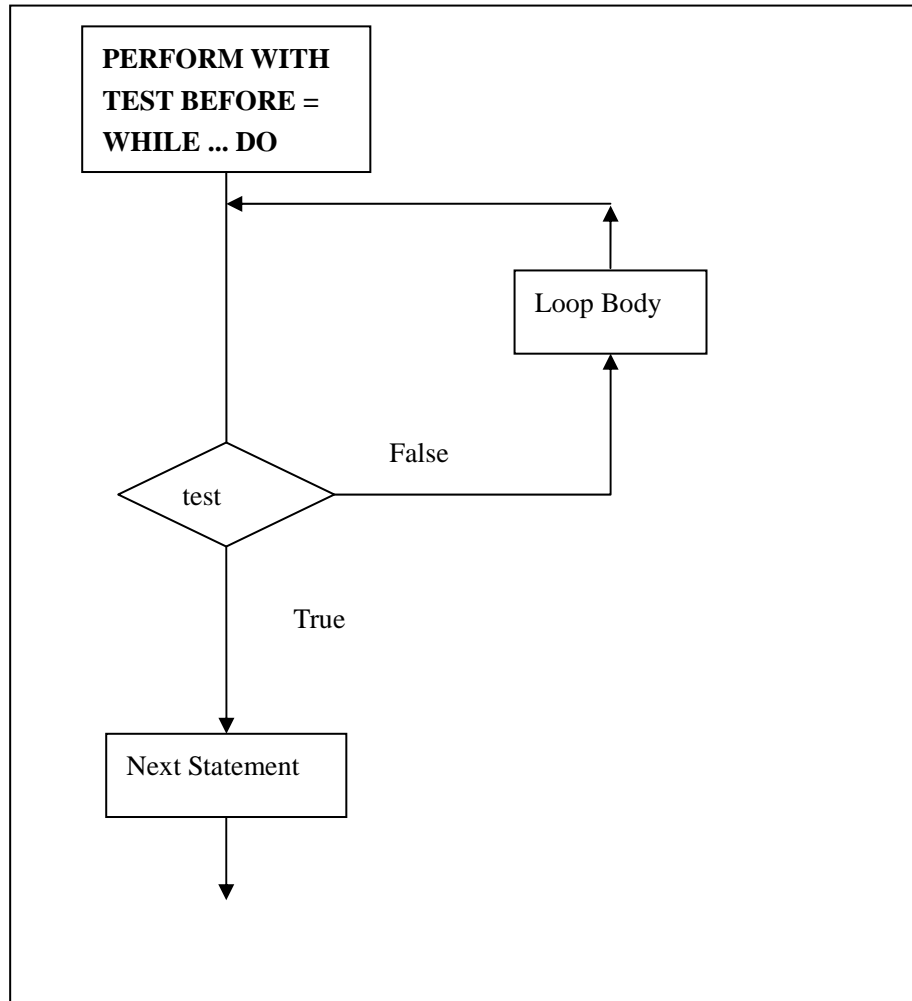
每次执行后，求值 condition，如果是 TRUE 执行终止。

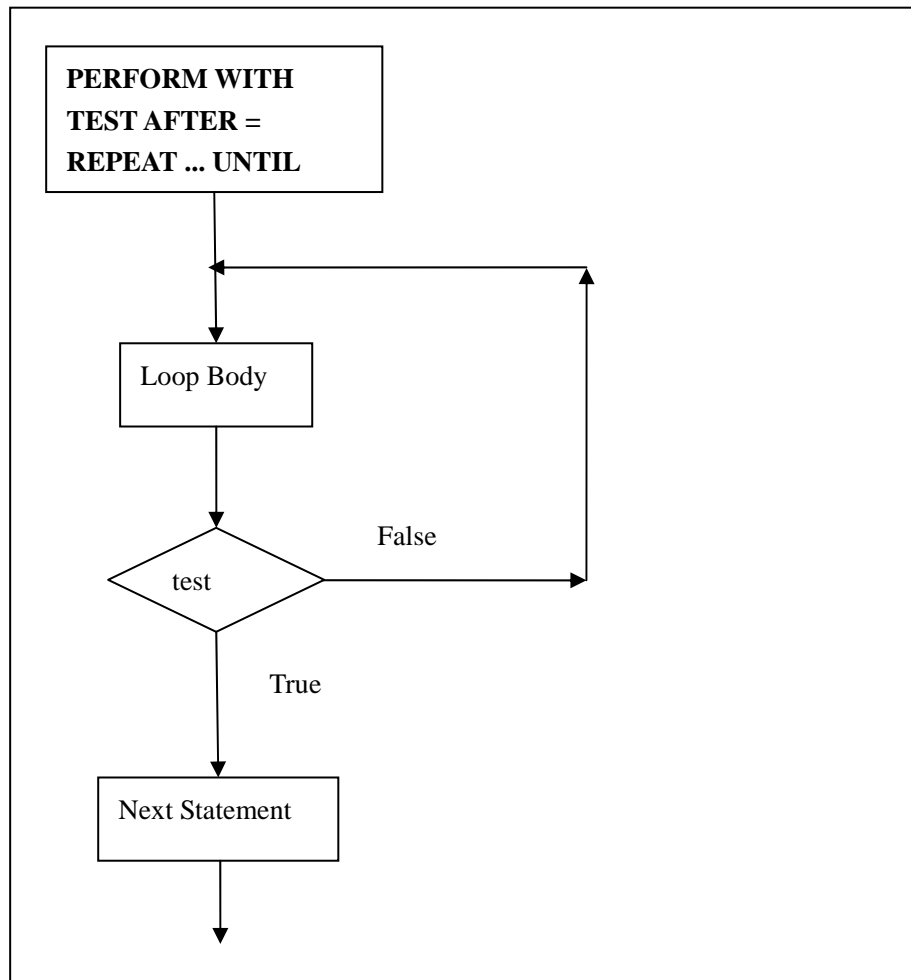
其功能相当于其他语言中的 while 或者 repeat 语句。

WITH TEST BEFORE = WHILE



WITH TEST AFTER = REPEAT





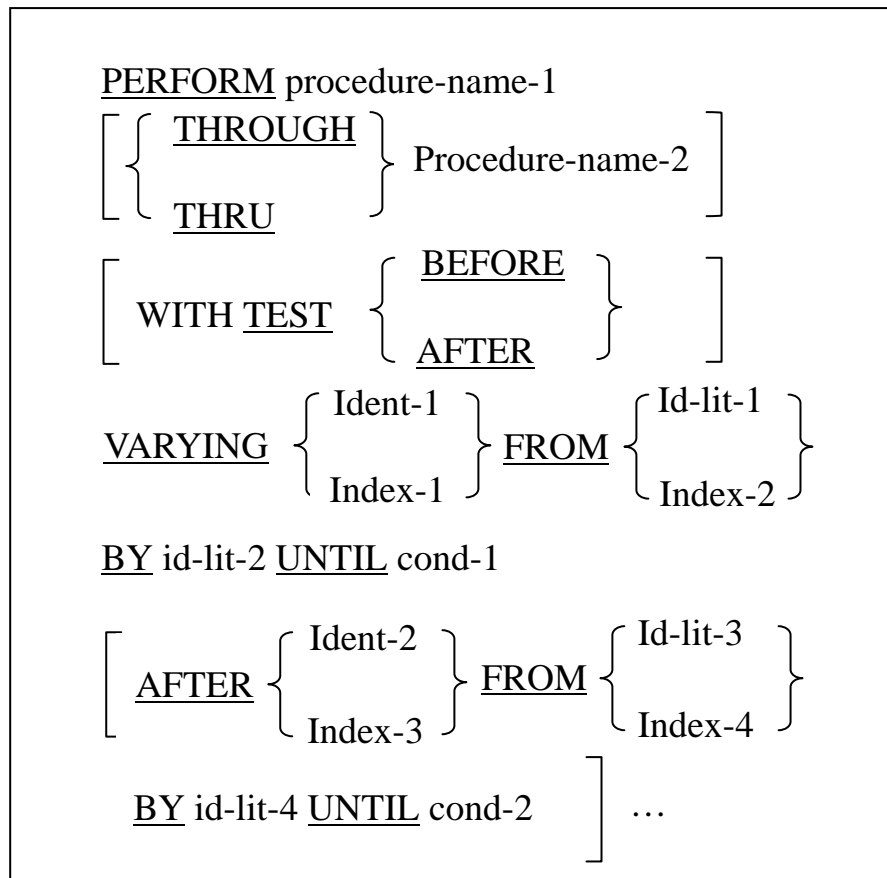
例 4:

```
READ StudentRecords
AT END MOVE HIGH-VALUES TO StudentRecord
END-READ
PERFORM UNTIL StudentRecord = HIGH-VALUES      DISPLAY
StudentRecord
    READ StudentRecords
    AT END MOVE HIGH-VALUES TO StudentRecord
    END-READ
END-PERFORM
```

## 5.2.4 PERFORM VARYING



格式如下：



第四种格式与第三种格式的区别在于条件的变化过程由命令显式给出，即每执行一次过程段，就使条件参量（数据名或者指标名）从初值开始（数据名，指标名或者常量）改变一个增量（数据名或者常量），并测试条件是否满足。

所有 **Ident** 应该为数字项目或索引名。

所有 **id-lit** 应该为数字。

**WITH TEST** 用法同 **PERFORM UNTIL**。

**AFTER** 用来指定第二（或更多）组要改变的标识符和终止内循环的对应条件。

由于 **PERFORM** 命令在实现转移后要返回，因此，如果 **PERFORM** 命令中给出的过程段直接位于 **PERFORM** 命令之后，则这个过程段的实际执行次数为 **PERFORM** 命令中规定的执行次数再加 1。

例 5：



**PERFORM 0010-PROCESS-PROCEDURE  
WITH TEST AFTER  
VARYING IX FROM 1 BY 1 UNTIL IX=3  
AFTER JX FROM 1 BY 1 UNTIL JX=2.**

结果如下：

<b>IX</b>	<b>JX</b>
<b>1</b>	<b>1</b>
<b>1</b>	<b>2</b>
<b>2</b>	<b>1</b>
<b>2</b>	<b>2</b>
<b>3</b>	<b>1</b>

### 5.3 CONTINUE

只是语句占位符

### 5.4 EXIT

格式极为简单：

**EXIT.**

在由 **PERFORM** 命令调用的过程段中，提供一个公共的返回点。

占位符，不起任何作用。



作用在段中。

## 5.5 STOP

此命令是用来使程序停止执行，或者使程序暂时挂起。

格式如下：

$\text{STOP} \left\{ \begin{array}{l} \text{RUN} \\ \text{Literal1} \end{array} \right\}$
---

### STOP RUN

结束程序，关闭所有打开的文件。最好在结束前关闭程序，不要用这条语句关闭。

每个程序必须有一个 **STOP RUN** 命令，作为程序运行结束的标志，但这个命令不一定在程序的最后，也不一定是一个独立的命令，可以出现在条件语句中。但是如果此命令用于一个命令语句序列，例如在 **IF** 语句中，当满足执行条件时，需要执行一系列命令语句，则此语句必须是最后一个语句，否则，其后语句未执行就停止了。

在子程序中不能使用 **STOP RUN**，而要用 **EXIT PROGRAM** 命令结束子程序返回主程序。如果使用了 **STOP RUN**，则子程序会返回操作系统。

### STOP literal1

使程序暂时挂起，通常将 **STOP** 后的直接量显示在控制台屏幕上，对操作员作必要的提示。然后由操作员采取行动（例如按回车键）恢复程序的执行。



## 第七章 算术语法

- 命令的基本语法
- 学会使用这些语法



本章主要介绍 COBOL 的运算语句。

在数据处理中，常常需要进行数据的计算处理，COBOL 语言是数据处理的专用计算机语言。数据处理的一个特点是一般不涉及复杂的运算，因此 COBOL 只有一些简单的运算功能功能，即只有加、减、乘、除运算，以及可以表示较为复杂的 COMPUTE 语句。

## 1. ADD

将两个或多个数字相加，将结果存放在数字型或数字编辑型字段中。

格式有以下三种：

- ADD TO
- ADD GIVING
- ADD CORRESPONDING

### 1.1. ADD TO

直接将一个或多个数据项与一个或多个数据项相加

格式如下：

```
ADD { num1} ... TO { ident [ ROUNDED ]} ...  
[ON SIZE ERROR statements1]  
[NOT ON SIZE ERROR statements]  
[END-ADD]
```

#### 例 1.1.1

```
ADD          NUM1    TO    NUM2.
```





BEFORE	10	25
AFTER	10	35

例 1.1.2

ADD	NUM1,15 TO NUM2, NUM3.		
BEFORE	10	25	11
AFTER	10	50	36

## 1.2. ADD GIVING

将一个或多个数据项与一个或多个数据项相加将结果放入另一项目中。

格式如下：

```

ADD { num1} ... TO  num2 GIVING { ident [ ROUNDED ]} ...
[ON SIZE ERROR statements1]
[NOT ON SIZE ERROR statements]
[END-ADD]

```

例 1.2.1

ADD	NUM1, NUM2 GIVING NUM3.		
BEFORE	10	25	11
AFTER	10	25	35



ADD TO 和 GIVING 不能同时出现。

### 1.3. ADD CORRESPONDING

加法命令的缩写形式，格式如下：

```
ADD  CORRESPONDING  num1 TO { ident } ...  
[ON SIZE ERROR statements1]  
[NOT ON SIZE ERROR statements]  
[END-ADD]
```

## 2. SUBTRACT

从一个数字项中减去一个或几个数据项的和。

格式有以下三种：

- **SUBTRACT FROM**
- **SUBTRACT GIVING**
- **SUBTRACT CORRESPONDING**

### 2.1. SUBTRACT FROM

直接从一个或多个数据项中减去与一个或多个数据项

格式如下：



```
SUBTRACT { num1} ... FROM { ident [ ROUNDED ]} ...  
  
[ON SIZE ERROR statements1]  
  
[NOT ON SIZE ERROR statements]  
  
[END-SUBTRACT]
```

例 2.1.1

SUBTRACT                  NUM1 FROM NUM2.		
BEFORE	10	25
AFTER	10	15

例 2.1.2

SUBTRACT NUM1,15 FROM NUM2, NUM3.			
BEFORE	10	55	100
AFTER	10	30	75

## 2.2. SUBTRACT GIVING

从一个或多个数据项中减去一个或多个数据项，将结果放入另一项目中。

格式如下：



**SUBTRACT { num1} ... FROM num2 GIVING { ident  
[ ROUNDED ]} ...**

**[ON SIZE ERROR statements1]**

**[NOT ON SIZE ERROR statements]**

**[END-SUBTRACT]**

#### 例 2.2.1

SUBTRACT NUM1 FROM NUM2 GIVING NUM3.			
BEFORE	10	25	11
AFTER	10	25	15

### 2.3. SUBTRACT CORRESPONDING

减法命令的缩写形式，格式如下：

**SUBTRACT CORRESPONDING num1 FROM { ident } ...**

**[ON SIZE ERROR statements1]**

**[NOT ON SIZE ERROR statements]**

**[END-ADD]**

### 3. MULTIPLY



将两个数字型数据项相乘，将结果存放在数字型或数字编辑型字段中。

格式有以下两种：

- MULTIPLY BY
- MULTIPLY GIVING

### 3.1. MULTIPLY BY

直接将一个数据项与一个数据项相乘

格式如下：

```
MULTIPLY  num1 BY { ident [ ROUNDED ]} ...  
  
[ON SIZE ERROR statements1]  
  
[NOT ON SIZE ERROR statements]  
  
[END-MULTIPLY]
```

#### 例 3.1.1

MULTIPLY	NUM1 BY	NUM2.
BEFORE	10	12
AFTER	10	120

### 3.2. MULTIPLY GIVING

一个数据项与另一个数据项相乘，将结果放入另一项目中。



格式如下：

```
MULTIPLY  num1BY  num2 GIVING { ident [ ROUNDED ]} ...  
[ON SIZE ERROR statements1]  
[NOT ON SIZE ERROR statements]  
[END-SUBTRACT]
```

### 例 3.2.1

MULTIPLY	NUM1 BY	NUM2 GIVING	NUM3.
BEFORE	10	12	11
AFTER	10	12	120

## 4. DIVIDE

将一个数字数据项（被除数）除以另一数字数据项（除数），并将结果存放在数字或数字编辑型数据项中。

有五种形式

- DIVIDE INTO
- DIVIDE INTO GIVING
- DIVIDE BY GIVING
- DIVIDE INTO EMAINDER
- DIVIDE BY ... REMAINDER

这五种格式的除法命令主要区别在以下三个方面



1. 用的介词是 INTO 还是 BY。用 INTO，其前为除数，其后是被除数；用 BY 的，其前是被除数，其后是除数。两种位置正好相反。
2. 有无 GIVING 短语，有 GIVING 短语的，专门指定一个数据项用来接收“商”，没有 GIVING 短语的，使用原先的被除数接收“商”
3. 有无 REMAINDER 短语，有 REMAINDER 短语的，由其后的数据项接收，保存余数，没有 REMAINDER 短语的，余数不保留。

#### 4.1. DIVIDE INTO

格式如下：

```
DIVIDE  num1 INTO { ident [ ROUNDED ]} ...  
  
[ON SIZE ERROR statements1]  
  
[NOT ON SIZE ERROR statements]  
  
[END-DIVIDE]
```

直接将一个数据项除以另一数据项。用 ident 除以 num1 的值，结果放在 ident

##### 例 4.1.1

DIVIDE	NUM1	INTO	NUM2.
BEFORE	10		120
AFTER	10		12
BEFORE	11		121
AFTER	11		11



## 4.2. DIVIDE INTO GIVING

格式如下：

```
DIVIDE  num1 INTO  num2 GIVING { ident [ ROUNDED ]} ...  
  
[ON SIZE ERROR statements1]  
  
[NOT ON SIZE ERROR statements]  
  
[END-DIVIDE]
```

将一个数据项除以另一数据项,结果放到第三项中。用 num2 的值除以 num1，结果放在 ident

### 例 4.2.1

DIVIDE	NUM1	INTO	NUM2	GIVING	NUM3.
BEFORE	10		120		6
AFTER	10		120		12

## 4.3. DIVIDE BY GIVING

格式如下：





```
DIVIDE  num1 BY  num2 GIVING { ident [ ROUNDED ]} ...  
[ON SIZE ERROR statements1]  
[NOT ON SIZE ERROR statements]  
[END-DIVIDE]
```

将一个数据项除以另一数据项,结果放到第三项中。用 num1 的值除以 num2, 结果放在 ident。

#### 例 4.3.1

DIVIDE	NUM1	BY	NUM2	GIVING	NUM3.
BEFORE	35		5		9
AFTER	35		5		7

#### 4.4. DIVIDE INTO ...REMAINDER

格式如下：

```
DIVIDE  num1 INTO  num2 GIVING  ident1 [ ROUNDED ]  
REMAINDER ident2  
[ON SIZE ERROR statements1]  
[NOT ON SIZE ERROR statements]  
[END-DIVIDE]
```

将一个数据项除以另一数据项,商和余数放到其他数据项中。用 num2 的值除以



num1，商放在 ident1，余数放到 ident2。

例 4.4.1

DIVIDE	num1 INTO	num2 GIVING	num3 REMAINDER	num4
BEFORE	9	110	100	1
AFTER	9	110	12	2

## 4.5. DIVIDE BY ...REMAINDER

格式如下：

```
DIVIDE  num1 BY  num2 GIVING  ident1 [ ROUNDED ]  
REMAINDER ident2  
  
[ON SIZE ERROR statements1]  
  
[NOT ON SIZE ERROR statements]  
  
[END-DIVIDE]
```

将一个数据项除以另一数据项,商和余数放到其他数据项中。用 num1 的值除以 num2，商放在 ident1，余数放到 ident2。

例 4.5.1

DIVIDE	num1 BY	num2 GIVING	num3 REMAINDER	num4
BEFORE	135	7	34	7



AFTER	135	7	19	2
-------	-----	---	----	---

## 5. COMPUTE

求值一个算术表达式，并将结果存放在一个或多个标识符中。可以用来求方次，但仅限于整数次方。

可以将各种算术运算合并一条语句中。

没有从除法操作中捕捉余数的算术表达式。

COMPUTE 的运算速度较慢，对简单的计算不宜采用。

此表达式相当于数学中的算式，有数值型数据项，数值型常量，运算符号，以及与圆括号组成，表示一定的运算意义，使用的符号如下：

1.   \*\*   =   POWER   NN
2.   \*    =   MULTIPLY   x  
   /   =   DIVIDE   ÷
3.   +    =   ADD        +  
   -    =   SUBTRACT   -

### 例 5.1

```
01 WK-X                PIC 9(4).
01 WK-19XACCT          PIC X(18).
01 WK-199ACCT.
    05 WK-19-1          PIC 9(1).
    05 WK-19-2          PIC 9(1).
    05 WK-19-3          PIC 9(1).
    05 WK-19-4          PIC 9(1).
COMPUTE WK-X = ( WK-19-1 + WK-19-3) * 2 - 9 * WK-JYW-COUNT +
( WK-19-2 + WK-19-4).
```

## 6. ROUNDED



四舍五入处理，将运算产生的数值先进行舍入处理再存入接收项目中。

其原数据与运算后的结果如下例

#### 例 1.6.1

Field Type	Actual Result	Rounded Result
PIC 9(3)V99	125.256	125.26
PIC 999	125.25	125

使用时，注意以下两点

4. ROUNDED 子句可以用在任何运算语句中，放在存放结果数据项之后
5. ROUNDED 子句只在发生小数被截断时发生作用

## 7. SIZE ERROR

算术操作的结果绝对值大于接收项目的最大存储能力。如果计算的结果整数位大于接收数据项的整数位数，则高位多出的数据被截断，这种情况，就叫长度溢出。

长度溢出会产生错误的运算结果，而计算机在执行程序时，对方生的长度溢出并不理会，继续执行程序，在我们分析程序产生的结果时才会发现。而为了发现这种问题常常会花费很长的时间。

为了避免这种情况的发生，我们使用 ON SIZE ERROR 语句来检测是否会产生长度溢出。如果产生了溢出，则转入执行 ON SIZE ERROR 之后给出的语句，如果不发生溢出，则不理睬 ON SIZE ERROR 之后给出的语句

除数为 0 时也会发生溢出错误。

数据发生溢出的情况，如下例：

#### 例 7.1

Receiving Field	Actual Result	SIZE ERROR
PIC 9(3)V9.	245.96	YES
PIC 9(3)V9.	1234.5	YES
PIC 9(3).	123	NO
PIC 9(3).	1234	YES



PIC 9(3)V9 Not Rounded	123.45	YES
PIC 9(3)V9 Rounded	123.45	NO
PIC 9(3)V9 Rounded	1234.45	YES

ON SIZE ERROR 后面可以跟一个或多个语句，当发生溢出时，由第一条语句开始执行，直到遇到句号为止，因此，要特别注意句号的位置。

如果在编程的时候，在定义数据时考虑的比较周全，仅在很少的情况下需要使用此子句。

## 8. CORRESPONDING

在数据项名相同时编写多条语句的缩写方法。

### 例 8.1

**01 GROUP-1.**

**05 SUB-1.**

**10 COUNTER1 PIC 9(3).**

**10 COUNTER2 PIC 9(3).**

**05 SUB-2.**

**10 COUNTER1 PIC 9(3).**

**10 COUNTER2 PIC 9(3).**

**ADD CORRESPONDING SUB-1 TO SUB-2.**

**||**

**ADD COUNTER1 OF SUB-1 TO COUNTER1 OF SUB-2.**

**ADD COUNTER1 OF SUB-1 TO COUNTER1 OF SUB-2.**



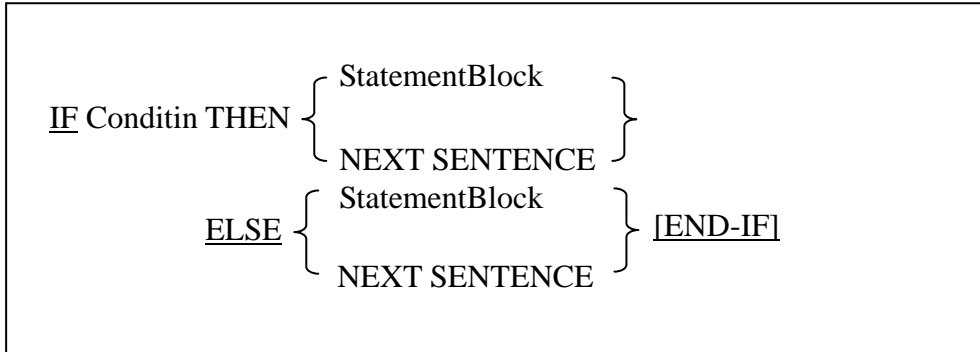
## 第八章 逻辑与控制语法

- 命令的基本语法
- 学会使用这些语法



## 1 IF THEN

基本格式如下：



IF 语句求一个条件表达式的逻辑值，并根据求值的结果决定语句的执行。

在 IF 语句的末尾必须有一个句号，否则，在 IF 以后直到下一个句号之前的每个字都会被当作 IF 语句的一部分。

条件的类型有

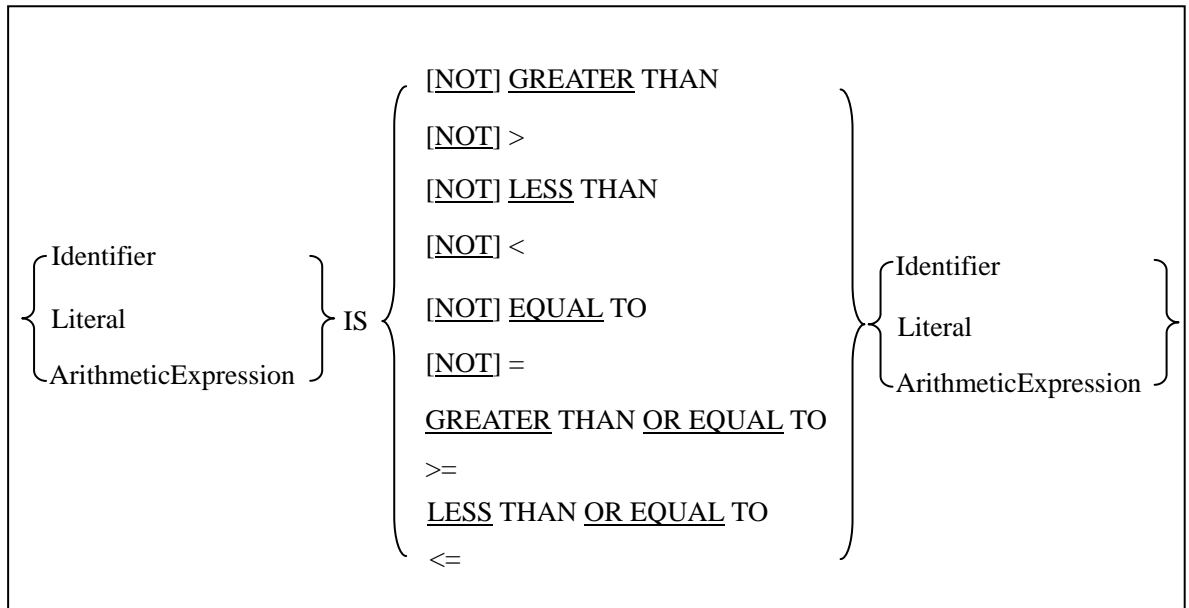
➤ 简单 Conditions

- 关系型条件
- 类条件
- 符号条件

➤ 复合 Conditions

### 1.1 关系型条件

格式如下：

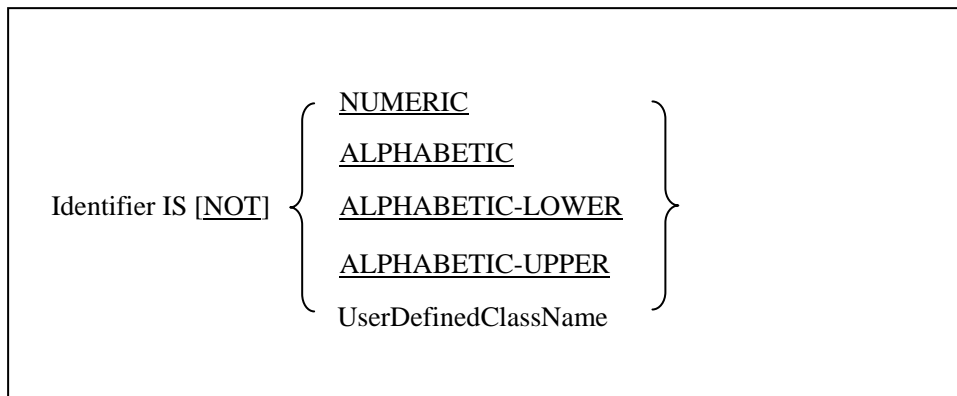


当比较数值项时，需要遵守以下规则

1. 把主语和宾语中的小数点位置对齐
2. 高位数和低位数位置上均填写零使两个字段在长度上面相等
3. 比较主语和宾语的代数值

## 1.2 类条件

格式如下：







使用方法如下：

例 1.2.1

```
ENVIRONMENT DIVISION.  
SPECIAL-NAMES.  
    CLASS REAL-NUMBER IS "0123456789+ -."  
DATA DIVISION.  
    01 BAL PIC X(9).  
    01 NAME PIC X(15).  
    01 FEES PIC X(20).  
  
IF BAL IS NOT NUMERIC THEN ...  
IF NAME IS ALPHABETIC-UPPER THEN...  
IF FEES IS REAL-NUMBER THEN...
```

### 1.3 符号条件

格式如下：

ArithExp IS [NOT] { POSITIVE  
                          NEGATIVE  
                          ZERO }

符号条件测试算术表达式的值大于、小于或等于 0

POSITIVE —— 大于 0

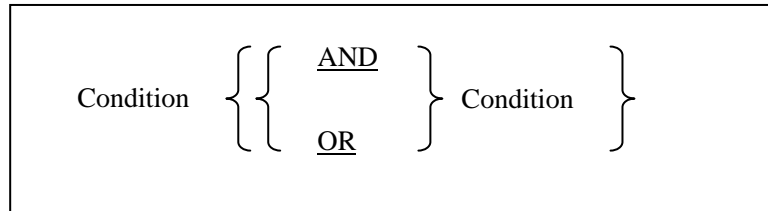
NEGATIVE —— 小于 0

ZERO —— 等于 0



## 1.4 复合Condition

格式如下：



程序经常需要各种组合情况的判断以决定下面的执行步骤。

与别种语言一样 COBOL 允许用 OR 、 AND 和括号组合简单的条件。

复合条件的判断也是要依据结果的 TRUE 和 FALSE。

复合条件实际是根据优先级从左到右顺序的表达式。

这种条件包括两个或者更多个利用运算符 AND, OR, NOT 组合在一起的简单条件。

### 1.4.1 逻辑操作符的优先级

优先级顺序

1. NOT = \*\*
2. AND = \* or /
3. OR = + or -

假设 A、B、C 为简单条件

$$A \text{ OR } \text{NOT } B \text{ AND } C = A \text{ OR } ((\text{NOT } B) \text{ AND } C)$$

$$A \text{ AND } B \text{ AND } C = (A \text{ AND } B) \text{ AND } C$$



## 1.5 缩写

```
IF A > 1000 AND A < 5000 THEN  
  
IF B = 5 OR B = 6 OR B = 7 THEN  
  
IF A > B AND A > C AND A > D THEN
```

||

```
IF A > 1000 AND < 5000 THEN  
  
IF B = 5 OR 6 OR 7 THEN  
  
IF A > B AND C AND D THEN
```

## 1.6 条件名

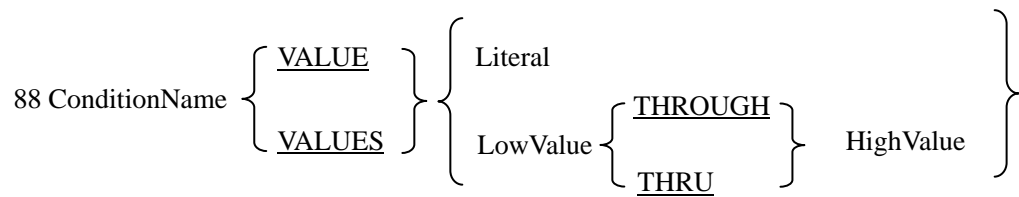
条件名条件测试条件变量是否定义了数值

条件名或条件变量在数据部定义层号 88

```
01 SW-NCRCRF-EOF PIC X          VALUE 'F'.  
      88 SW-END-NCRCRF          VALUE 'T'.  
  
READ NCRCRF NEXT RECORD  
  AT END  
    MOVE 'T' TO SW-NCRCRF-EOF  
  NOT AT END  
    MOVE 'F' TO SW-NCRCRF-EOF  
END-READ.  
IF NOT SW-END-NCRCRF THEN ...
```

## 1.7 数据定义88层的定义

定义 88 层数据的格式如下：



使用方法如下例

### 例 1.7.1

```
01 MONTH PIC XXX.

88          FIRST-KIND          VALUES          ARE
"JAN","MAR","MAY","JUY","AUG","OCT","DEC".

88 SEC-KIND VALUES ARE "APR","JUN","SEP","NOV".

88 TRD-KIND VALUES "FEB".

IF FIRST-KIND THEN Do something.
```

### 例 1.7.2



```

01 InputChar    PIC X.

88 Vowel    VALUE  "A","E","I","O","U".

88 Consonant    VALUE  "B" THRU "D", "F","G","H"
                  "J" THRU "N", "P" THRU "T"
                  "V" THRU "Z".

88 Digit    VALUE  "0" THRU "9".

88 LowerCase    VALUE  "a" THRU "z".

88 ValidChar    VALUE  "A" THRU "Z","0" THRU "9".

```

## 1.8 EVALUATE

其基本格式如下：

```

EVALUATE expression1 [ ALSO express2 ] ...

{ { WHEN { ANY
           condition1
           [ NOT ] expression3 [ THROUGH expression4 ] } }

[ ALSO { ANY
         condition2
         [ NOT ] expression5 [ THROUGH expression6 ] } ] ... } ...

statement1 } ...

[ WHEN OTHER statement2 ]

[ END-EVALUATE ]

```

只有一个主语表达式时，求值 expression1



数值有 3 种

- 数字值
- 字母数字值
- 真假值

比较主语值与 **WHEN** 短语

找到匹配，执行所选短语

### **1.8.1 选择WHEN短语**

如果是 **ANY**，则不管主语的类型和数值如何，均选择此短语

若是条件表达式和保留字 **TRUE** 和 **FALSE**，则选择与主语真假值相同的

如果是第三种，则比较主语的值与表达式或者表达式范围，若结果为真，则选择

若多个主语表达式（第一个 **WHEN** 之前的 **ALSO** 短语表示），则应有个数相同的宾语短语。

#### **例 1.8.1**



**01 CREDBFS PIC 9.**

**EVALUATE CREDBFS**

**WHEN 1**

**MOVE '1' TO CR-UDTK-TYP**

**WHEN 2**

**MOVE '5' TO CR-UDTK-TYP**

**WHEN 3**

**MOVE '2' TO CR-UDTK-TYP**

**WHEN 4**

**MOVE '1' TO CR-UDTK-TYP**

**WHEN 5**

**MOVE '4' TO CR-UDTK-TYP**

**WHEN OTHER**

**MOVE '0' TO CR-UDTK-TYP**

**END-EVALUATE.**



## 第九章 排序与合并

- 排序与合并的功能
- 如何实现这些功能





如下例

```
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
  01 StudentDetails.  
    02 StudentId      PIC 9(7).  
    02 StudentName    PIC X(10) .  
    02 DateOfBirth     PIC 9(6).  
    02 Gender          PIC X.    02 ClassCode      PIC 9(4).  
    02 Grand           PIC 9.
```

StudentFile 是一顺序文件，根据 StudentId 的升序排列。

假设我们要写一只程序显示每班课程的学生人数，该如何实现？

## 1 基本定义

所谓排序，就是根据一个或几个排序关键字来排序文件。

所谓合并，即取两个或多个记录结构相同的文件（已经排序），将其合并成一个文件

## 2 对环境部，数据部的要求

### 环境部

每个排序和合并文件都要有文件控制项，在 FILE-CONTROL 中定义

也可以在 I-O-CONTROL 中指定与其他文件共享的内存区

### 数据部

每个排序和合并文件都要有文件描述项，对应于文件控制项。

文件名前用保留字 SD

## 3 排序



格式如下：

```
SORT file1 { ON { ASCENDING } KEY { data1 } ... } ...  
                { DESCENDING }  
[ WITH DUPLICATES IN ORDER ]  
[ COLLATING SEQUENCE IS alphabet-name ]  
{ INPUT PROCEDURE IS procedure1 [ THROUGH procedure2 ] }  
  { USING { file2 } ... }  
{ OUTPUT PROCEDURE IS procedure3 [ THROUGH procedure4 ] }  
  { GIVING { file3 } ... }
```

使用时，要注意以下方面

关键字（data1 等等）应为排序文件 file1 中的一个数据项。

关键字的指定顺序定义 sort 过程所用关键字的顺序。第一个关键字是主关键字。

若记录的所有关键字都相同，如果有 with duplicates in order，则按照记录发送给 sort 语句的顺序返回，否则，顺序是任意的。

数字关键字按照代数值，非数字关键字按照字母顺序排列。

若有 collating sequence 从句，则按照 alphabet-name 作为比较的排列顺序。

USING 指定提供要排序的记录的文件。

INPUT PROCEDURE 指定一个或几个向排序合并模块提供记录的过程。

GIVING 语句将排好的记录些入一个或几个文件中。

OUTPUT PROCEDURE 指定一个或几个向排序合并模块写记录的过程。

输出文件要等到所有的输入文件处理完成后才打开，因此，可以用同一文件进行输入和输出

例 3.1



**ENVIRONMENT DIVISION.**

**INPUT-OUTPUT SECTION.**

**FILE-CONTROL.**

**SELECT WorkFile ASSIGN TO "WORK.TMP".**

◦

◦

**SD WorkFile.**

**01 WorkRecord.**

**02 FILLER**

**PIC X(24).**

**02 ClassCode**

**PIC 9(4).**

**02 GrandCode**

**PIC 9.**

◦

◦

◦

**PROCEDURE DIVISION.**

**Begin.**

**SORT WorkFile ON ASCENDING KEY GrandCode  
DESCENDING KEY ClassCode**

**USING StudentFile**

**GIVING SortedFile.**

**OPEN INPUT SortedFile.**

## 4 合并

格式如下：

**MERGE** file1 { **ON** { **ASCENDING** } **KEY** { data1 } ... } ...  
[ **COLLATING SEQUENCE** IS alphabet-name ]  
**USING** file 2 { file3 } ...  
{ **OUTPUT PROCEDURE** IS procedure1 [ **THROUGH** procedure2 ] }  
{ **GIVING** { file4 } ... }

启动合并过程，读取并合并两个或多个输入文件中的记录。



按顺序将记录写入文件或提供给程序处理

输出文件处理时，输入文件并不关闭，因此不能用同一文件进行输入和输出。



## 第十章 第九章 表格处理

- 基本概念
- 如果建立表格
- 如何初始化
- 如何使用



## 1 基本定义

表格——在内存中相邻的一段空间，其数据名相同，根据不同的角标相互区分。

定长表格——数据项个数固定的表格

变长表格——数据项个数可变

组表——在组层上使用 occurs 从句，组中的每个元素都是表中的元素

多维表格——<sup>a</sup> 用从属于带 OCCURS 从句的数据项的另一个 OCCURS 从句定义的数据项

## 2 定义表格

表的建立是通过 OCCURS 子句实现的，格式如下：

level-number  $\left[ \begin{array}{l} \text{Data 1} \\ \text{FILLER} \end{array} \right]$

OCCURS  $\left\{ \begin{array}{l} \text{integer 1 TIMES} \\ \text{integer 1 TO integer 2 TIMES DEPENDING ON data 2} \end{array} \right\}$

$\left[ \left\{ \begin{array}{l} \text{ASCENDING} \\ \text{DESCENDING} \end{array} \right\} \text{ KEY IS } \{ \text{data 3} \} \dots \right] \dots$

INDEXED BY { index 1 } ...]

### 例 2.1

01 GROUP-ITEM-A.

03 ITEM-A PIC 9(3) OCCURS 10 TIMES.

### 例 2.2

01 I-MAX PIC 99.

01 GROUP-ITEM-B.



03 ITEM-B PIC 9(3) OCCURS 1 TO 10 TIMES DEPENDING ON I-MAX.

### 例 2.3

01 GROUP-ITEM-C.

03 ITEM-C OCCURS 10 TIMES.

05 SUB-ITEM-A PIC 9.

05 SUB-ITEM-B PIC 9.

## 2.1 索引

INDEXED BY 从句后面定义一个或多个索引名。

索引名是用户定义字，不在其他地方定义。

索引名可以用作 PERFORM 语句的变量，用于 SET 和 SEARCH 语句，用作下标

KEY 短语表示表格按照 data 3 的顺序排列。

用于 SEARCH 语句带 ALL 短语时，表格的排序

KEY 不保证表格按顺序排列，也不使表格进行排序，只是希望表格用这个顺序。

### 例 2.1.1

```
01 ABOUT-NAME.  
    05 ABOUT-NAME-O OCCURS 26 TIMES  
        ASCENDING KEY IS OLD-NAME  
        INDEXED BY INDEX-NAME.  
            10 OLD-NAME          PIC 9(3).  
            10 NEW-NAME         PIC 9(9).
```

### 例 2.1.2



```
01 TEST-GROUP.  
  02 ITEMA OCCURS 4 TIMES.  
    03 SUB-ITEMA OCCURS 2 TIMES.  
      04 MINI-ITEMA PIC 9(8)V99.  
      04 MINI-ITEMB PIC 9(7).
```

ITEMA SUB-ITEMA	1		2		3		4	
	1	2	1	2	1	2	1	2

### 3 初始化表格

没有直接的静态表格初始化方法

若对带 OCCURS 从句的数据项指定 VALUE 值，或对带 OCCURS 从句的数据项所属的数据项指定 VALUE 值，则表中每个项目都初始化为该值。

若对带 OCCURS 从句的组数据项指定 VALUE 值，则不能在所属数据项中再指定 VALUE 值

若 OCCURS 从句定义变长表格，则初始化最多的表格。

#### 例 3.1

```
01 FRIEND-INFO.
```

```
  05 FRIEND-LIST OCCURS 100 TIMES.
```

```
    10 NAME PIC X(20) VALUE SPACES.
```

```
    10 GENDER PIC X VALUE SPACES.
```

#### 例 3.2

```
01 FRIEND-INFO.
```

```
  05 FRIEND-LIST OCCURS 100 TIMES VALUE SPACES.
```





10 NAME PIC X(20).

10 GENDER PIC X.

## 4 表格的引用

对表元素的引用有下标法和指标法两种方法，方法类似，但是内部实现不同。

下标法：引用表中的数据项时，要提供给标识符特定项目的下标或下标清单。表格的每一维都需要一个整型下标。

可以用整数或整数类型的数值数据项作为下标。

指标法：定义一个指标数据项，由编译程序自动形成的存指标的区域，有特殊的存储区域分配和特定的表达形式，不需要也不允许在程序的其他地方另行定义。

采用指标法，指标名的内容直接给出了特定表元素的相对地址（以二进制形式），无须换算，因此寻址响应更快。

采用指标法，OCCURS 子句的形式应该有如下形式

OCCURS n TIMES INDEXED BY 指标名

## 5 处理表格的2种语法

PERFORM VARYING

SEARCH

PERFORM VARYING

线外

PERFORM 语句中包含过程名，表格处理在过程中执行

线上

PERFORM 语句中包含要执行的表格处理

线外

线上



```
PERFORM VARYING INDEX-NAME FROM 1 BY 1 UNTIL INDEX-NAME > 26
```

```
.....
```

```
END-PERFORM.
```

```
SEARCH
```

线性查找——用某个表格索引，从头往下照

排序表格查找——查找已经排序的表格

```
01 ABOUT-NAME.
```

```
    05 ABOUT-NAME-O OCCURS 26 TIMES  
        ASCENDING KEY IS OLD-NAME  
        INDEXED BY INDEX-NAME.
```

```
    10 OLD-NAME          PIC 9(3).
```

```
    10 NEW-NAME          PIC 9(9).
```

```
SEARCH ALL ABOUT-NAME-O
```

```
AT END
```

```
    DISPLAY 'NO SUCH OLD NAME:', MID-FKH
```

```
    MOVE 110210000 TO NEW-BRN
```

```
    WHEN OLD-NAME(INDEX-NAME) IS MID-FKH
```

```
        MOVE NEW-NAME(INDEX-NAME) TO NEW-BRN
```

```
END-SEARCH.
```

子程序

子程序的分类

如何调用子程序

子程序的范围

子程序

子程序可以包含在主程序中

也可以是单独编译的子程序，运行时与主程序链接

可以接受参数输入，向调用程序返回结果

内部子程序

程序放在源程序中

外部子程序

单独编译要执行的程序，作为一个运行单元

公用子程序

子程序通常只能由包含它的程序调用

相邻的子程序通常不能互相调用

若在 PROGRAM-ID 中声明程序为 COMMON（公用）则可以修改上述两个属性



只有程序中包含的程序才能使用 **common**

公用程序程序中的子程序仍旧无法被外部程序调用

全局数据和文件

通常，数据和文件都是专用的

可以在定义中加入 **GLOBAL** 从句，使程序中的任意子程序都能利用这个数据或文件

**GLOBAL** 只能用于定义 01 层数据

定义全局文件，在 **FD** 语句时使用 **GLOBAL** 从句

数据项具有 **GLOBAL** 属性时，所属的全部文件名都具有全局属性

**GLOBAL** 只能是子程序调用主程序的数据或文件，主程序不能调用子程序的数据或文件

如果子程序定义了相同名称的数据或文件，则在子程序中，使用子程序的定义

## **CALL**

子程序参数的传递

调用程序中的 **CALL** 语句列出要传递的参数

被调用程序中的过程部标题，列出相应的 **CALL** 语句

被调用程序中的连接节，提供参数存储空间

### **CALL Parameters**

#### **CALL Parameters**

位置对应 而非名称对应

参数传递的方式

### **BY CONTENT**

参数的内容复制道被调用程序的连接节，被调用程序实际上拥有数据的局部拷贝

被调用程序可以修改这个数据，但调用程序中的参数的内容不变

### **BY REFERENCE**

被调用程序引用参数的内容

子程序对数据的任何改变都反映到调用程序的数据中

子程序通过写入调入用的参数而向调用程序返回结果

参数传递的机制

子程序的初始状态

子程序再次被调用时，仍保持上次推出是的状态

为避免此种行为，使用 **INITIAL** 从句

主程序若使用 **INITIAL** 从句，则其子程序不必声明，也都具有 **INITIAL** 属性

### **CANCEL** 语句

重新初始化子程序，但不提供程序以 **INITIAL** 属性时，使用此语句

**CANCEL { id-lit }.....**

– **Id-lit**为可调用的程序名



- CANCEL语句引用的程序不实际执行
- 引用的程序返回初始状态，关闭打开的文件，使用VALUE的数据项复位原值

取消程序时，同时取消其中包含的任何程序