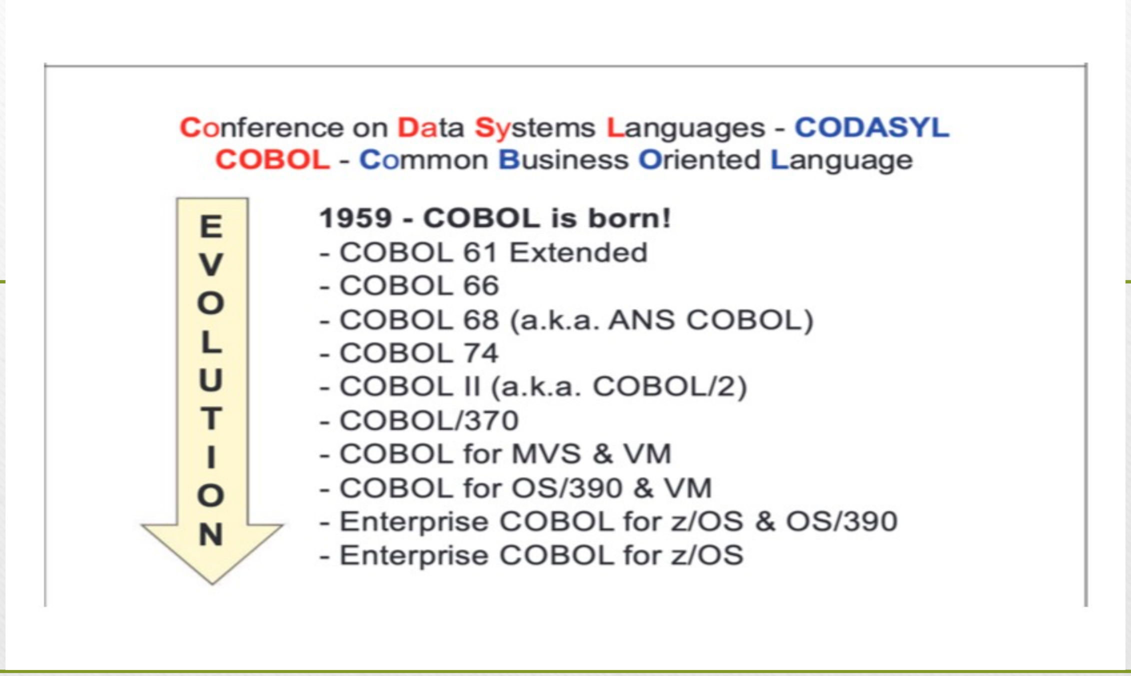
# ההיסטוריה של קובול



Conference on Data Systems Languages - CODASYL

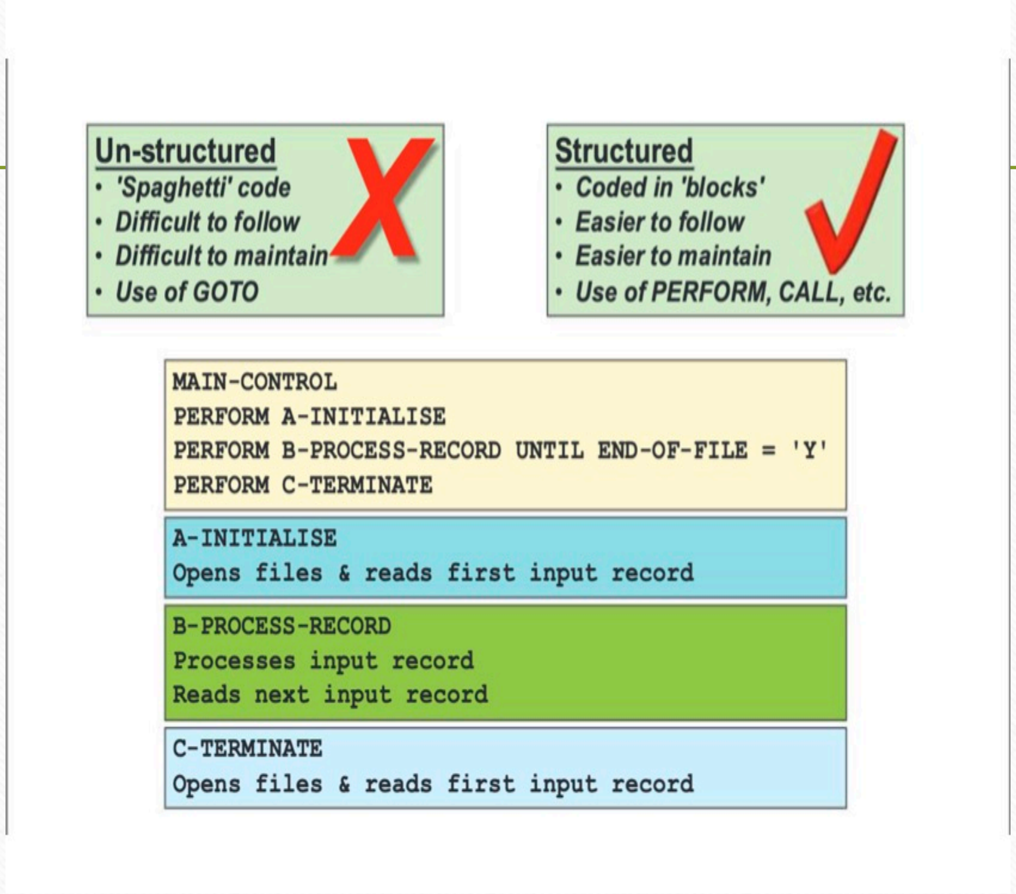
COBOL - Common Business Oriented Language

**1959 - COBOL is born!**

- COBOL 61 Extended
- COBOL 66
- COBOL 68 (a.k.a. ANS COBOL)
- COBOL 74
- COBOL II (a.k.a. COBOL/2)
- COBOL/370
- COBOL for MVS & VM
- COBOL for OS/390 & VM
- Enterprise COBOL for z/OS & OS/390
- Enterprise COBOL for z/OS

# Structured vs. unstructured code

(Cobol is not object oriented)



**Un-structured** ✗
- *'Spaghetti' code*
- *Difficult to follow*
- *Difficult to maintain*
- *Use of GOTO*

**Structured** ✓
- *Coded in 'blocks'*
- *Easier to follow*
- *Easier to maintain*
- *Use of PERFORM, CALL, etc.*

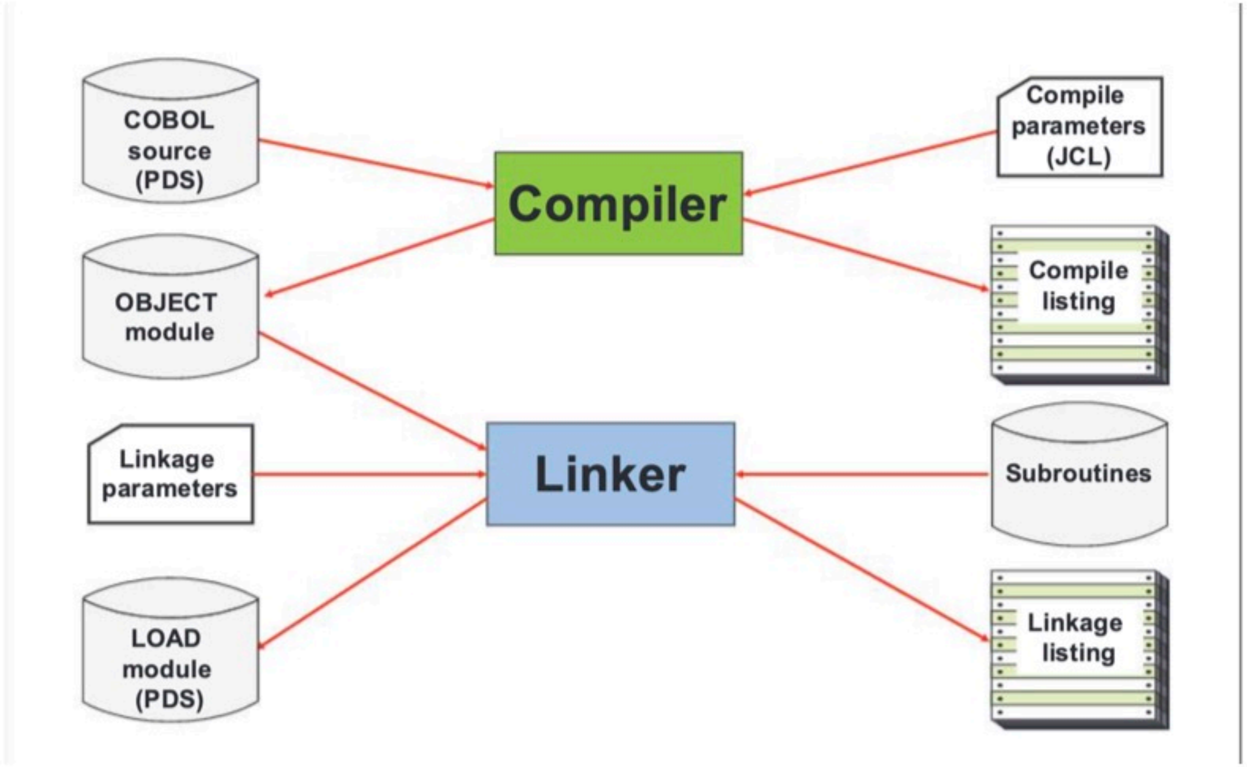
```
MAIN-CONTROL
PERFORM A-INITIALISE
PERFORM B-PROCESS-RECORD UNTIL END-OF-FILE = 'Y'
PERFORM C-TERMINATE
```

```
A-INITIALISE
Opens files & reads first input record
```

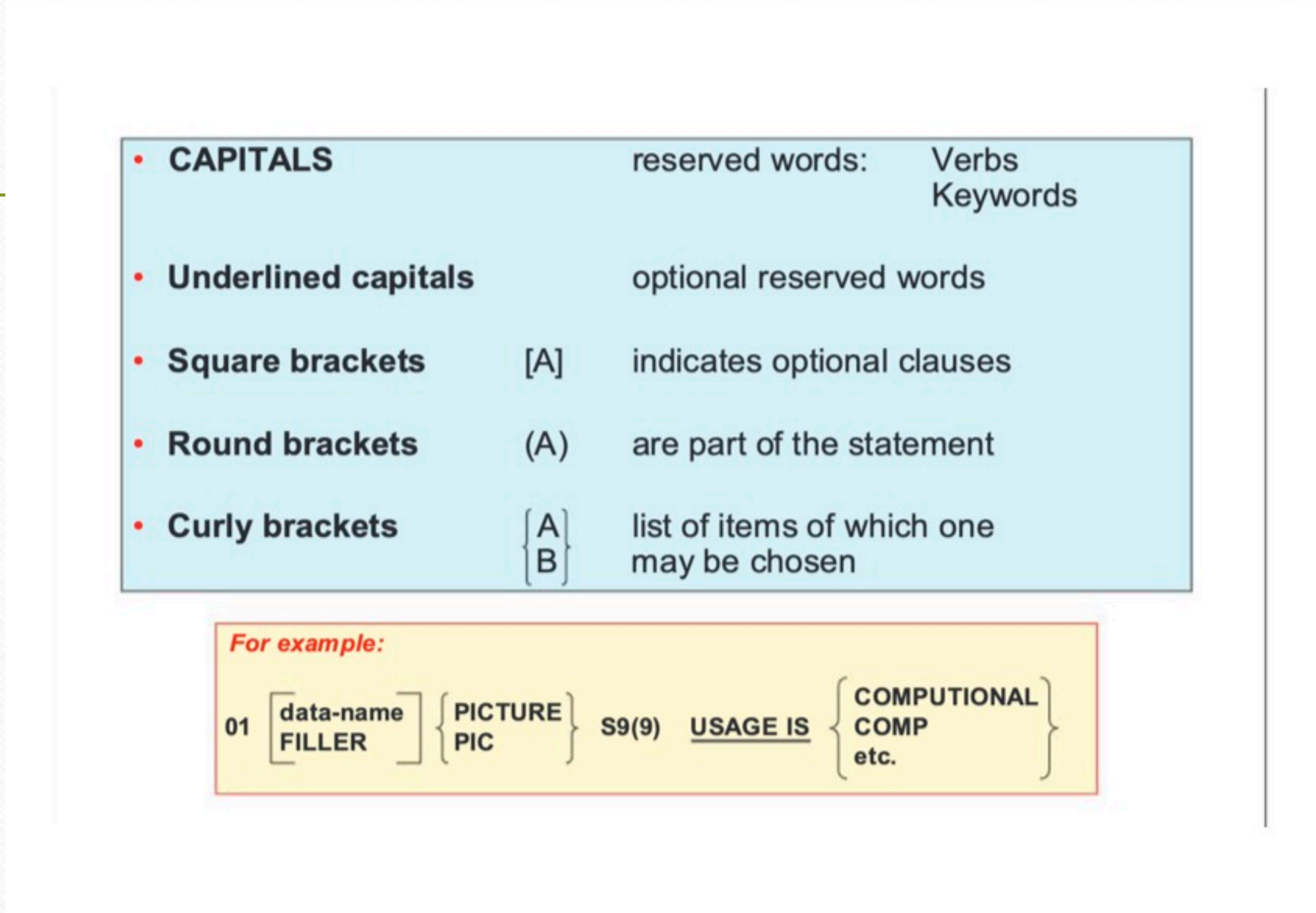
```
B-PROCESS-RECORD
Processes input record
Reads next input record
```

```
C-TERMINATE
Opens files & reads first input record
```

# Creating an executable (load module)



# Manual Conventions (IBM style)

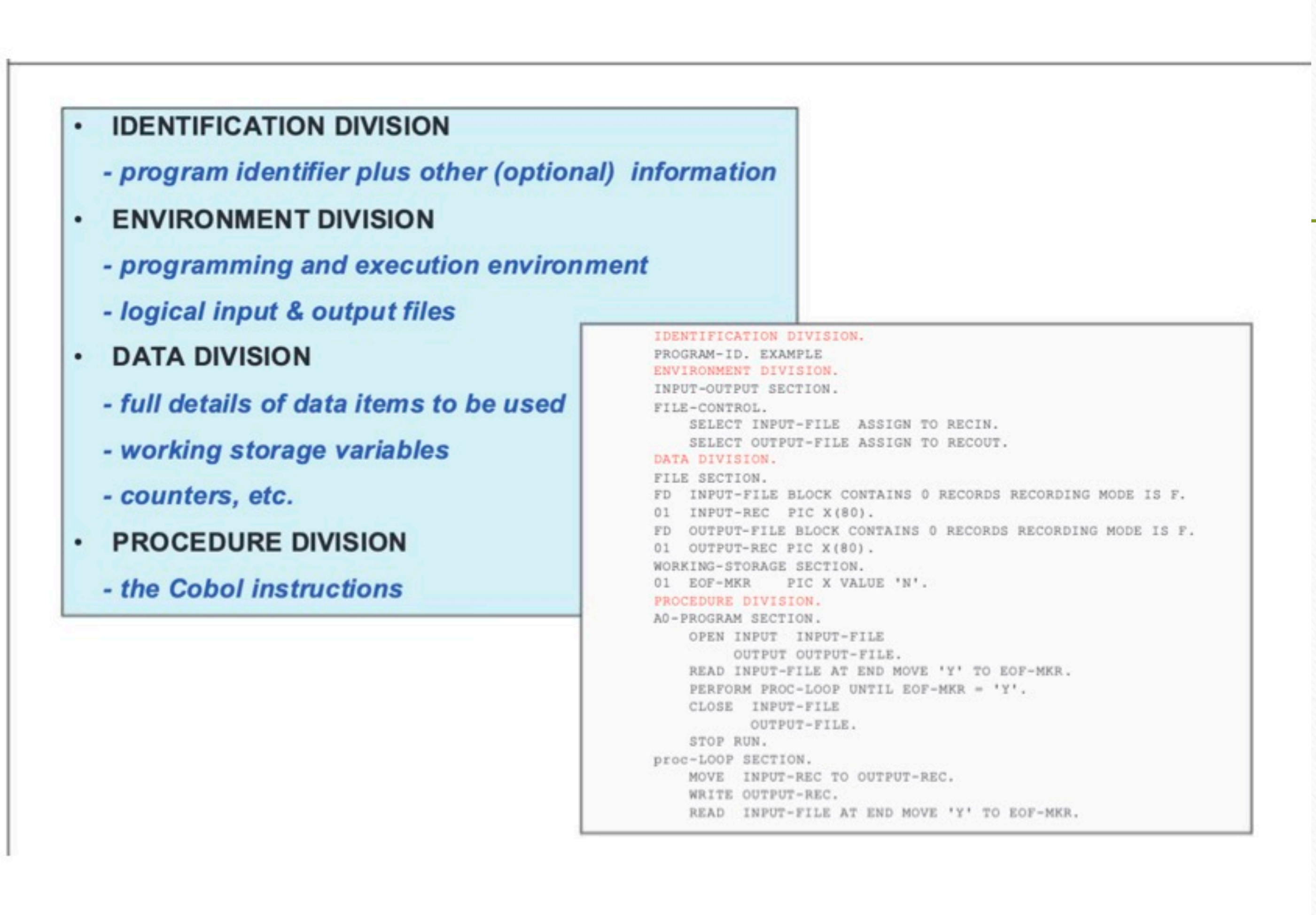


| | | | |
|---|---|---|---|
| CAPITALS | | reserved words: | Verbs<br>Keywords |
| Underlined capitals | | optional reserved words | |
| Square brackets | [A] | indicates optional clauses | |
| Round brackets | (A) | are part of the statement | |
| Curly brackets | $\begin{Bmatrix}A\\B\end{Bmatrix}$ | list of items of which one may be chosen | |

*For example:*

$$01 \begin{bmatrix}\text{data-name}\\\text{FILLER}\end{bmatrix} \begin{Bmatrix}\text{PICTURE}\\\text{PIC}\end{Bmatrix} \text{ S9(9) } \underline{\text{USAGE IS}} \begin{Bmatrix}\text{COMPUTIONAL}\\\text{COMP}\\\text{etc.}\end{Bmatrix}$$

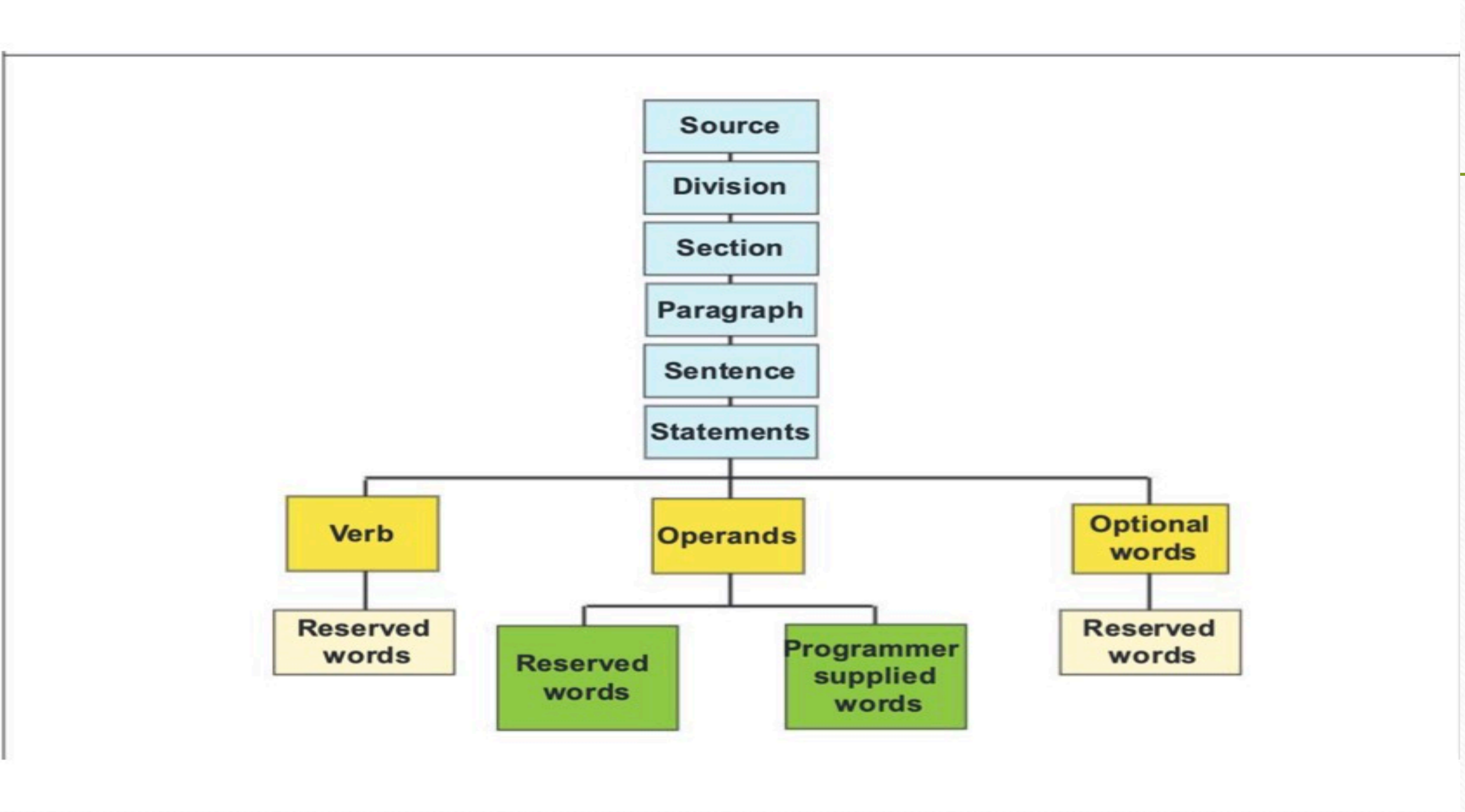
# Cobol Program structure



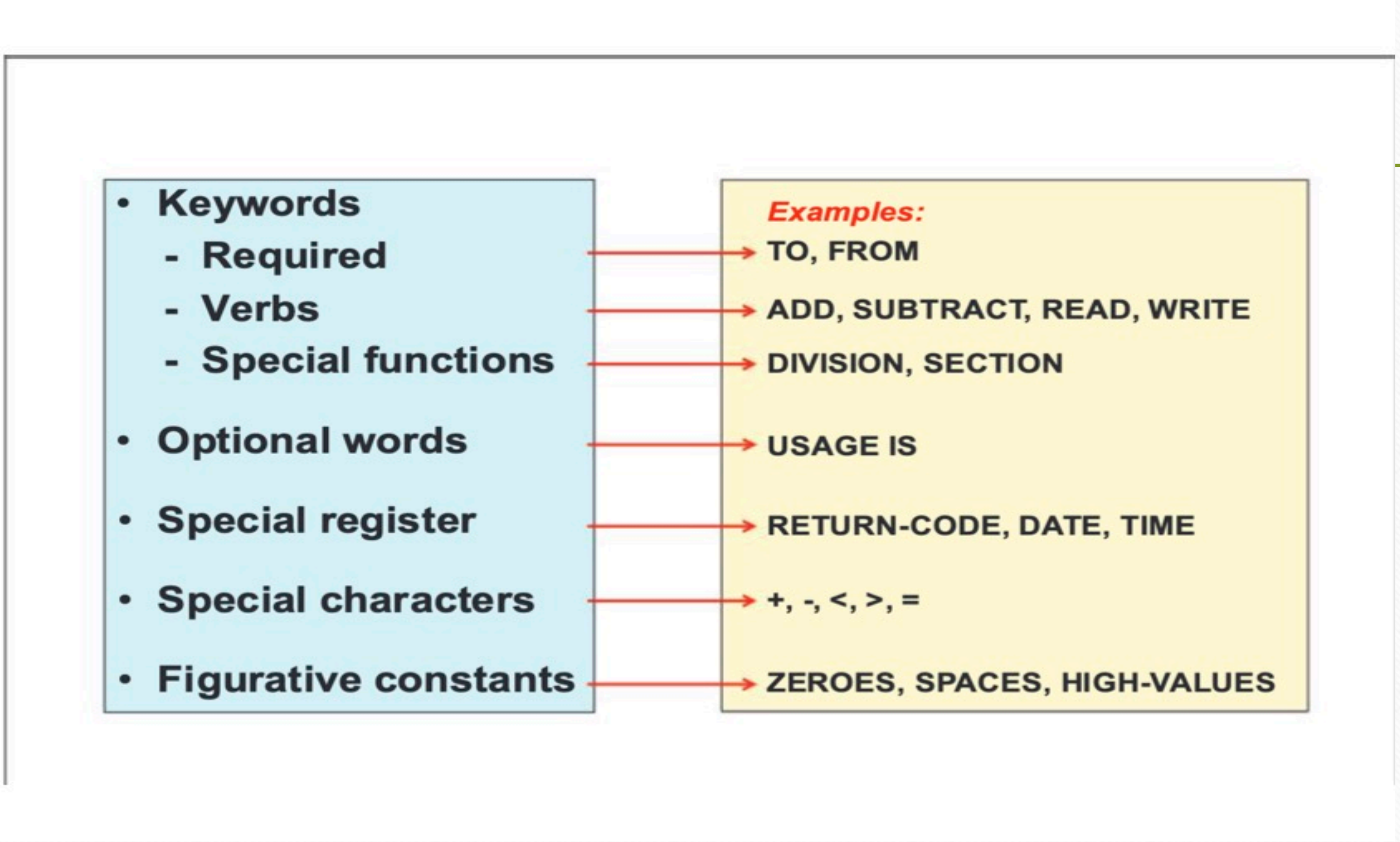
- IDENTIFICATION DIVISION
  - *- program identifier plus other (optional) information*
- ENVIRONMENT DIVISION
  - *- programming and execution environment*
  - *- logical input & output files*
- DATA DIVISION
  - *- full details of data items to be used*
  - *- working storage variables*
  - *- counters, etc.*
- PROCEDURE DIVISION
  - *- the Cobol instructions*

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE  ASSIGN TO RECIN.
    SELECT OUTPUT-FILE ASSIGN TO RECOUT.
DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
01  INPUT-REC  PIC X(80).
FD  OUTPUT-FILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
01  OUTPUT-REC PIC X(80).
WORKING-STORAGE SECTION.
01  EOF-MKR    PIC X VALUE 'N'.
PROCEDURE DIVISION.
A0-PROGRAM SECTION.
    OPEN INPUT  INPUT-FILE
         OUTPUT OUTPUT-FILE.
    READ INPUT-FILE AT END MOVE 'Y' TO EOF-MKR.
    PERFORM PROC-LOOP UNTIL EOF-MKR = 'Y'.
    CLOSE  INPUT-FILE
           OUTPUT-FILE.
    STOP RUN.
proc-LOOP SECTION.
    MOVE  INPUT-REC TO OUTPUT-REC.
    WRITE OUTPUT-REC.
    READ  INPUT-FILE AT END MOVE 'Y' TO EOF-MKR.
```

# Cobol Language Hierarchy

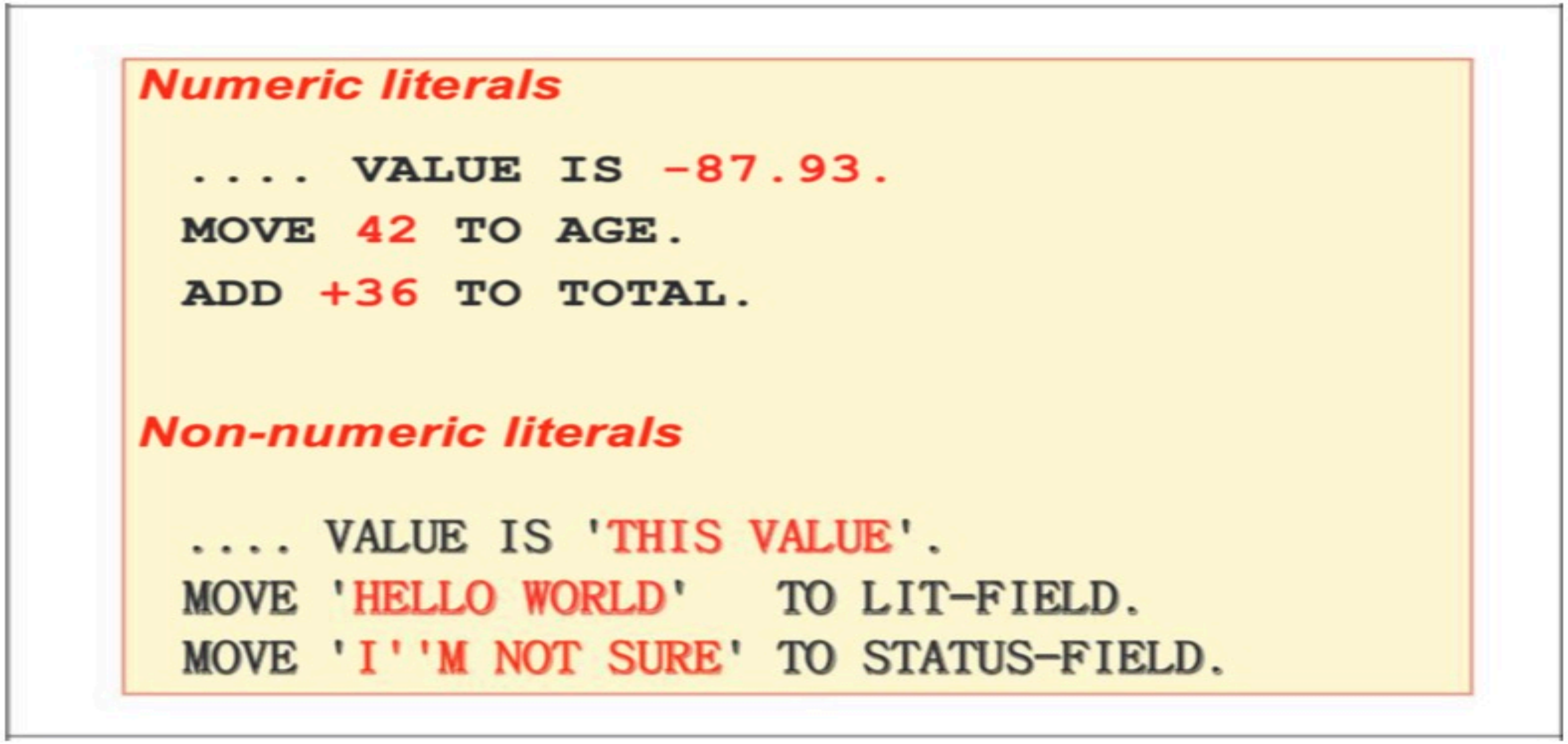


# Language components – sample keywords



| | *Examples:* |
|---|---|
| • **Keywords** | |
| - **Required** | TO, FROM |
| - **Verbs** | ADD, SUBTRACT, READ, WRITE |
| - **Special functions** | DIVISION, SECTION |
| • **Optional words** | USAGE IS |
| • **Special register** | RETURN-CODE, DATE, TIME |
| • **Special characters** | +, -, <, >, = |
| • **Figurative constants** | ZEROES, SPACES, HIGH-VALUES |

# Command Example (verbose)



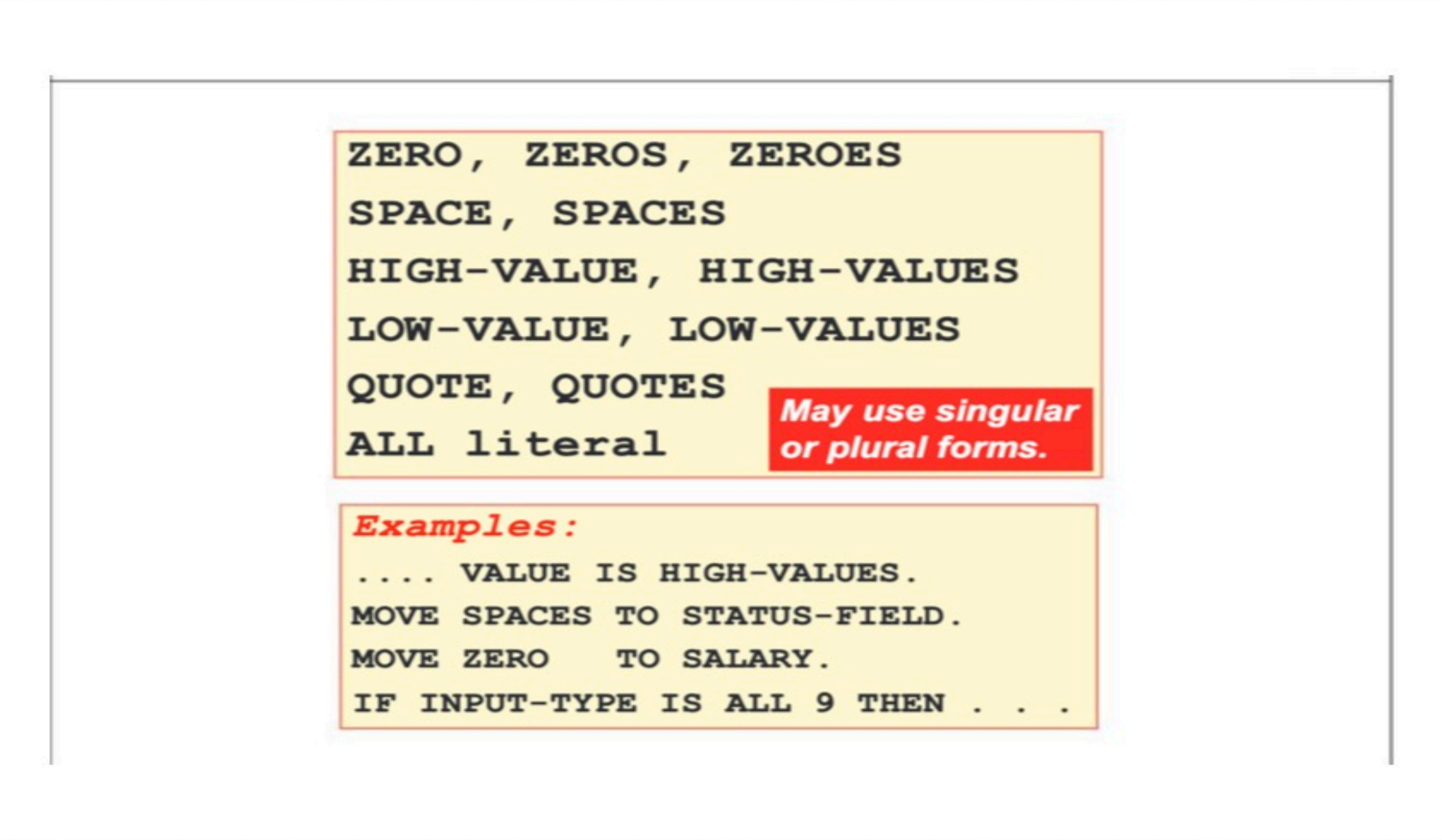
*Numeric literals*

```
.... VALUE IS -87.93.
MOVE 42 TO AGE.
ADD +36 TO TOTAL.
```

*Non-numeric literals*

```
.... VALUE IS 'THIS VALUE'.
MOVE 'HELLO WORLD'   TO LIT-FIELD.
MOVE 'I''M NOT SURE' TO STATUS-FIELD.
```

# Literal names representing values



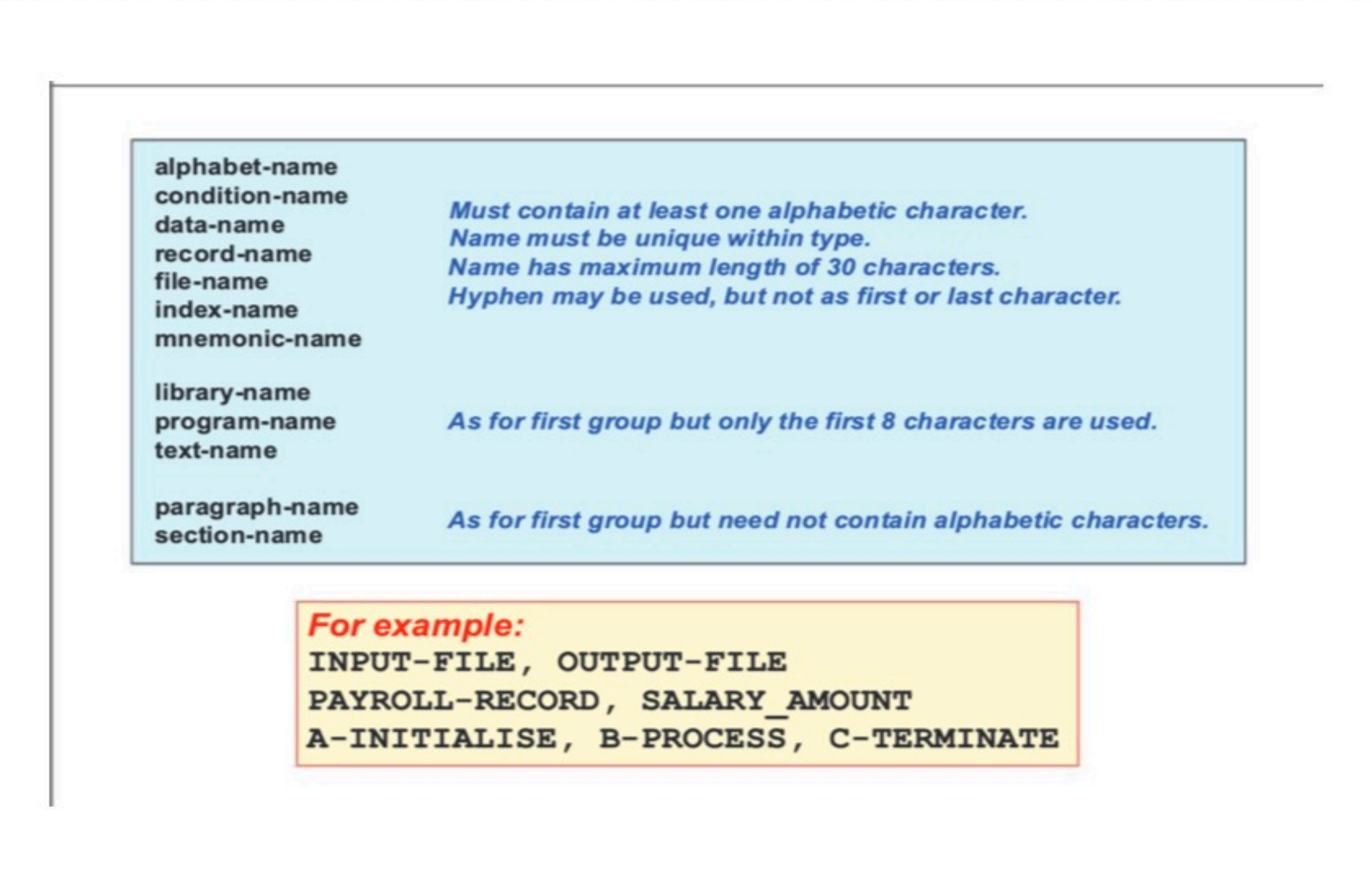
```
ZERO, ZEROS, ZEROES
SPACE, SPACES
HIGH-VALUE, HIGH-VALUES
LOW-VALUE, LOW-VALUES
QUOTE, QUOTES
ALL literal
```

*May use singular or plural forms.*

*Examples:*

```
.... VALUE IS HIGH-VALUES.
MOVE SPACES TO STATUS-FIELD.
MOVE ZERO   TO SALARY.
IF INPUT-TYPE IS ALL 9 THEN . . .
```

# Naming rules for user defined names

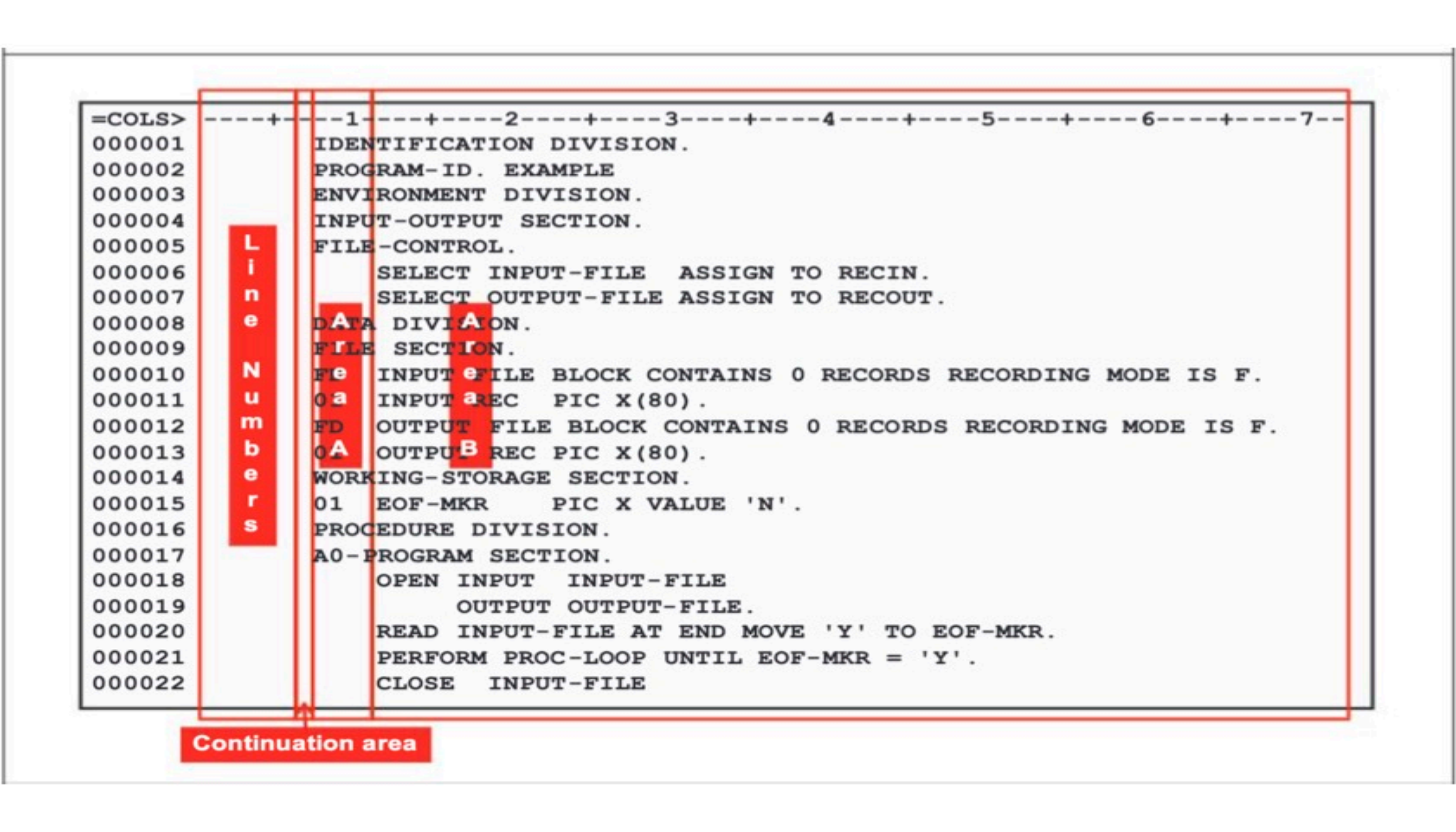


| | |
|---|---|
| alphabet-name<br>condition-name<br>data-name<br>record-name<br>file-name<br>index-name<br>mnemonic-name | *Must contain at least one alphabetic character.*<br>*Name must be unique within type.*<br>*Name has maximum length of 30 characters.*<br>*Hyphen may be used, but not as first or last character.* |
| library-name<br>program-name<br>text-name | *As for first group but only the first 8 characters are used.* |
| paragraph-name<br>section-name | *As for first group but need not contain alphabetic characters.* |

*For example:*

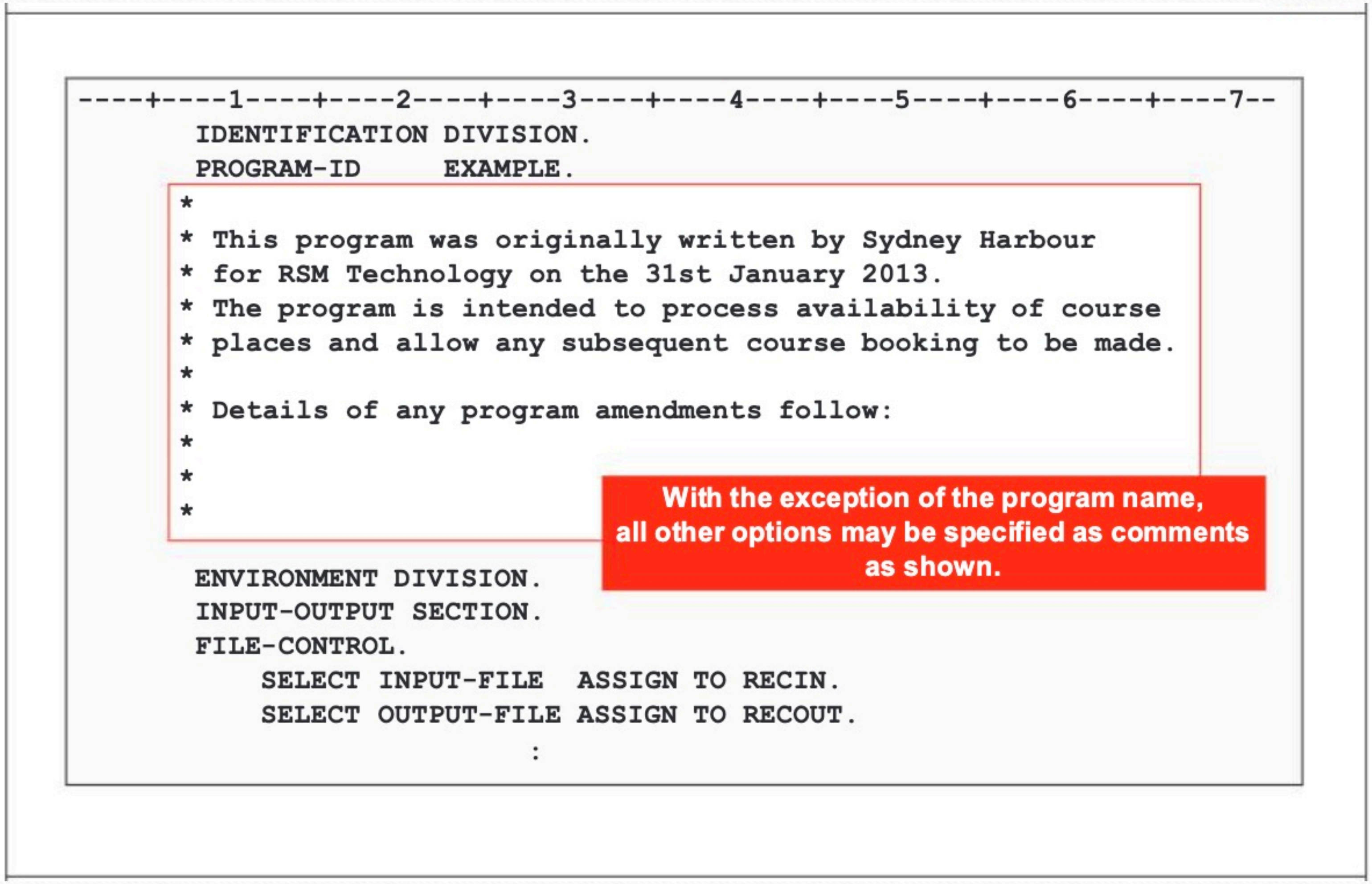
```
INPUT-FILE, OUTPUT-FILE
PAYROLL-RECORD, SALARY_AMOUNT
A-INITIALISE, B-PROCESS, C-TERMINATE
```

# Column designation (areas A, B)



```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001        IDENTIFICATION DIVISION.
000002        PROGRAM-ID. EXAMPLE
000003        ENVIRONMENT DIVISION.
000004        INPUT-OUTPUT SECTION.
000005        FILE-CONTROL.
000006            SELECT INPUT-FILE  ASSIGN TO RECIN.
000007            SELECT OUTPUT-FILE ASSIGN TO RECOUT.
000008        DATA DIVISION.
000009        FILE SECTION.
000010        FD  INPUT-FILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
000011        01  INPUT-REC  PIC X(80).
000012        FD  OUTPUT-FILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
000013        01  OUTPUT-REC PIC X(80).
000014        WORKING-STORAGE SECTION.
000015        01  EOF-MKR      PIC X VALUE 'N'.
000016        PROCEDURE DIVISION.
000017        A0-PROGRAM SECTION.
000018            OPEN INPUT  INPUT-FILE
000019                 OUTPUT OUTPUT-FILE.
000020            READ INPUT-FILE AT END MOVE 'Y' TO EOF-MKR.
000021            PERFORM PROC-LOOP UNTIL EOF-MKR = 'Y'.
000022            CLOSE  INPUT-FILE
```

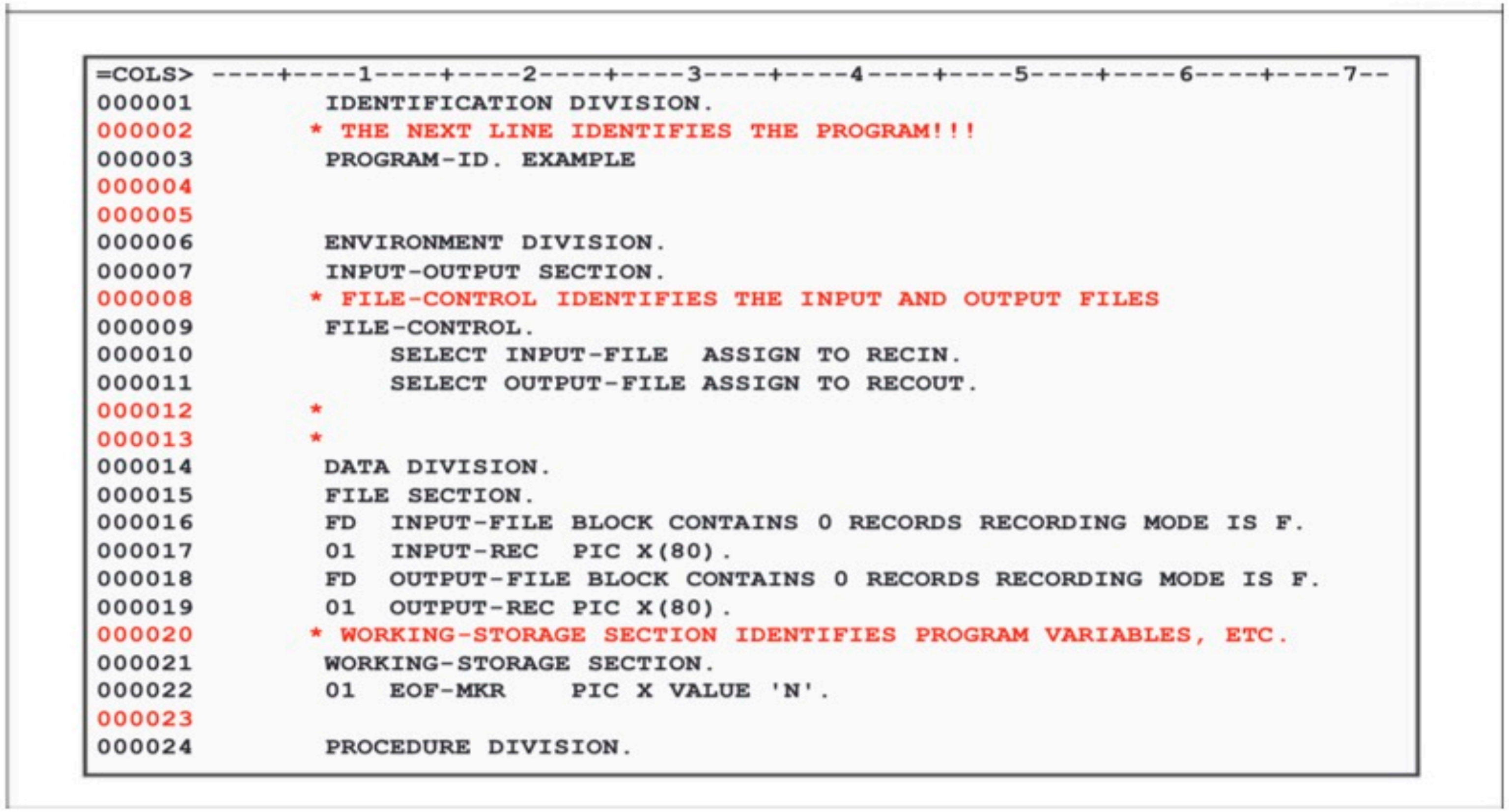
# Comments



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       IDENTIFICATION DIVISION.
       PROGRAM-ID     EXAMPLE.
      *
      * This program was originally written by Sydney Harbour
      * for RSM Technology on the 31st January 2013.
      * The program is intended to process availability of course
      * places and allow any subsequent course booking to be made.
      *
      * Details of any program amendments follow:
      *
      *
      *
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE  ASSIGN TO RECIN.
           SELECT OUTPUT-FILE ASSIGN TO RECOUT.
                           :
```

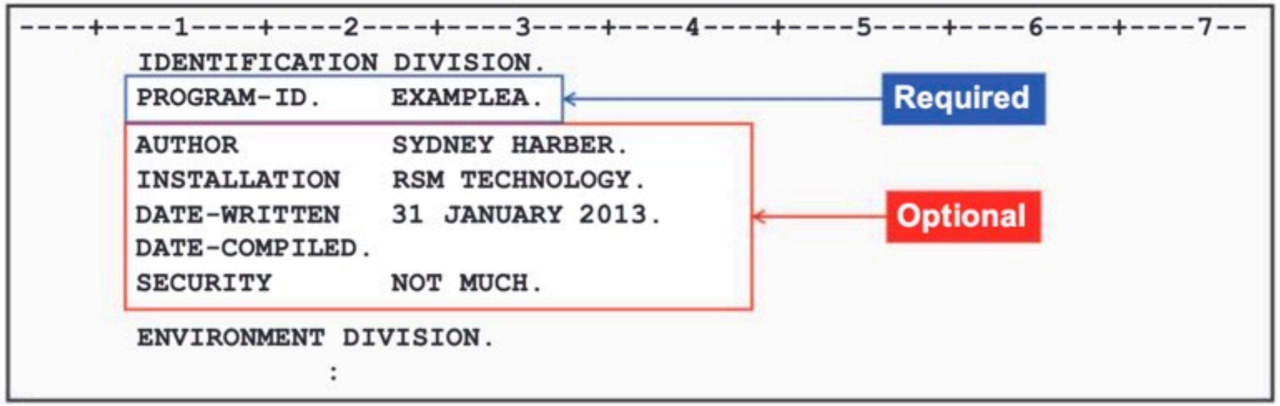
With the exception of the program name, all other options may be specified as comments as shown.

# Sample program with comments / blank lines (red)



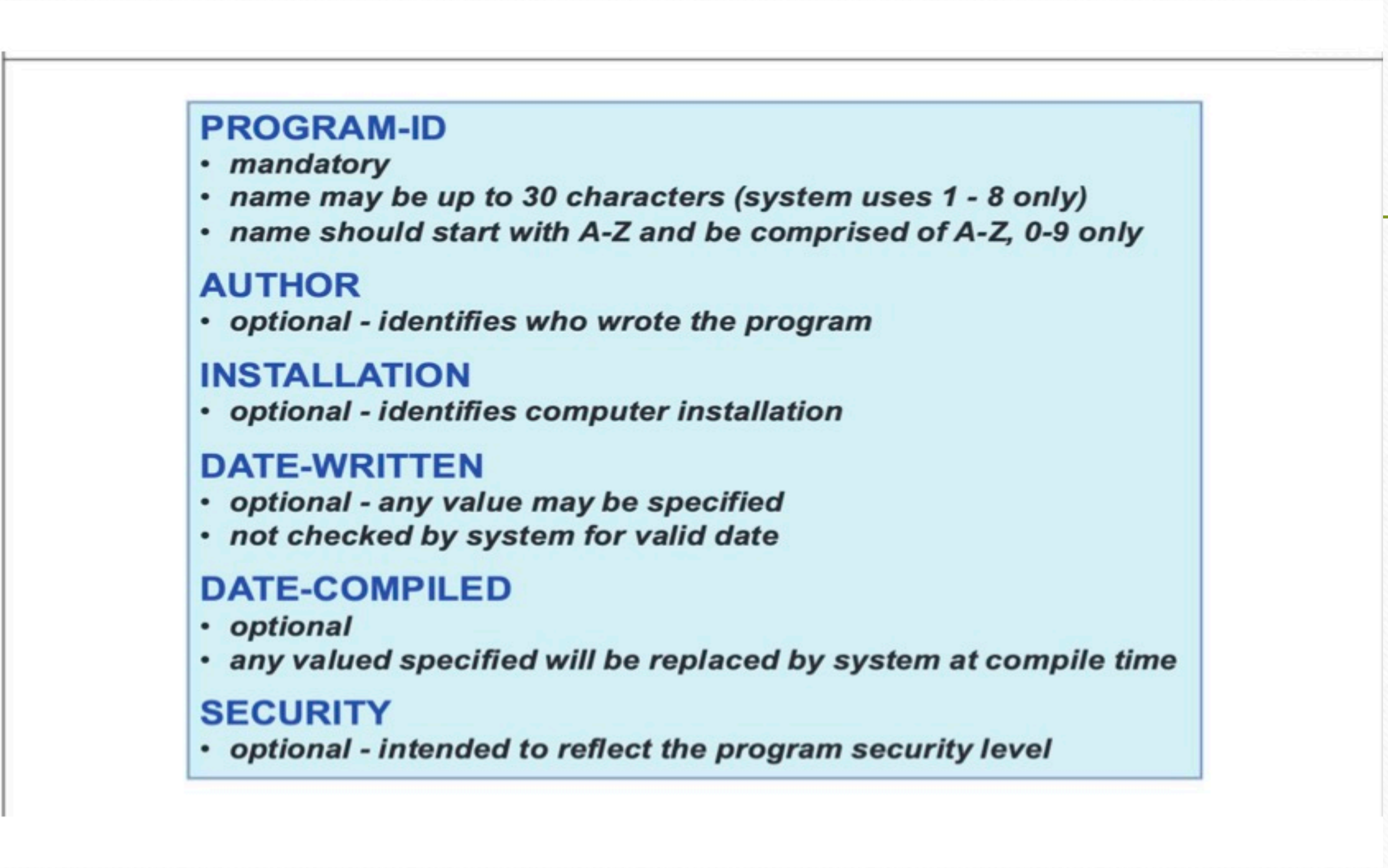
```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001        IDENTIFICATION DIVISION.
000002       * THE NEXT LINE IDENTIFIES THE PROGRAM!!!
000003        PROGRAM-ID. EXAMPLE
000004
000005
000006        ENVIRONMENT DIVISION.
000007        INPUT-OUTPUT SECTION.
000008       * FILE-CONTROL IDENTIFIES THE INPUT AND OUTPUT FILES
000009        FILE-CONTROL.
000010            SELECT INPUT-FILE  ASSIGN TO RECIN.
000011            SELECT OUTPUT-FILE ASSIGN TO RECOUT.
000012       *
000013       *
000014        DATA DIVISION.
000015        FILE SECTION.
000016        FD  INPUT-FILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
000017        01  INPUT-REC  PIC X(80).
000018        FD  OUTPUT-FILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
000019        01  OUTPUT-REC PIC X(80).
000020       * WORKING-STORAGE SECTION IDENTIFIES PROGRAM VARIABLES, ETC.
000021        WORKING-STORAGE SECTION.
000022        01  EOF-MKR    PIC X VALUE 'N'.
000023
000024        PROCEDURE DIVISION.
```

# ID DIVISION



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    EXAMPLEA.
       AUTHOR         SYDNEY HARBER.
       INSTALLATION   RSM TECHNOLOGY.
       DATE-WRITTEN   31 JANUARY 2013.
       DATE-COMPILED.
       SECURITY       NOT MUCH.

       ENVIRONMENT DIVISION.
                  :
```

# IDENTIFICATION DIVISION sentences



**PROGRAM-ID**

- *mandatory*
- *name may be up to 30 characters (system uses 1 - 8 only)*
- *name should start with A-Z and be comprised of A-Z, 0-9 only*

**AUTHOR**

- *optional - identifies who wrote the program*

**INSTALLATION**

- *optional - identifies computer installation*

**DATE-WRITTEN**

- *optional - any value may be specified*
- *not checked by system for valid date*

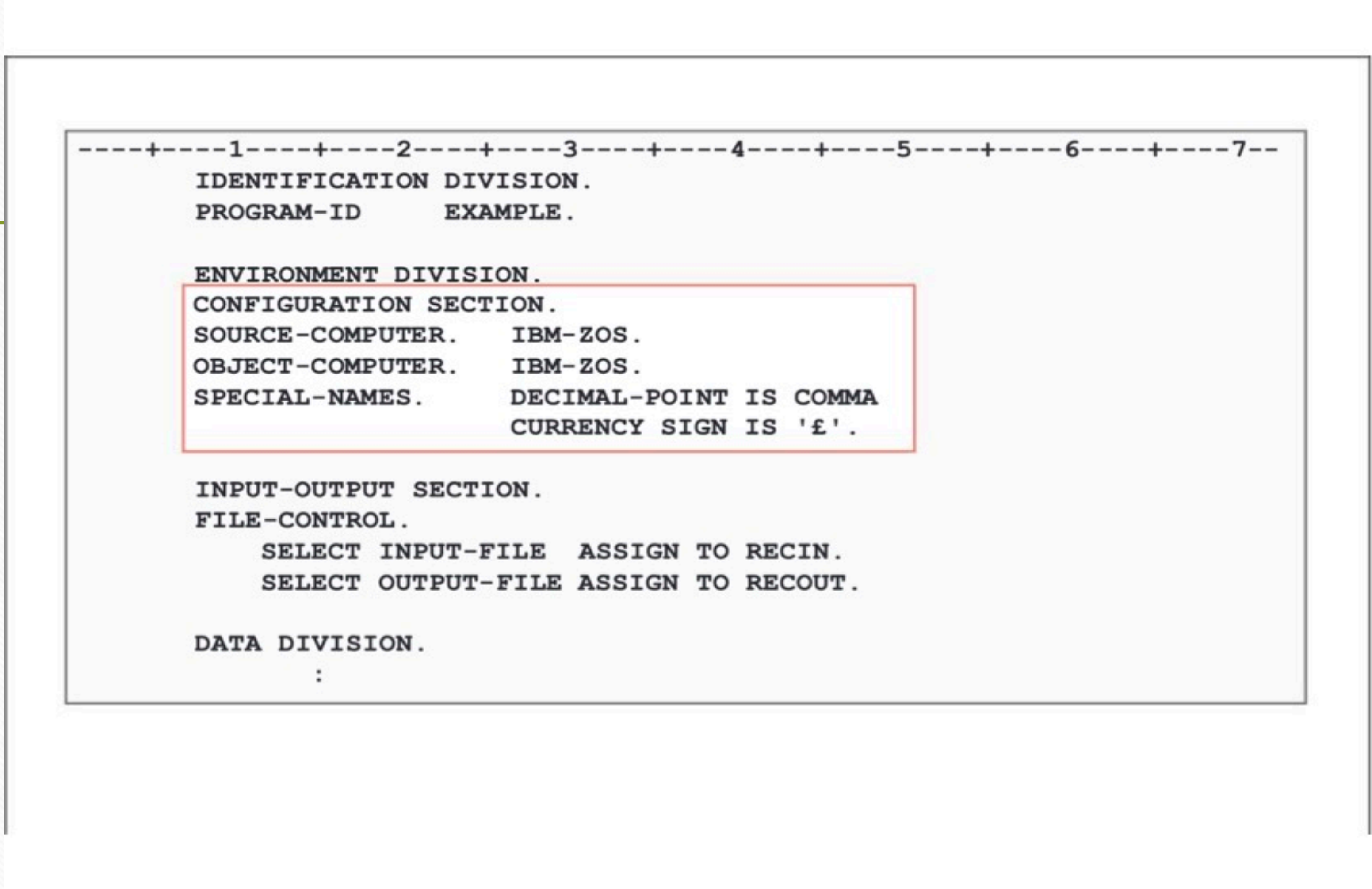
**DATE-COMPILED**

- *optional*
- *any valued specified will be replaced by system at compile time*

**SECURITY**

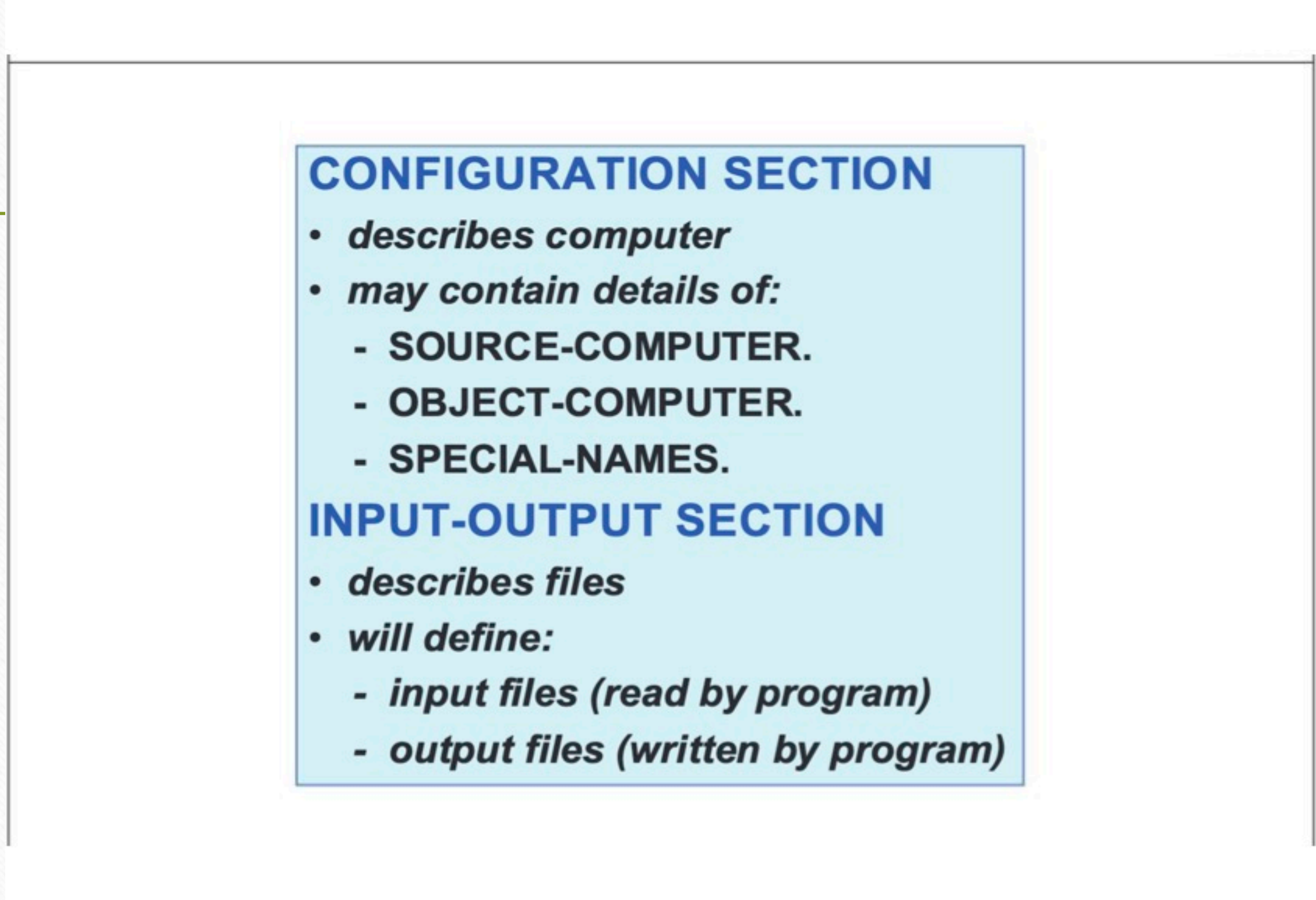
- *optional - intended to reflect the program security level*

# Environment Division



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       IDENTIFICATION DIVISION.
       PROGRAM-ID     EXAMPLE.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.   IBM-ZOS.
       OBJECT-COMPUTER.   IBM-ZOS.
       SPECIAL-NAMES.     DECIMAL-POINT IS COMMA
                          CURRENCY SIGN IS '£'.

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE  ASSIGN TO RECIN.
           SELECT OUTPUT-FILE ASSIGN TO RECOUT.

       DATA DIVISION.
              :
```

# Sections of the ENV Division



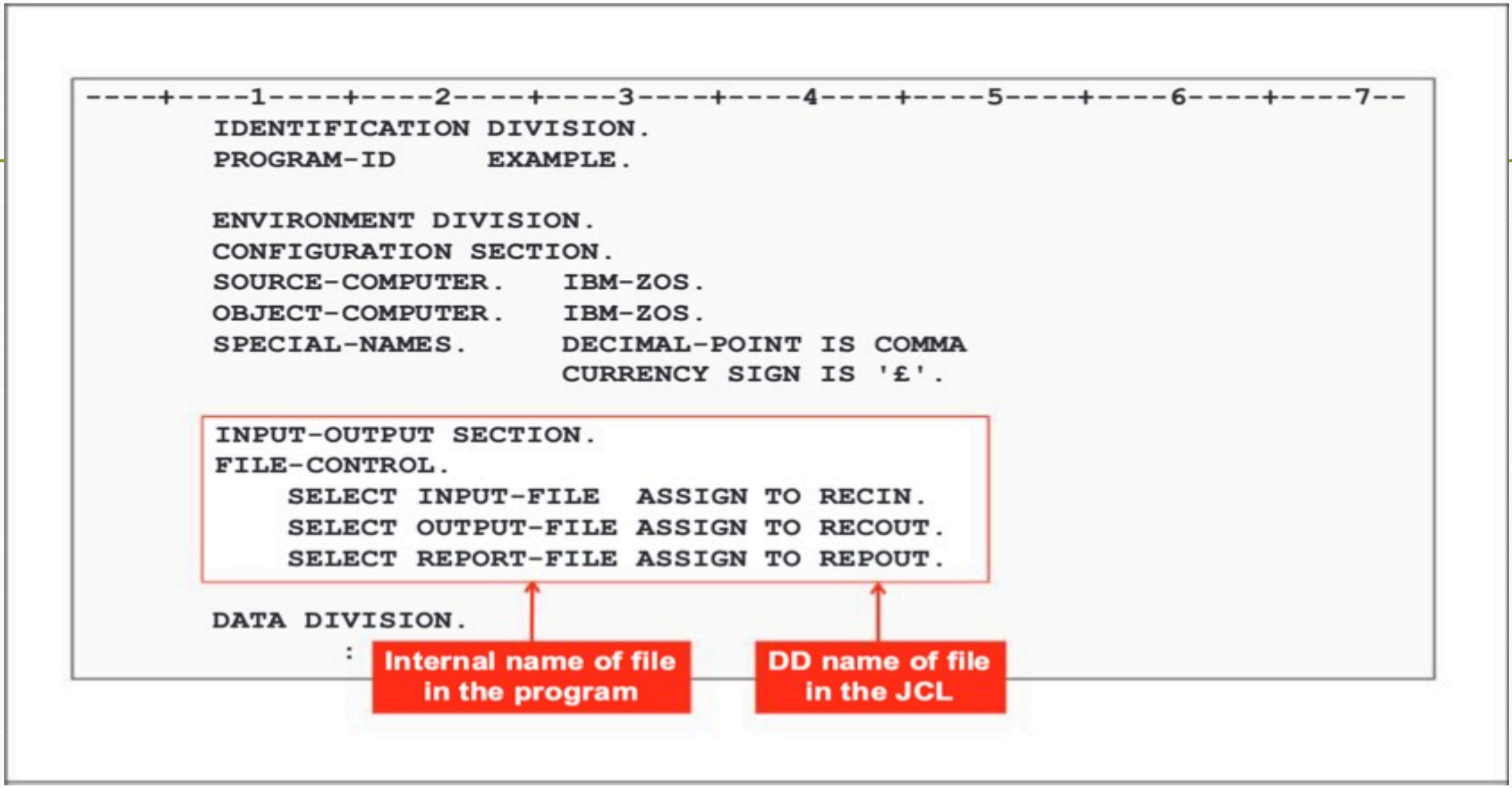
## CONFIGURATION SECTION

- *describes computer*
- *may contain details of:*
  - SOURCE-COMPUTER.
  - OBJECT-COMPUTER.
  - SPECIAL-NAMES.

## INPUT-OUTPUT SECTION

- *describes files*
- *will define:*
  - *input files (read by program)*
  - *output files (written by program)*

# INPUT-OUTPUT SECTION , FILE-CONTROL

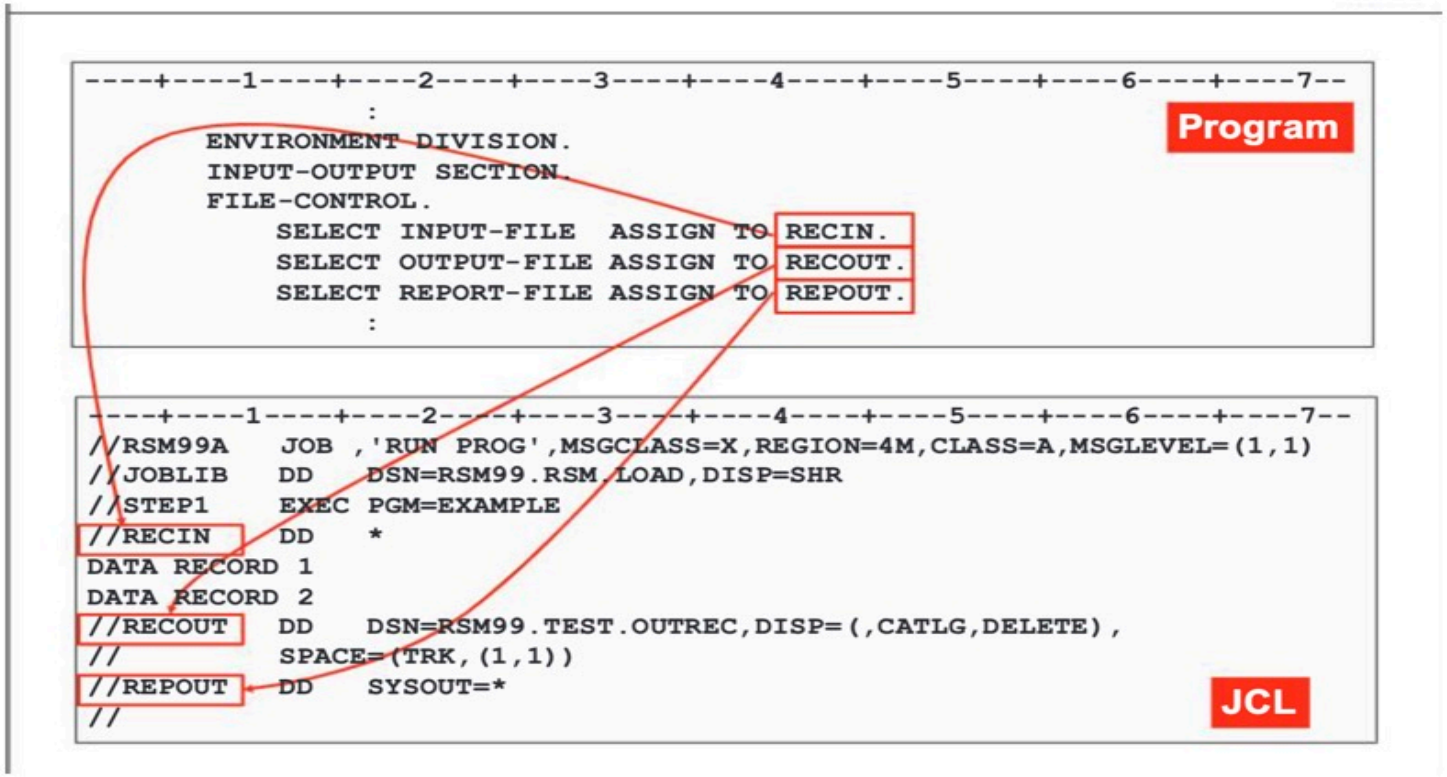


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       IDENTIFICATION DIVISION.
       PROGRAM-ID     EXAMPLE.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.   IBM-ZOS.
       OBJECT-COMPUTER.   IBM-ZOS.
       SPECIAL-NAMES.     DECIMAL-POINT IS COMMA
                          CURRENCY SIGN IS '£'.

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE  ASSIGN TO RECIN.
           SELECT OUTPUT-FILE ASSIGN TO RECOUT.
           SELECT REPORT-FILE ASSIGN TO REPOUT.

       DATA DIVISION.
              :
```

Internal name of file in the program

DD name of file in the JCL

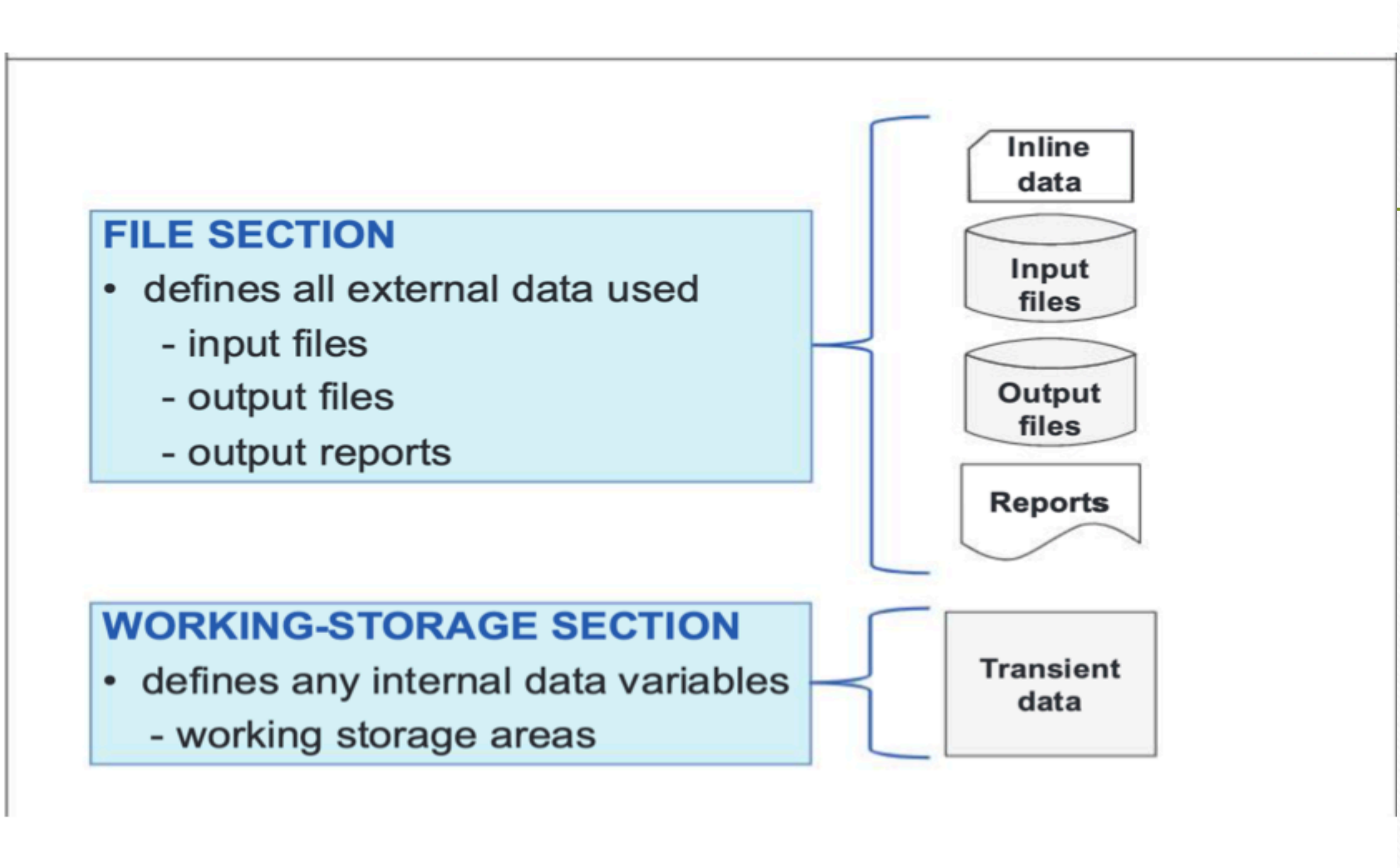
# Connecting program files to JCL



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
                        :
           ENVIRONMENT DIVISION.
           INPUT-OUTPUT SECTION.
           FILE-CONTROL.
                 SELECT INPUT-FILE  ASSIGN TO RECIN.
                 SELECT OUTPUT-FILE ASSIGN TO RECOUT.
                 SELECT REPORT-FILE ASSIGN TO REPOUT.
                        :
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
//RSM99A    JOB ,'RUN PROG',MSGCLASS=X,REGION=4M,CLASS=A,MSGLEVEL=(1,1)
//JOBLIB    DD    DSN=RSM99.RSM.LOAD,DISP=SHR
//STEP1     EXEC PGM=EXAMPLE
//RECIN     DD    *
DATA RECORD 1
DATA RECORD 2
//RECOUT    DD    DSN=RSM99.TEST.OUTREC,DISP=(,CATLG,DELETE),
//          SPACE=(TRK,(1,1))
//REPOUT    DD    SYSOUT=*
//
```

# DATA DIVISION (FILE SECTION, WS SECTION)



**FILE SECTION**

- defines all external data used
  - input files
  - output files
  - output reports

Inline data

Input files

Output files

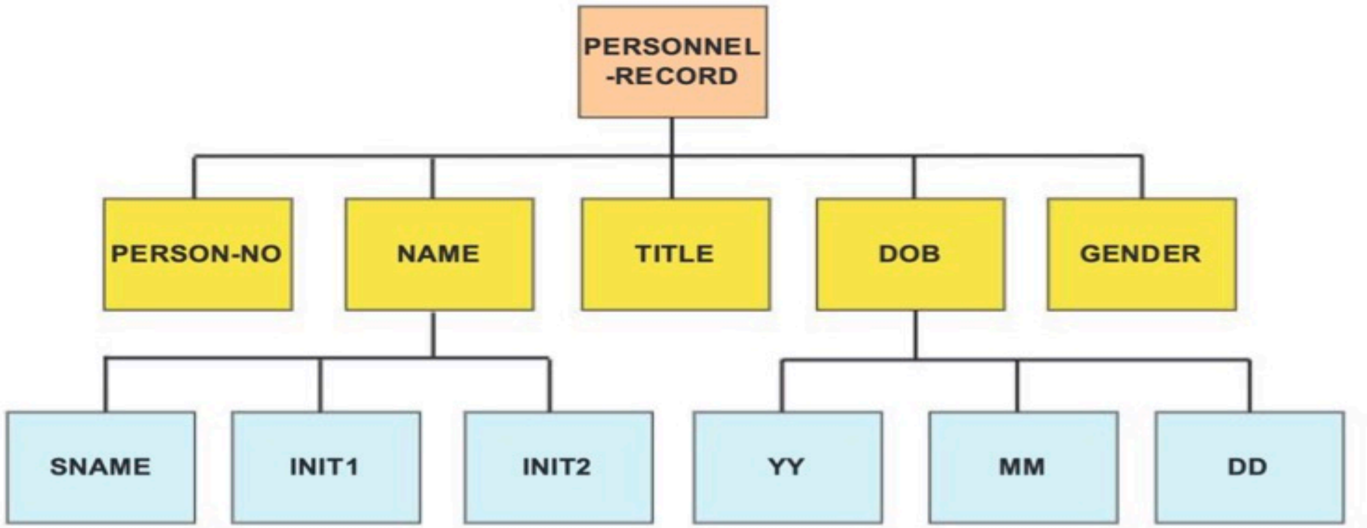
Reports

**WORKING-STORAGE SECTION**

- defines any internal data variables
  - working storage areas

Transient data

# Data Hierarchy sample



PERSONNEL -RECORD

PERSON-NO

NAME

TITLE

DOB

GENDER

SNAME

INIT1

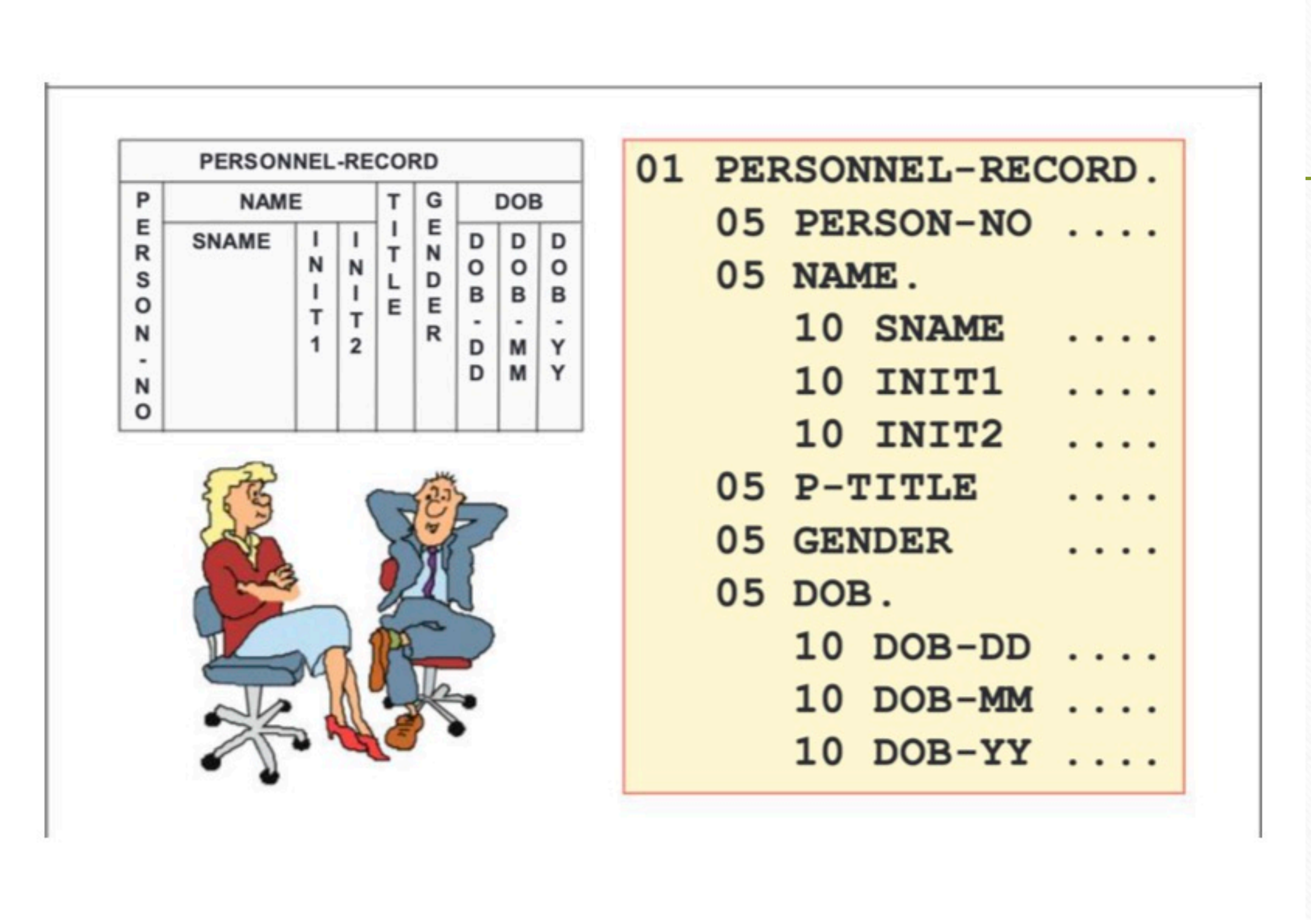
INIT2

YY

MM

DD

# Proper way to code the data elements (indentation)



| PERSONNEL-RECORD | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PERSON-NO | NAME | | | TITLE | GENDER | DOB | | |
| | SNAME | INIT1 | INIT2 | | | DOB-DD | DOB-MM | DOB-YY |

```
01 PERSONNEL-RECORD.
   05 PERSON-NO  ....
   05 NAME.
      10 SNAME   ....
      10 INIT1   ....
      10 INIT2   ....
   05 P-TITLE    ....
   05 GENDER     ....
   05 DOB.
      10 DOB-DD  ....
      10 DOB-MM  ....
      10 DOB-YY  ....
```

# File definitions in COBOL

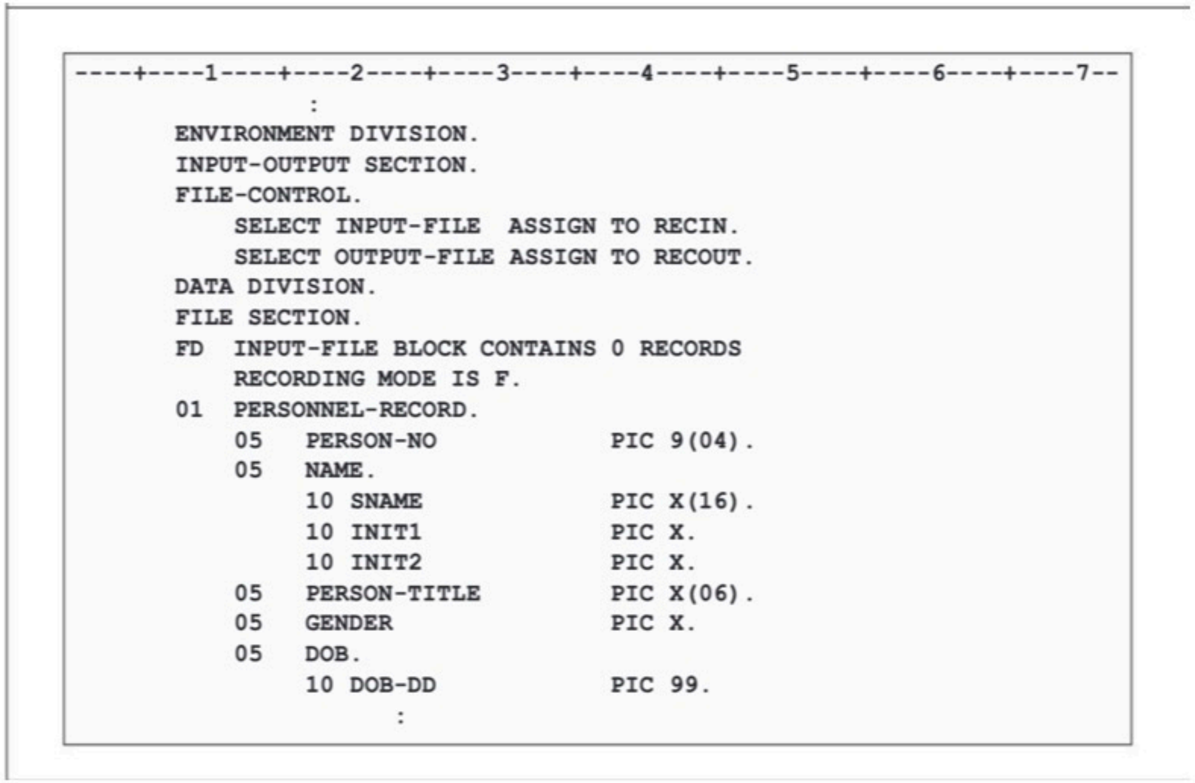
Note that the ENVIRONMENT DIVISION has the INPUT-OUTPUT SECTION, under which, we have the FILE-CONTROL paragraph, where we define the:

SELECT **internal-name** ASSIGN TO **dd-name** for all program files.

In addition, the DATA DIVISION contains the FILE SECTION, in which we code the corresponding **FD and associated record**, for the files (matching the SELECT internal names of the ENVIRONMENT DIVISION

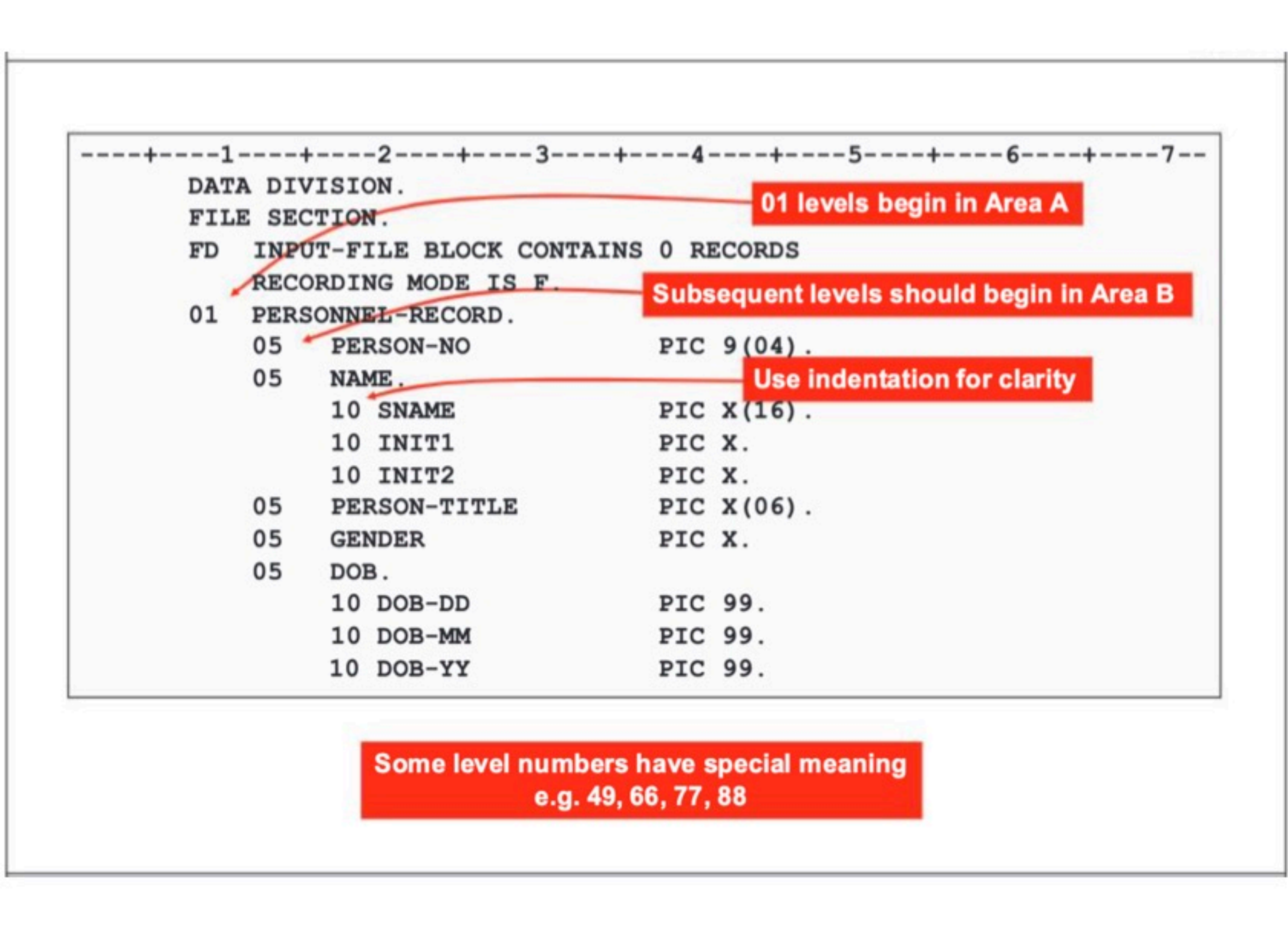
The next slide provides an example.

# Coding the file record (FILE SECTION)



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
                :
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE  ASSIGN TO RECIN.
           SELECT OUTPUT-FILE ASSIGN TO RECOUT.
       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE BLOCK CONTAINS 0 RECORDS
           RECORDING MODE IS F.
       01  PERSONNEL-RECORD.
           05   PERSON-NO             PIC 9(04).
           05   NAME.
                10 SNAME              PIC X(16).
                10 INIT1              PIC X.
                10 INIT2              PIC X.
           05   PERSON-TITLE          PIC X(06).
           05   GENDER                PIC X.
           05   DOB.
                10 DOB-DD             PIC 99.
                      :
```

# Mandatory coding columns



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE BLOCK CONTAINS 0 RECORDS
           RECORDING MODE IS F.
       01  PERSONNEL-RECORD.
           05   PERSON-NO            PIC 9(04).
           05   NAME.
                10 SNAME             PIC X(16).
                10 INIT1             PIC X.
                10 INIT2             PIC X.
           05   PERSON-TITLE         PIC X(06).
           05   GENDER               PIC X.
           05   DOB.
                10 DOB-DD            PIC 99.
                10 DOB-MM            PIC 99.
                10 DOB-YY            PIC 99.
```

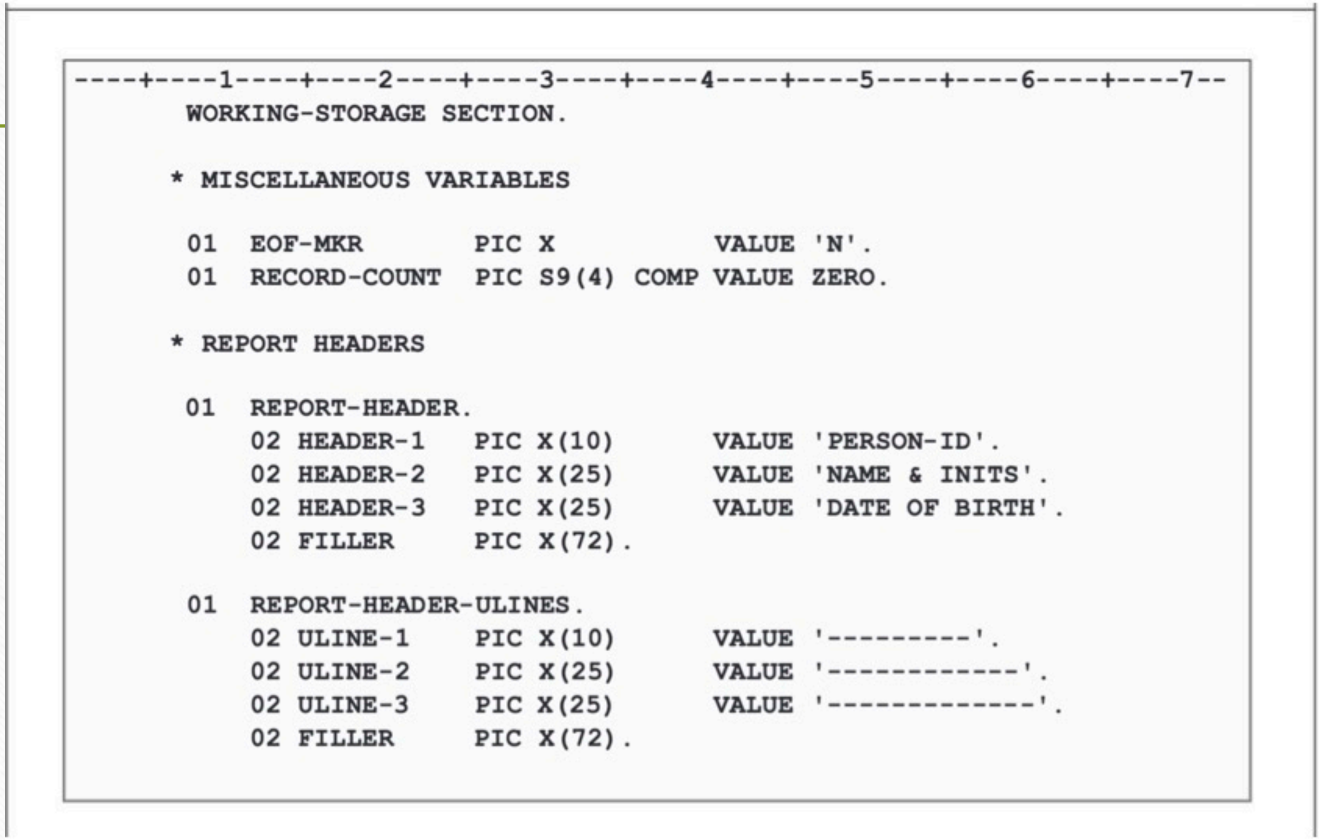
01 levels begin in Area A

Subsequent levels should begin in Area B

Use indentation for clarity

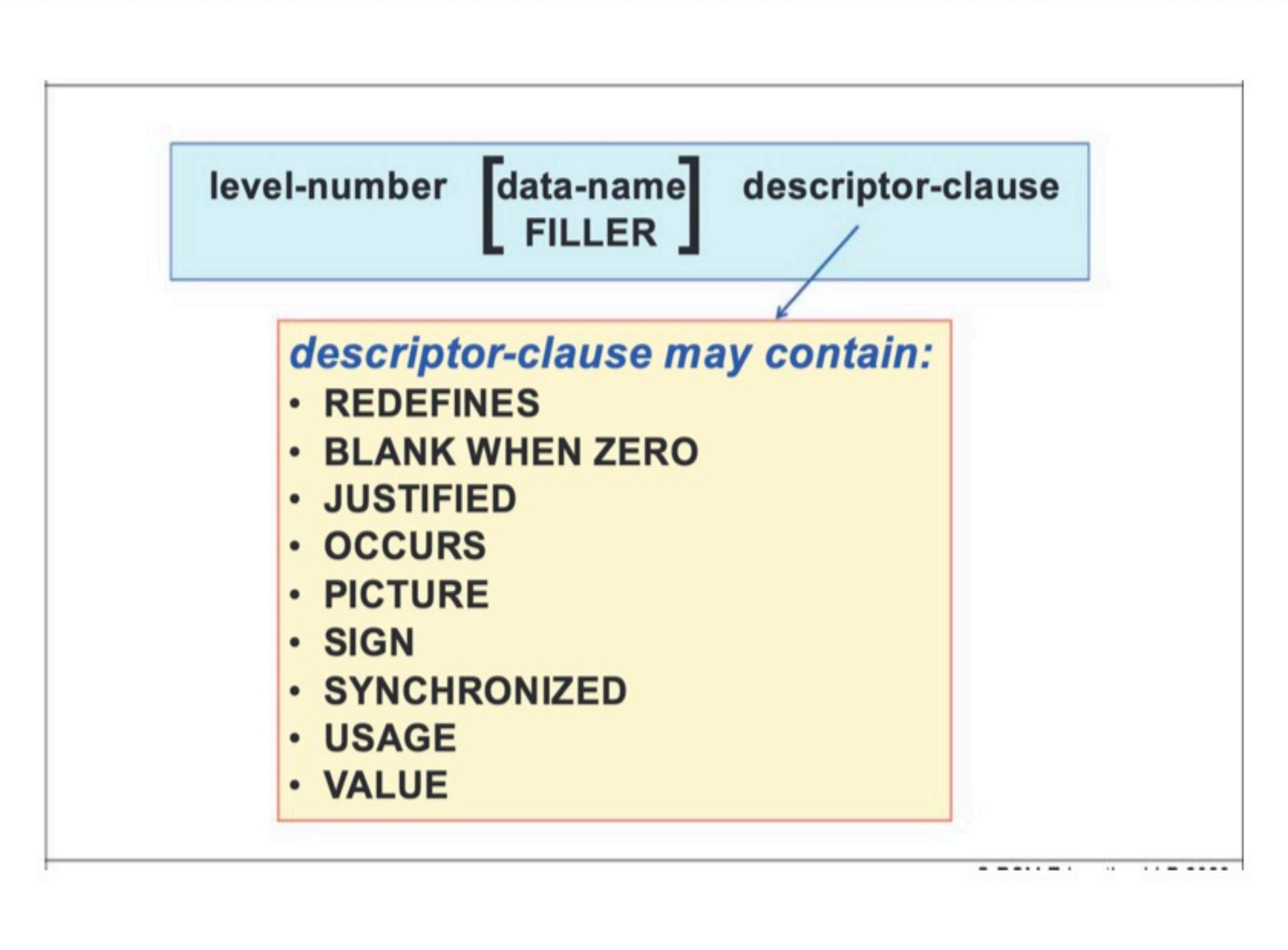
Some level numbers have special meaning
e.g. 49, 66, 77, 88

# Data Division / Working-storage section



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       WORKING-STORAGE SECTION.

      * MISCELLANEOUS VARIABLES

       01   EOF-MKR       PIC X          VALUE 'N'.
       01   RECORD-COUNT  PIC S9(4) COMP VALUE ZERO.

      * REPORT HEADERS

       01   REPORT-HEADER.
            02 HEADER-1   PIC X(10)      VALUE 'PERSON-ID'.
            02 HEADER-2   PIC X(25)      VALUE 'NAME & INITS'.
            02 HEADER-3   PIC X(25)      VALUE 'DATE OF BIRTH'.
            02 FILLER     PIC X(72).

       01   REPORT-HEADER-ULINES.
            02 ULINE-1    PIC X(10)      VALUE '---------'.
            02 ULINE-2    PIC X(25)      VALUE '------------'.
            02 ULINE-3    PIC X(25)      VALUE '-------------'.
            02 FILLER     PIC X(72).
```

# General rules for variable definition

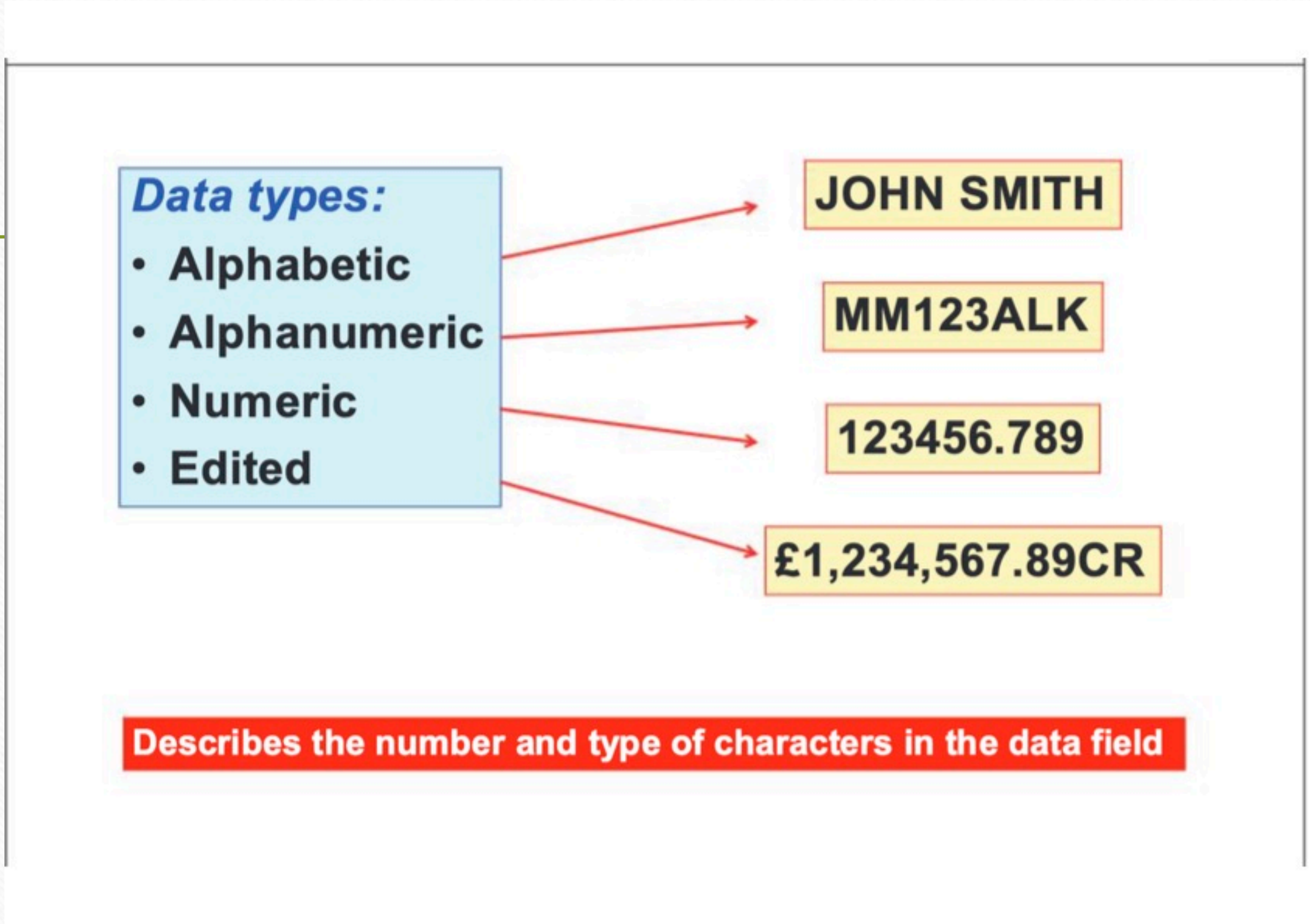


level-number [data-name / FILLER] descriptor-clause

*descriptor-clause may contain:*

- REDEFINES
- BLANK WHEN ZERO
- JUSTIFIED
- OCCURS
- PICTURE
- SIGN
- SYNCHRONIZED
- USAGE
- VALUE

# Exaples for data type names and values



*Data types:*

- Alphabetic
- Alphanumeric
- Numeric
- Edited

JOHN SMITH

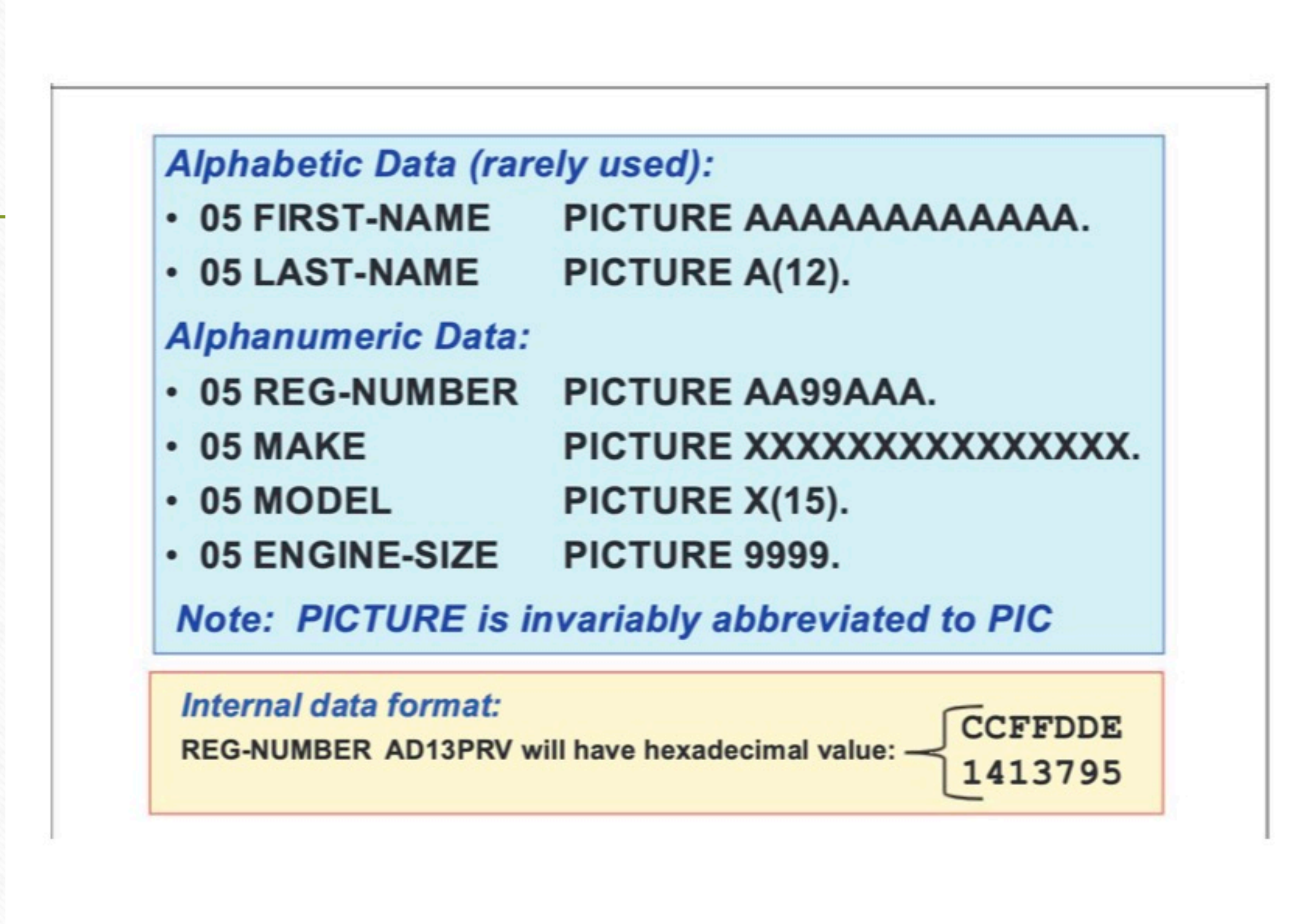
MM123ALK

123456.789

£1,234,567.89CR

Describes the number and type of characters in the data field

# In code definition



*Alphabetic Data (rarely used):*

- 05 FIRST-NAME PICTURE AAAAAAAAAAAA.
- 05 LAST-NAME PICTURE A(12).

*Alphanumeric Data:*

- 05 REG-NUMBER PICTURE AA99AAA.
- 05 MAKE PICTURE XXXXXXXXXXXXXXX.
- 05 MODEL PICTURE X(15).
- 05 ENGINE-SIZE PICTURE 9999.

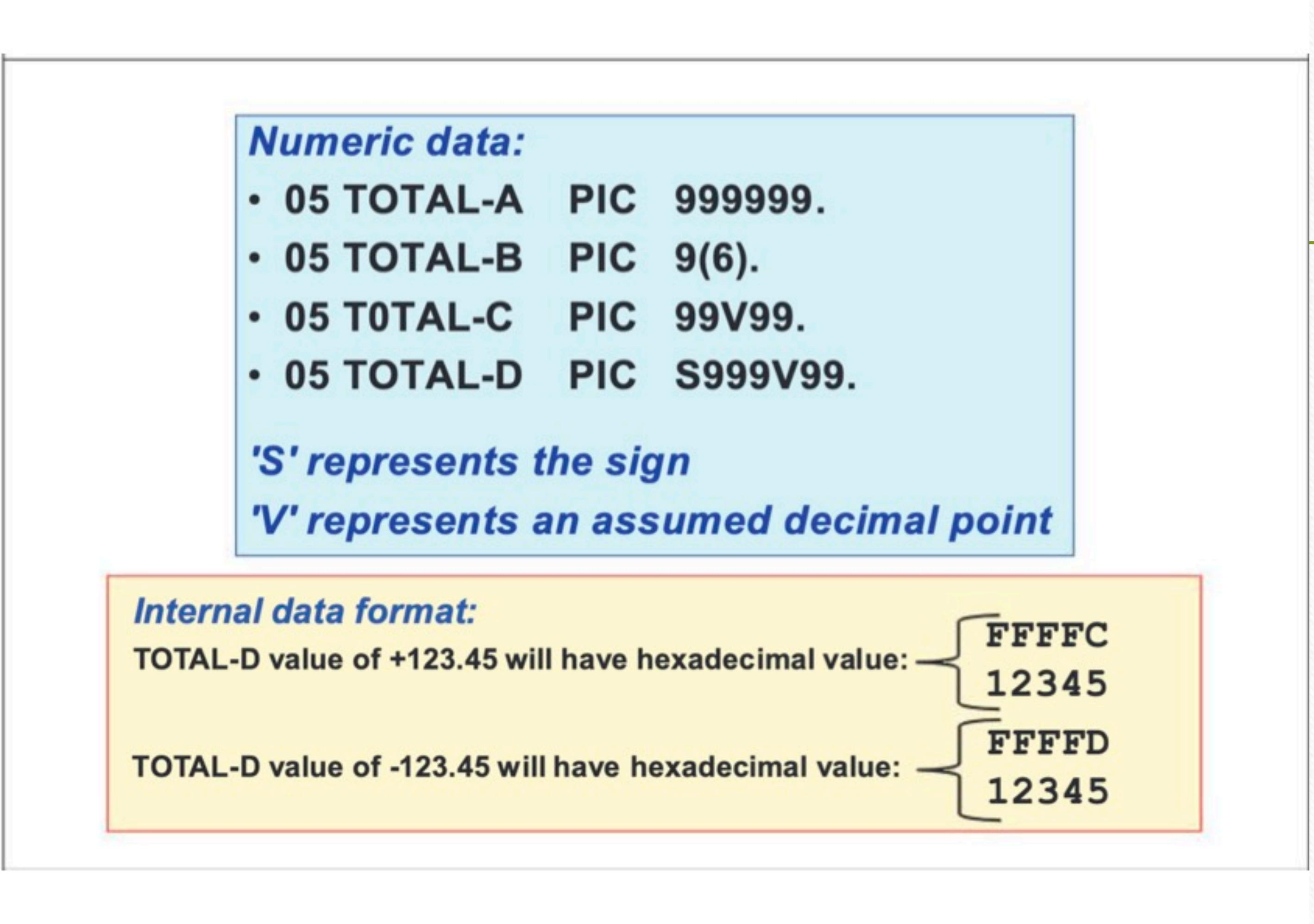
*Note: PICTURE is invariably abbreviated to PIC*

*Internal data format:*

REG-NUMBER AD13PRV will have hexadecimal value:

```
CCFFDDE
1413795
```

# Decimal point assumed and sign



*Numeric data:*

```
• 05 TOTAL-A   PIC   999999.
• 05 TOTAL-B   PIC   9(6).
• 05 T0TAL-C   PIC   99V99.
• 05 TOTAL-D   PIC   S999V99.
```

*'S' represents the sign*

*'V' represents an assumed decimal point*

*Internal data format:*

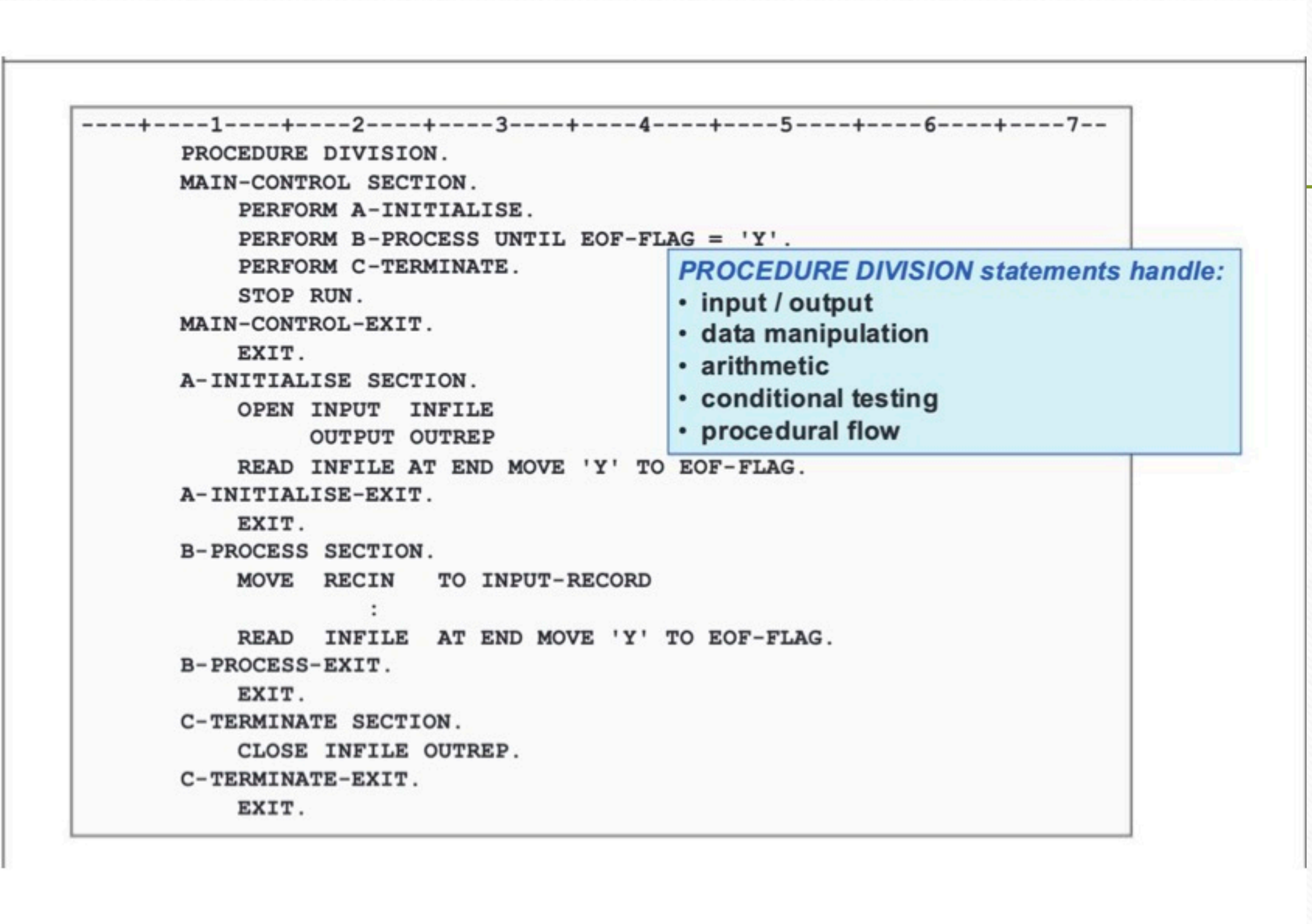
TOTAL-D value of +123.45 will have hexadecimal value:

```
FFFFC
12345
```

TOTAL-D value of -123.45 will have hexadecimal value:

```
FFFFD
12345
```

# PROCEDURE DIVISION (program logic)

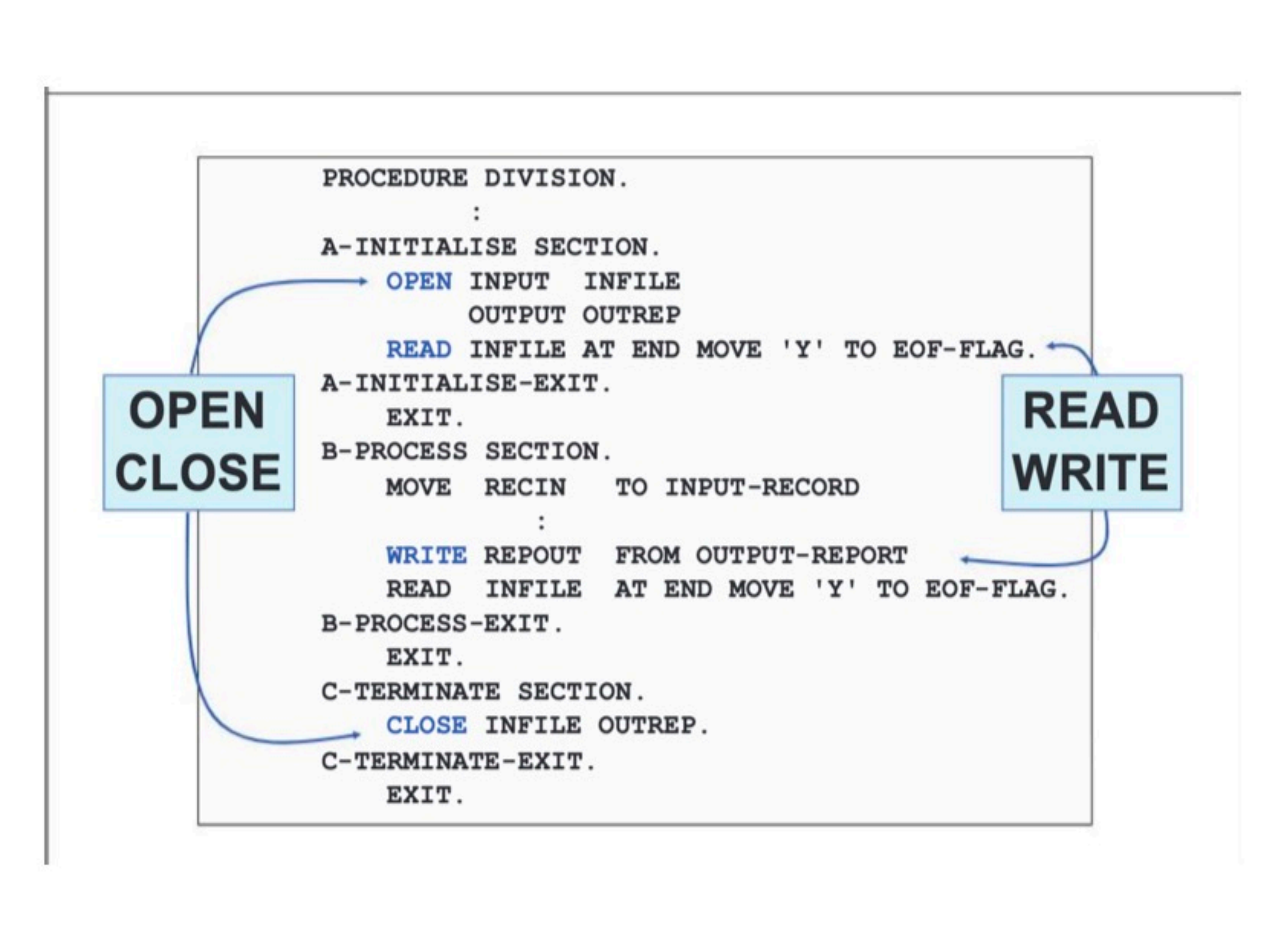


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
       MAIN-CONTROL SECTION.
           PERFORM A-INITIALISE.
           PERFORM B-PROCESS UNTIL EOF-FLAG = 'Y'.
           PERFORM C-TERMINATE.
           STOP RUN.
       MAIN-CONTROL-EXIT.
           EXIT.
       A-INITIALISE SECTION.
           OPEN INPUT  INFILE
                OUTPUT OUTREP
           READ INFILE AT END MOVE 'Y' TO EOF-FLAG.
       A-INITIALISE-EXIT.
           EXIT.
       B-PROCESS SECTION.
           MOVE  RECIN   TO INPUT-RECORD
                    :
           READ  INFILE  AT END MOVE 'Y' TO EOF-FLAG.
       B-PROCESS-EXIT.
           EXIT.
       C-TERMINATE SECTION.
           CLOSE INFILE OUTREP.
       C-TERMINATE-EXIT.
           EXIT.
```

***PROCEDURE DIVISION statements handle:***

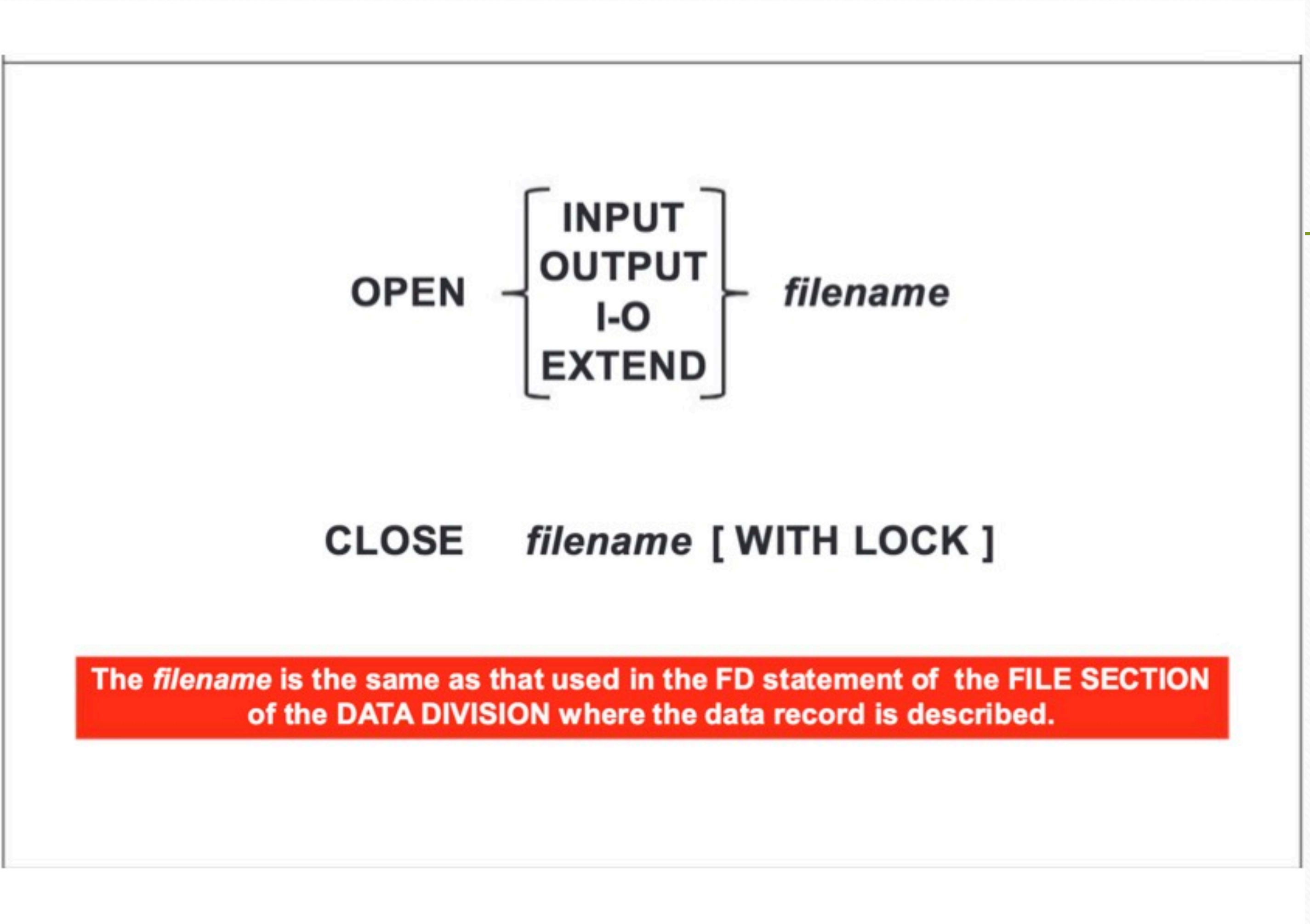
- **input / output**
- **data manipulation**
- **arithmetic**
- **conditional testing**
- **procedural flow**

# File processing



```
PROCEDURE DIVISION.
         :
A-INITIALISE SECTION.
    OPEN INPUT  INFILE
         OUTPUT OUTREP
    READ INFILE AT END MOVE 'Y' TO EOF-FLAG.
A-INITIALISE-EXIT.
    EXIT.
B-PROCESS SECTION.
    MOVE  RECIN   TO INPUT-RECORD
             :
    WRITE REPOUT  FROM OUTPUT-REPORT
    READ  INFILE  AT END MOVE 'Y' TO EOF-FLAG.
B-PROCESS-EXIT.
    EXIT.
C-TERMINATE SECTION.
    CLOSE INFILE OUTREP.
C-TERMINATE-EXIT.
    EXIT.
```

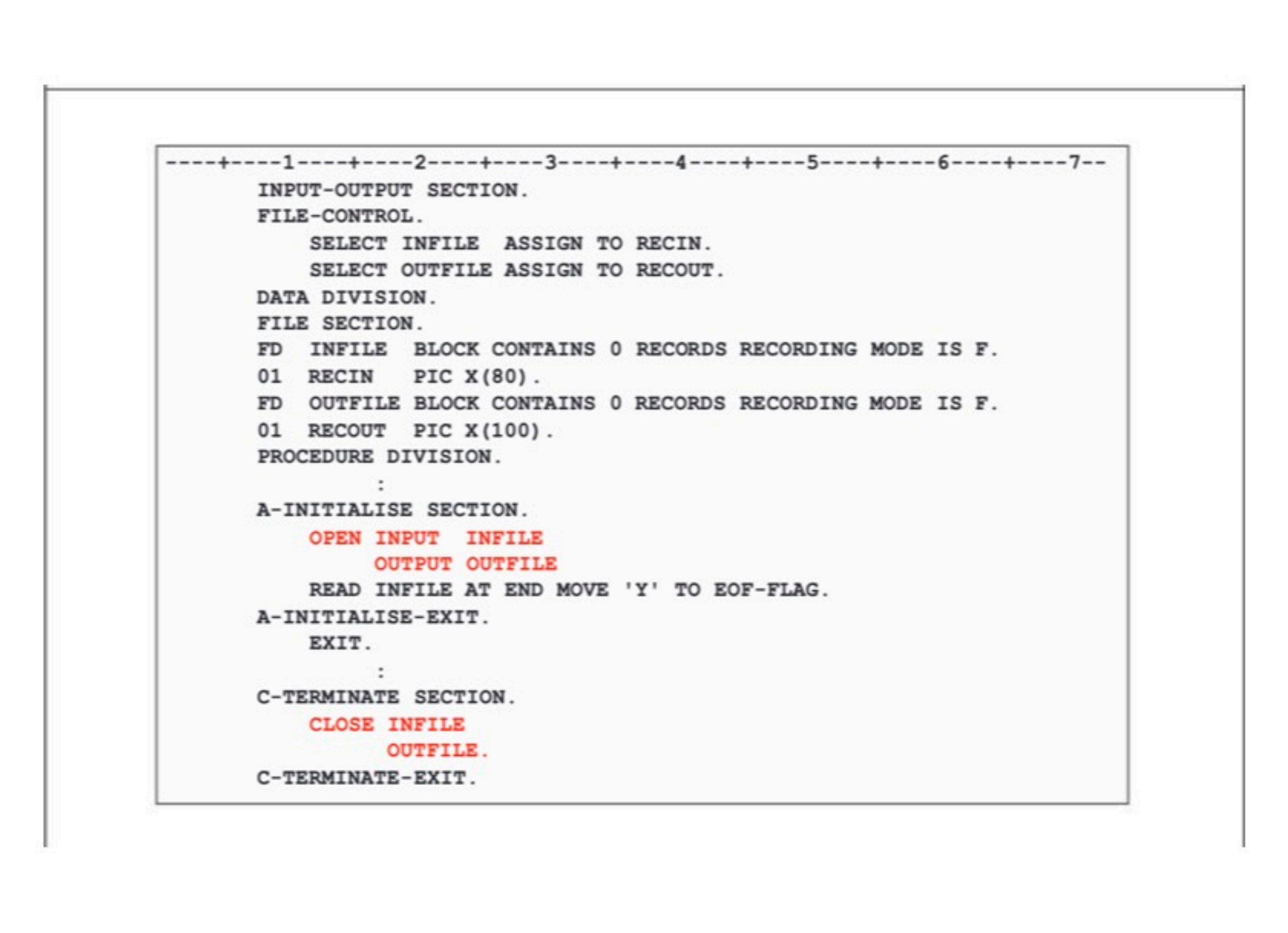
# Open/Close files



```
        ⎧ INPUT  ⎫
OPEN   ⎨ OUTPUT ⎬   filename
        ⎪ I-O    ⎪
        ⎩ EXTEND ⎭

CLOSE     filename  [ WITH LOCK ]
```

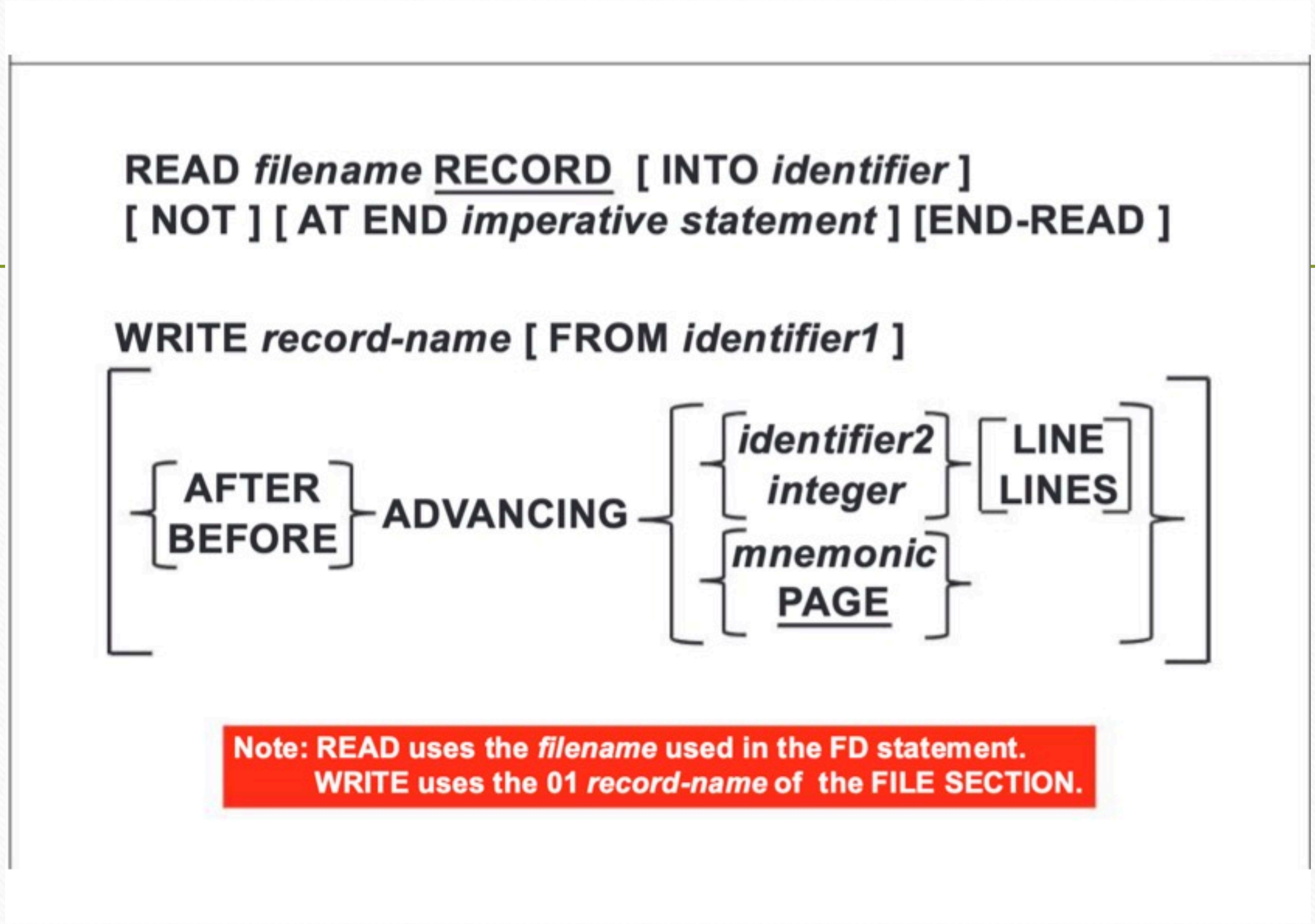
The *filename* is the same as that used in the FD statement of the FILE SECTION of the DATA DIVISION where the data record is described.

# File Open/Close in a program



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INFILE  ASSIGN TO RECIN.
           SELECT OUTFILE ASSIGN TO RECOUT.
       DATA DIVISION.
       FILE SECTION.
       FD  INFILE  BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
       01  RECIN   PIC X(80).
       FD  OUTFILE BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
       01  RECOUT  PIC X(100).
       PROCEDURE DIVISION.
                :
       A-INITIALISE SECTION.
           OPEN INPUT  INFILE
                OUTPUT OUTFILE
           READ INFILE AT END MOVE 'Y' TO EOF-FLAG.
       A-INITIALISE-EXIT.
           EXIT.
                :
       C-TERMINATE SECTION.
           CLOSE INFILE
                 OUTFILE.
       C-TERMINATE-EXIT.
```

# READ/WRITE options



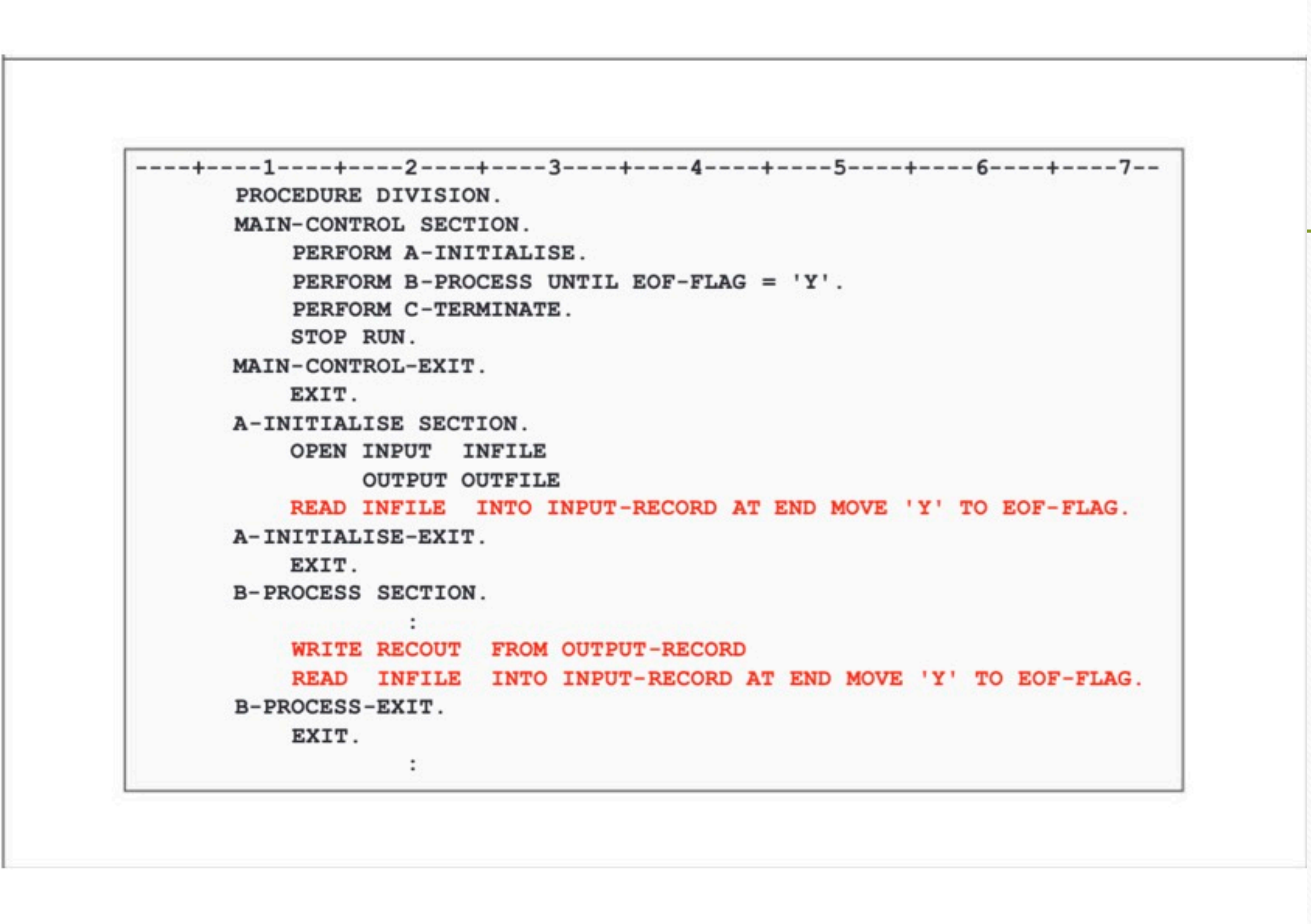
READ *filename* RECORD [ INTO *identifier* ]
[ NOT ] [ AT END *imperative statement* ] [END-READ ]

WRITE *record-name* [ FROM *identifier1* ]

$$\left[\begin{Bmatrix}\text{AFTER}\\ \text{BEFORE}\end{Bmatrix}\text{ADVANCING}\begin{Bmatrix}\begin{Bmatrix}\textit{identifier2}\\ \textit{integer}\end{Bmatrix}\begin{bmatrix}\text{LINE}\\ \text{LINES}\end{bmatrix}\\ \begin{Bmatrix}\textit{mnemonic}\\ \underline{\text{PAGE}}\end{Bmatrix}\end{Bmatrix}\right]$$

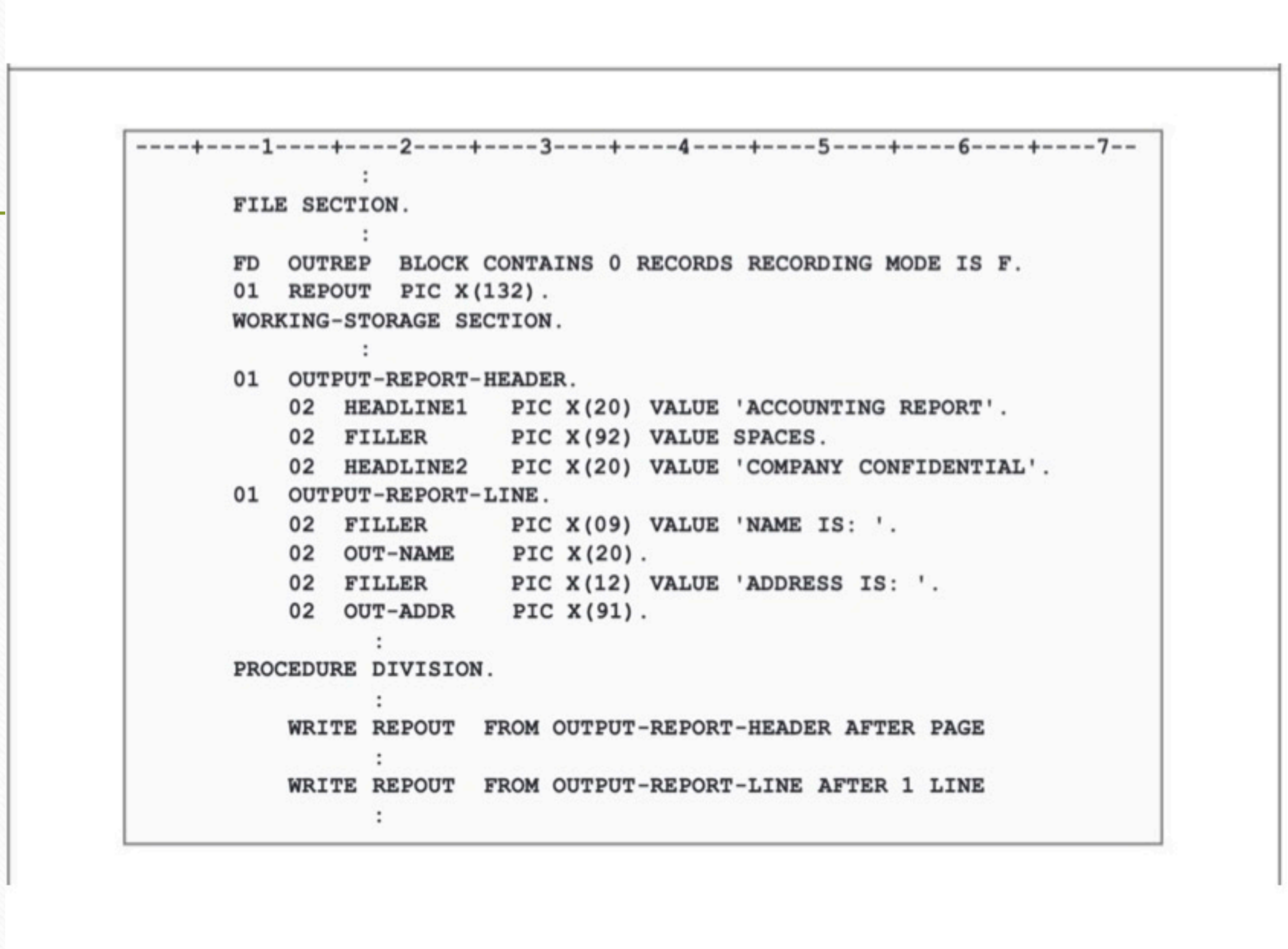
Note: READ uses the *filename* used in the FD statement.
WRITE uses the 01 *record-name* of the FILE SECTION.

# Reading and writing in a program



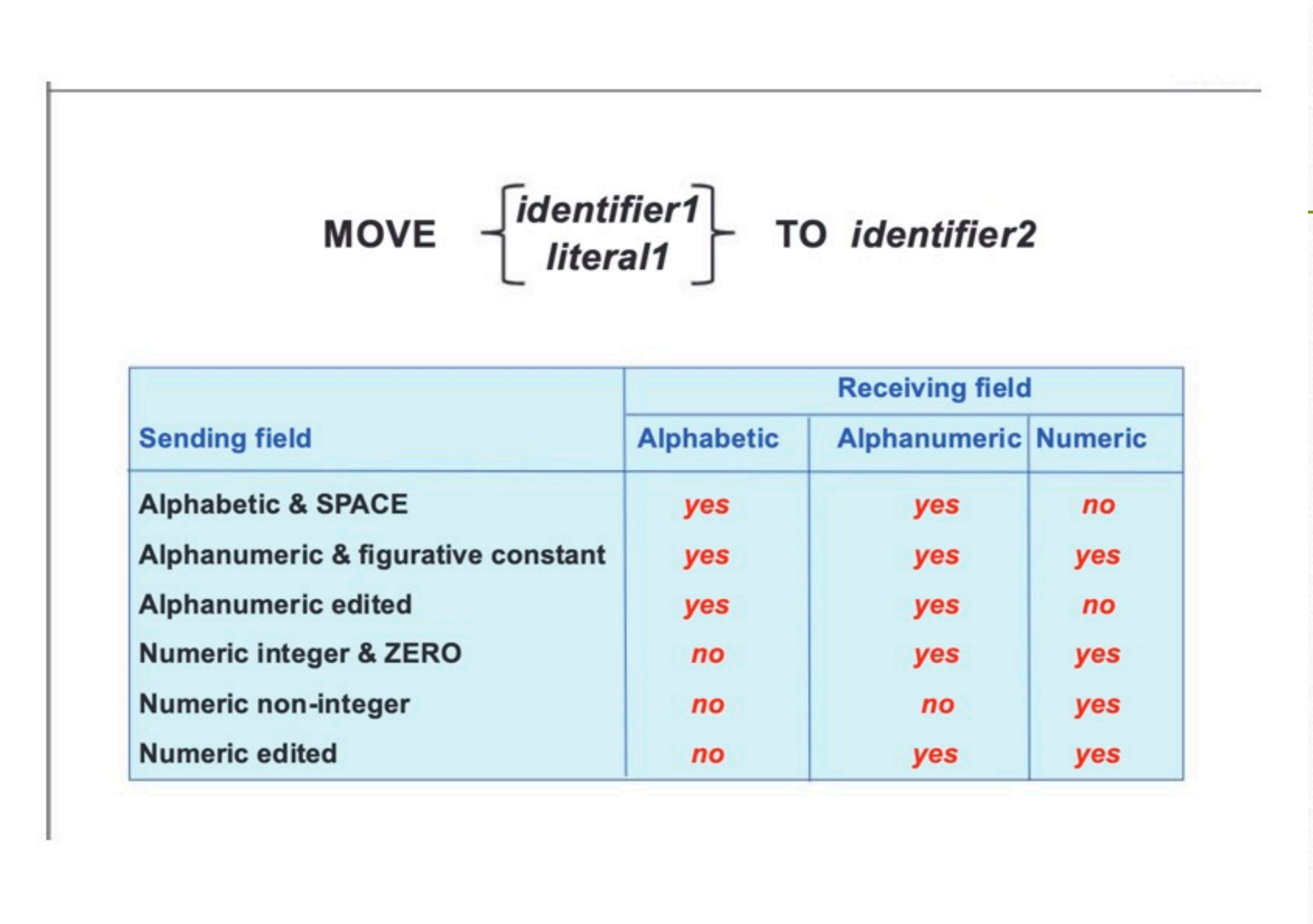
```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
       MAIN-CONTROL SECTION.
           PERFORM A-INITIALISE.
           PERFORM B-PROCESS UNTIL EOF-FLAG = 'Y'.
           PERFORM C-TERMINATE.
           STOP RUN.
       MAIN-CONTROL-EXIT.
           EXIT.
       A-INITIALISE SECTION.
           OPEN INPUT  INFILE
                OUTPUT OUTFILE
           READ INFILE  INTO INPUT-RECORD AT END MOVE 'Y' TO EOF-FLAG.
       A-INITIALISE-EXIT.
           EXIT.
       B-PROCESS SECTION.
                   :
           WRITE RECOUT   FROM OUTPUT-RECORD
           READ   INFILE  INTO INPUT-RECORD AT END MOVE 'Y' TO EOF-FLAG.
       B-PROCESS-EXIT.
           EXIT.
                   :
```

# Report header and lines



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
                :
       FILE SECTION.
                :
       FD  OUTREP  BLOCK CONTAINS 0 RECORDS RECORDING MODE IS F.
       01  REPOUT  PIC X(132).
       WORKING-STORAGE SECTION.
                :
       01  OUTPUT-REPORT-HEADER.
           02  HEADLINE1   PIC X(20) VALUE 'ACCOUNTING REPORT'.
           02  FILLER      PIC X(92) VALUE SPACES.
           02  HEADLINE2   PIC X(20) VALUE 'COMPANY CONFIDENTIAL'.
       01  OUTPUT-REPORT-LINE.
           02  FILLER      PIC X(09) VALUE 'NAME IS: '.
           02  OUT-NAME    PIC X(20).
           02  FILLER      PIC X(12) VALUE 'ADDRESS IS: '.
           02  OUT-ADDR    PIC X(91).
                 :
       PROCEDURE DIVISION.
                 :
           WRITE REPOUT  FROM OUTPUT-REPORT-HEADER AFTER PAGE
                 :
           WRITE REPOUT  FROM OUTPUT-REPORT-LINE AFTER 1 LINE
                 :
```

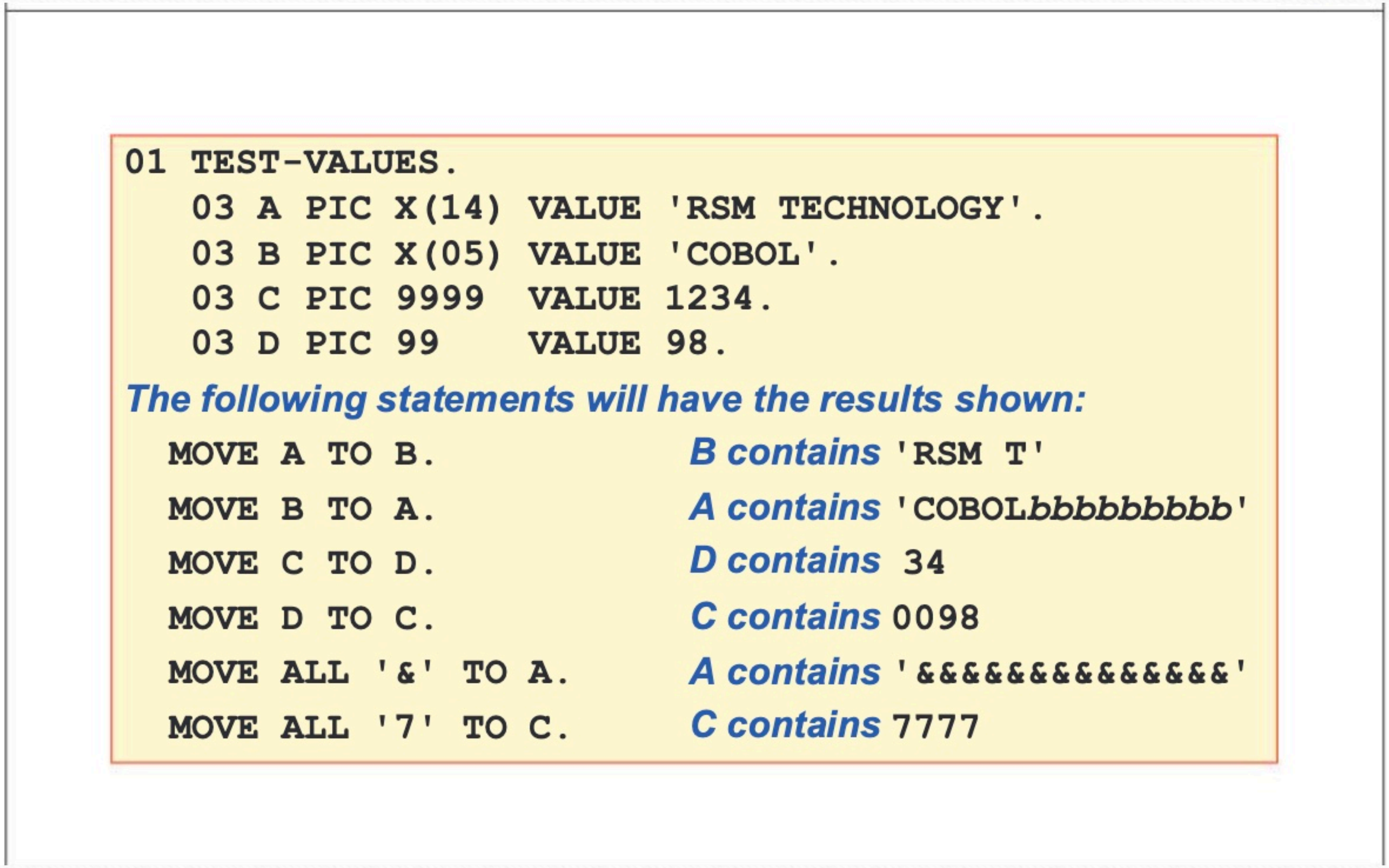
# The move statement



$$\textbf{MOVE} \begin{Bmatrix} \textit{identifier1} \\ \textit{literal1} \end{Bmatrix} \textbf{TO} \ \textit{identifier2}$$

| Sending field | Receiving field | | |
|---|---|---|---|
| | Alphabetic | Alphanumeric | Numeric |
| Alphabetic & SPACE | yes | yes | no |
| Alphanumeric & figurative constant | yes | yes | yes |
| Alphanumeric edited | yes | yes | no |
| Numeric integer & ZERO | no | yes | yes |
| Numeric non-integer | no | no | yes |
| Numeric edited | no | yes | yes |

# Simple MOVE examples

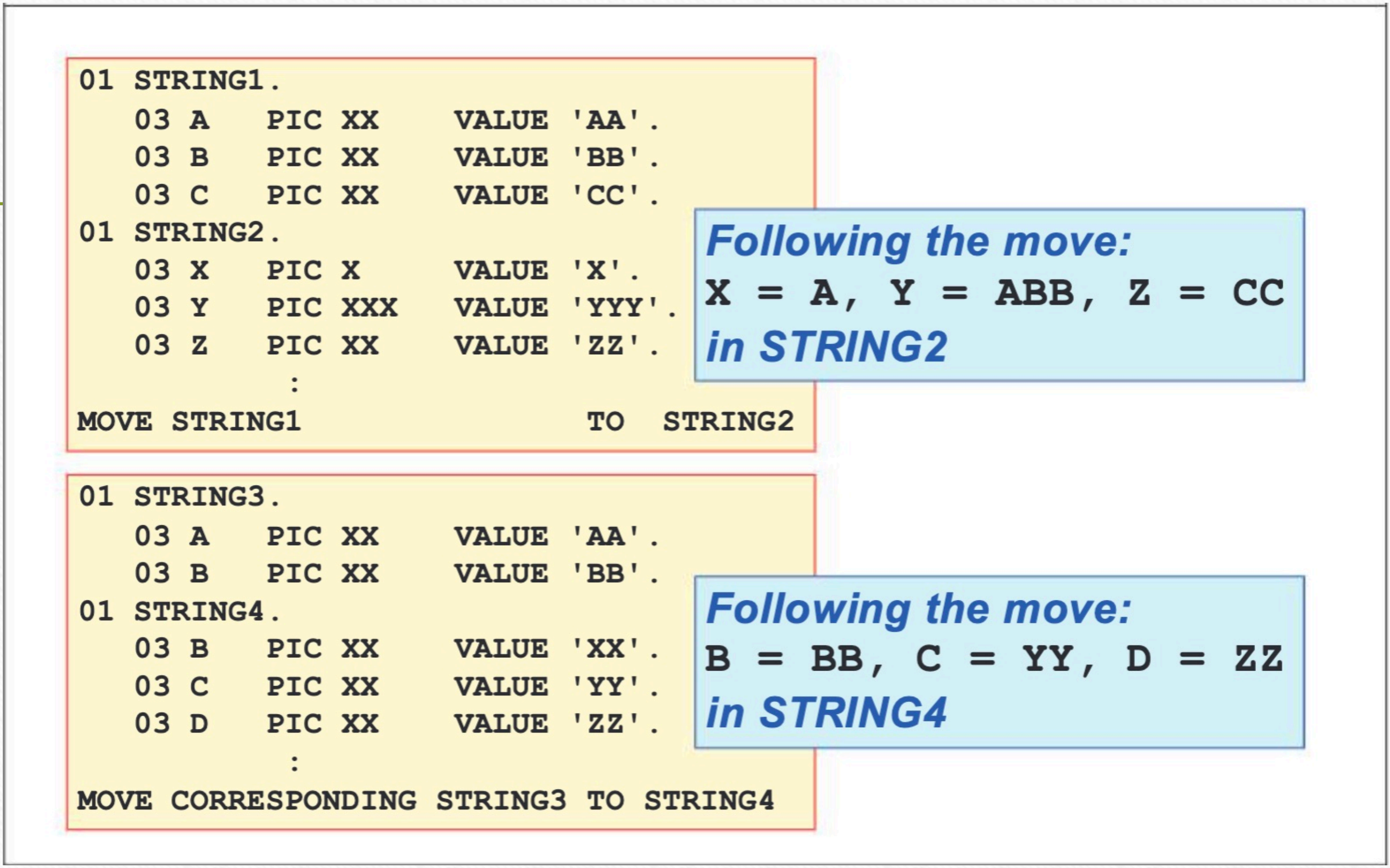


```
01 TEST-VALUES.
   03 A PIC X(14) VALUE 'RSM TECHNOLOGY'.
   03 B PIC X(05) VALUE 'COBOL'.
   03 C PIC 9999  VALUE 1234.
   03 D PIC 99    VALUE 98.
```

*The following statements will have the results shown:*

```
  MOVE A TO B.             B contains 'RSM T'
  MOVE B TO A.             A contains 'COBOLbbbbbbbbb'
  MOVE C TO D.             D contains  34
  MOVE D TO C.             C contains 0098
  MOVE ALL '&' TO A.       A contains '&&&&&&&&&&&&&&'
  MOVE ALL '7' TO C.       C contains 7777
```

# Examples of group MOVE and corresponding



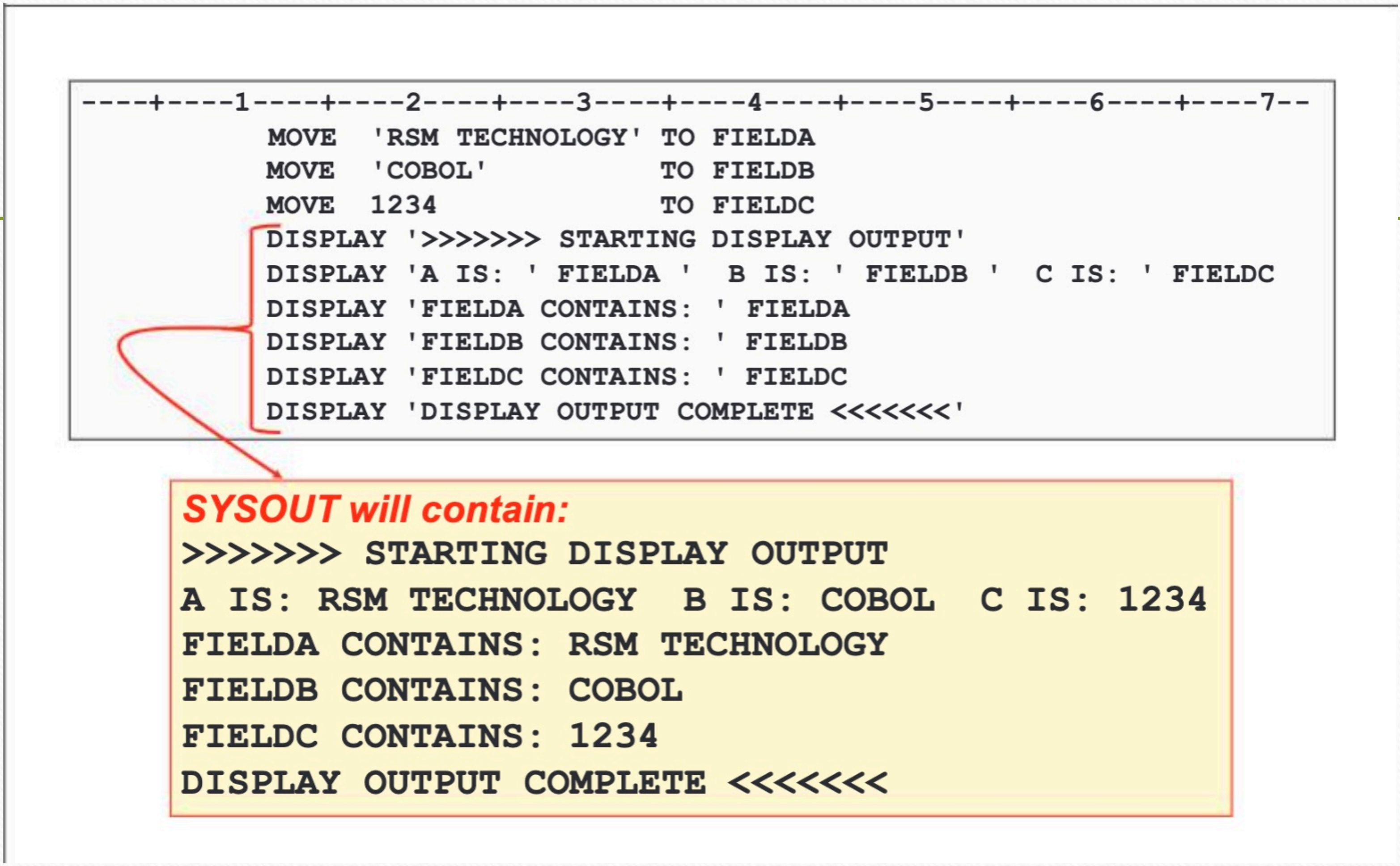
```
01 STRING1.
   03 A    PIC XX    VALUE 'AA'.
   03 B    PIC XX    VALUE 'BB'.
   03 C    PIC XX    VALUE 'CC'.
01 STRING2.
   03 X    PIC X     VALUE 'X'.
   03 Y    PIC XXX   VALUE 'YYY'.
   03 Z    PIC XX    VALUE 'ZZ'.
           :
MOVE STRING1               TO  STRING2
```

*Following the move:*
X = A, Y = ABB, Z = CC
*in STRING2*

```
01 STRING3.
   03 A    PIC XX    VALUE 'AA'.
   03 B    PIC XX    VALUE 'BB'.
01 STRING4.
   03 B    PIC XX    VALUE 'XX'.
   03 C    PIC XX    VALUE 'YY'.
   03 D    PIC XX    VALUE 'ZZ'.
           :
MOVE CORRESPONDING STRING3 TO STRING4
```

*Following the move:*
B = BB, C = YY, D = ZZ
*in STRING4*

# DISPLAY statement

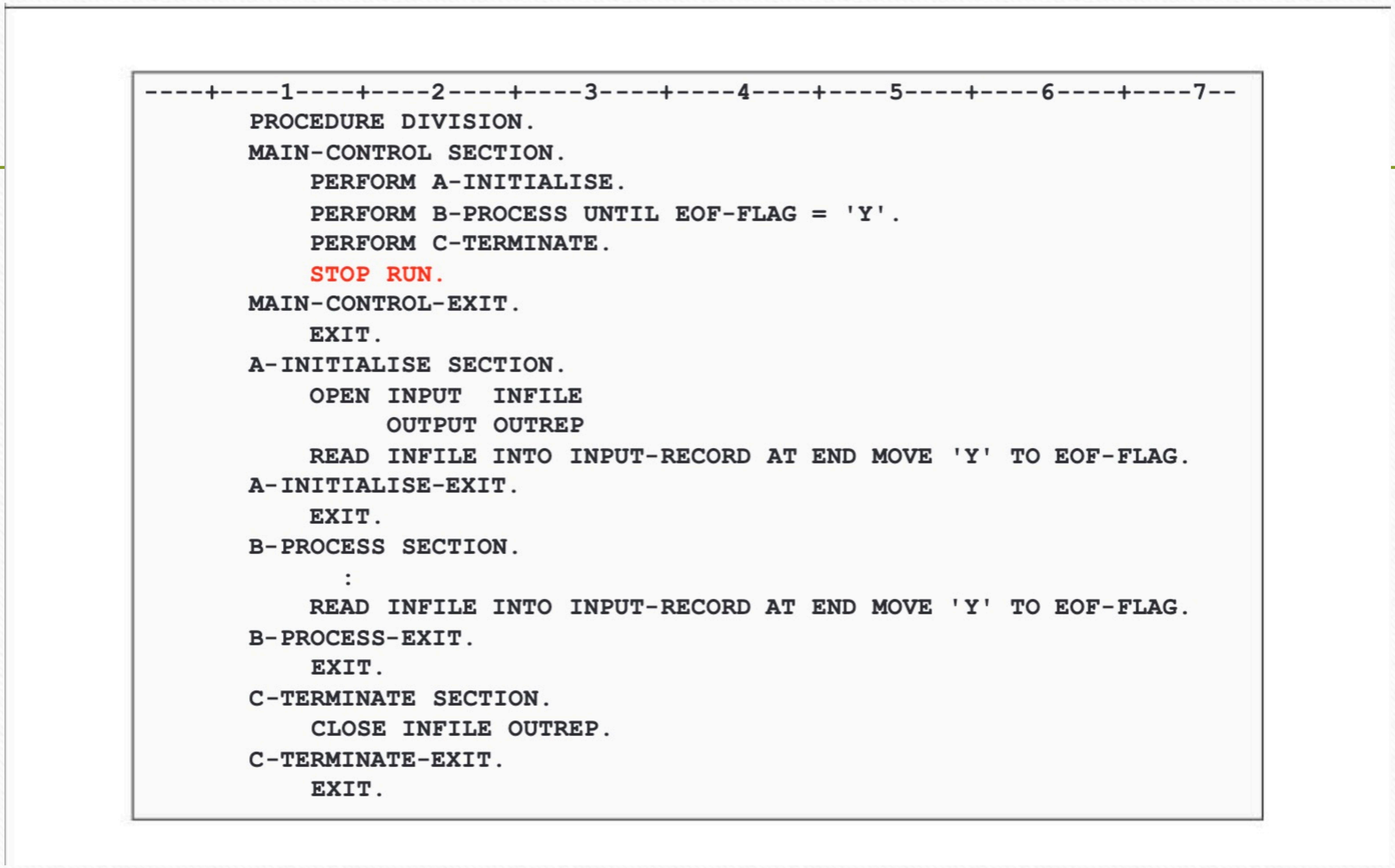


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
           MOVE  'RSM TECHNOLOGY' TO FIELDA
           MOVE  'COBOL'          TO FIELDB
           MOVE  1234             TO FIELDC
           DISPLAY '>>>>>>>> STARTING DISPLAY OUTPUT'
           DISPLAY 'A IS: ' FIELDA '  B IS: ' FIELDB '  C IS: ' FIELDC
           DISPLAY 'FIELDA CONTAINS: ' FIELDA
           DISPLAY 'FIELDB CONTAINS: ' FIELDB
           DISPLAY 'FIELDC CONTAINS: ' FIELDC
           DISPLAY 'DISPLAY OUTPUT COMPLETE <<<<<<<'
```

*SYSOUT will contain:*

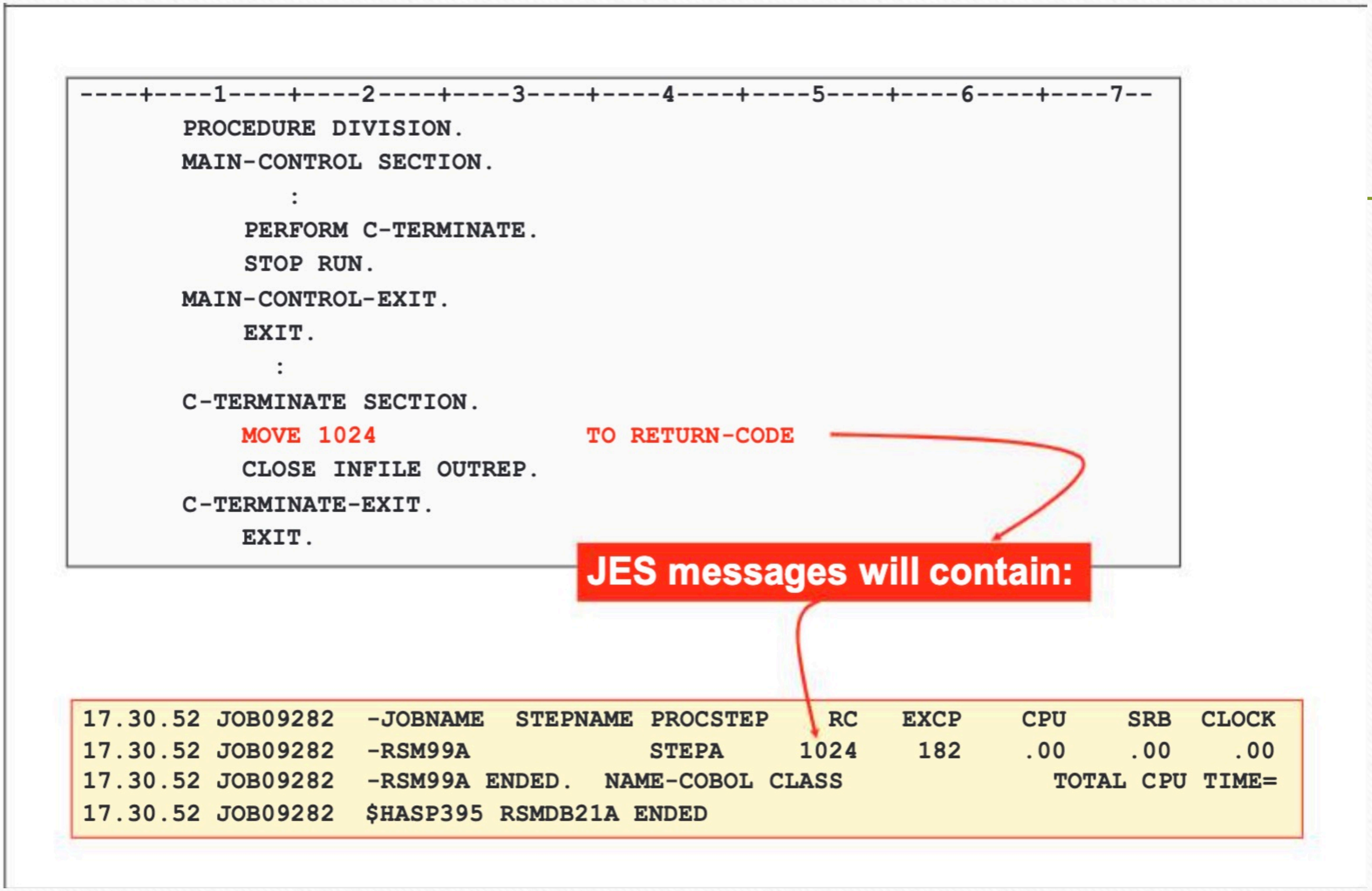
```
>>>>>>>> STARTING DISPLAY OUTPUT
A IS: RSM TECHNOLOGY  B IS: COBOL  C IS: 1234
FIELDA CONTAINS: RSM TECHNOLOGY
FIELDB CONTAINS: COBOL
FIELDC CONTAINS: 1234
DISPLAY OUTPUT COMPLETE <<<<<<<
```

# Terminating a program



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
       MAIN-CONTROL SECTION.
           PERFORM A-INITIALISE.
           PERFORM B-PROCESS UNTIL EOF-FLAG = 'Y'.
           PERFORM C-TERMINATE.
           STOP RUN.
       MAIN-CONTROL-EXIT.
           EXIT.
       A-INITIALISE SECTION.
           OPEN INPUT  INFILE
                OUTPUT OUTREP
           READ INFILE INTO INPUT-RECORD AT END MOVE 'Y' TO EOF-FLAG.
       A-INITIALISE-EXIT.
           EXIT.
       B-PROCESS SECTION.
             :
           READ INFILE INTO INPUT-RECORD AT END MOVE 'Y' TO EOF-FLAG.
       B-PROCESS-EXIT.
           EXIT.
       C-TERMINATE SECTION.
           CLOSE INFILE OUTREP.
       C-TERMINATE-EXIT.
           EXIT.
```

# RETURM-CODE and JCL

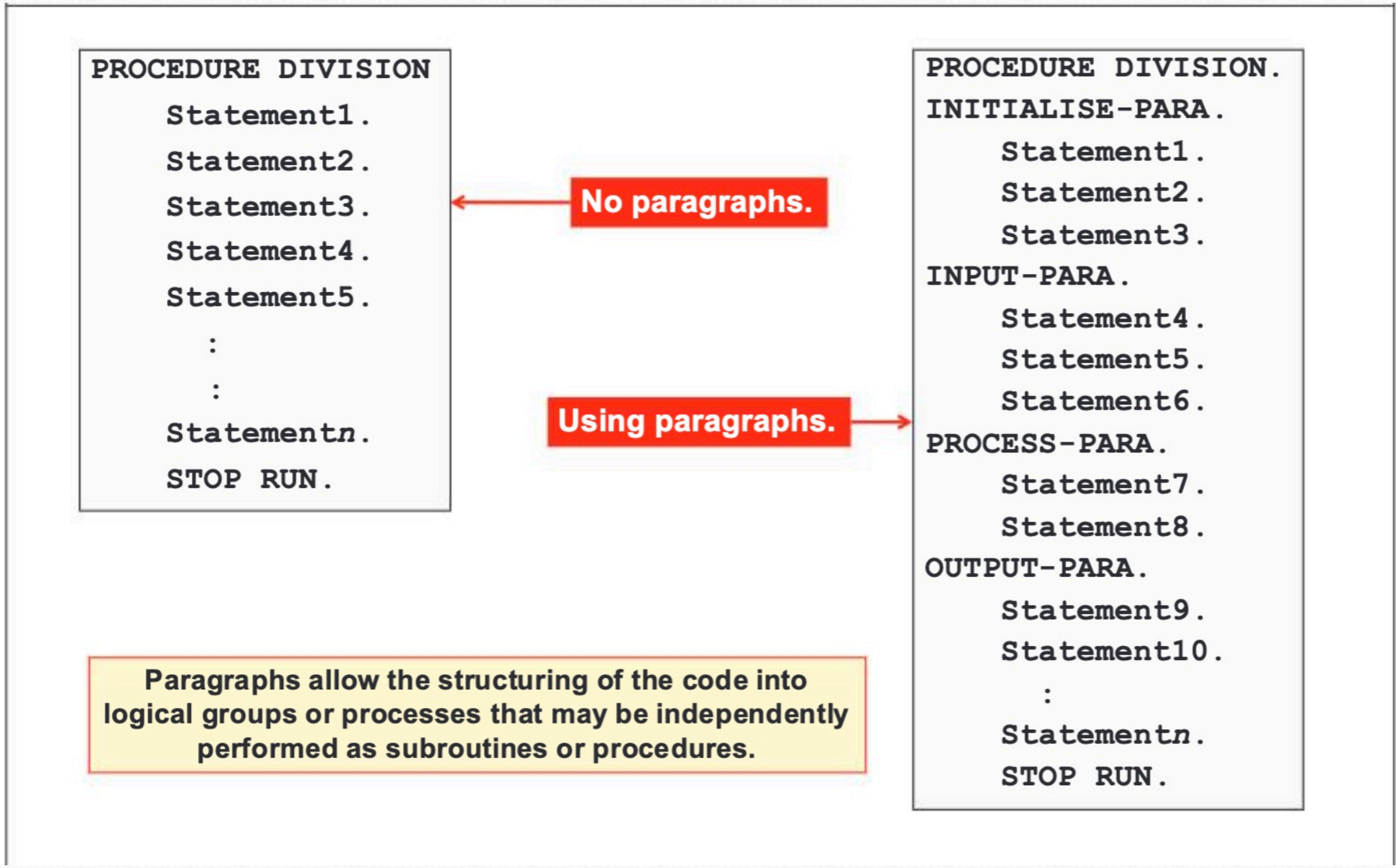


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
       MAIN-CONTROL SECTION.
              :
           PERFORM C-TERMINATE.
           STOP RUN.
       MAIN-CONTROL-EXIT.
           EXIT.
             :
       C-TERMINATE SECTION.
           MOVE 1024              TO RETURN-CODE
           CLOSE INFILE OUTREP.
       C-TERMINATE-EXIT.
           EXIT.
```

JES messages will contain:

```
17.30.52 JOB09282  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK
17.30.52 JOB09282  -RSM99A            STEPA     1024    182    .00    .00    .00
17.30.52 JOB09282  -RSM99A ENDED.  NAME-COBOL CLASS              TOTAL CPU TIME=
17.30.52 JOB09282  $HASP395 RSMDB21A ENDED
```

# Program paragraphs



```
PROCEDURE DIVISION
    Statement1.
    Statement2.
    Statement3.
    Statement4.
    Statement5.
       :
       :
    Statementn.
    STOP RUN.
```

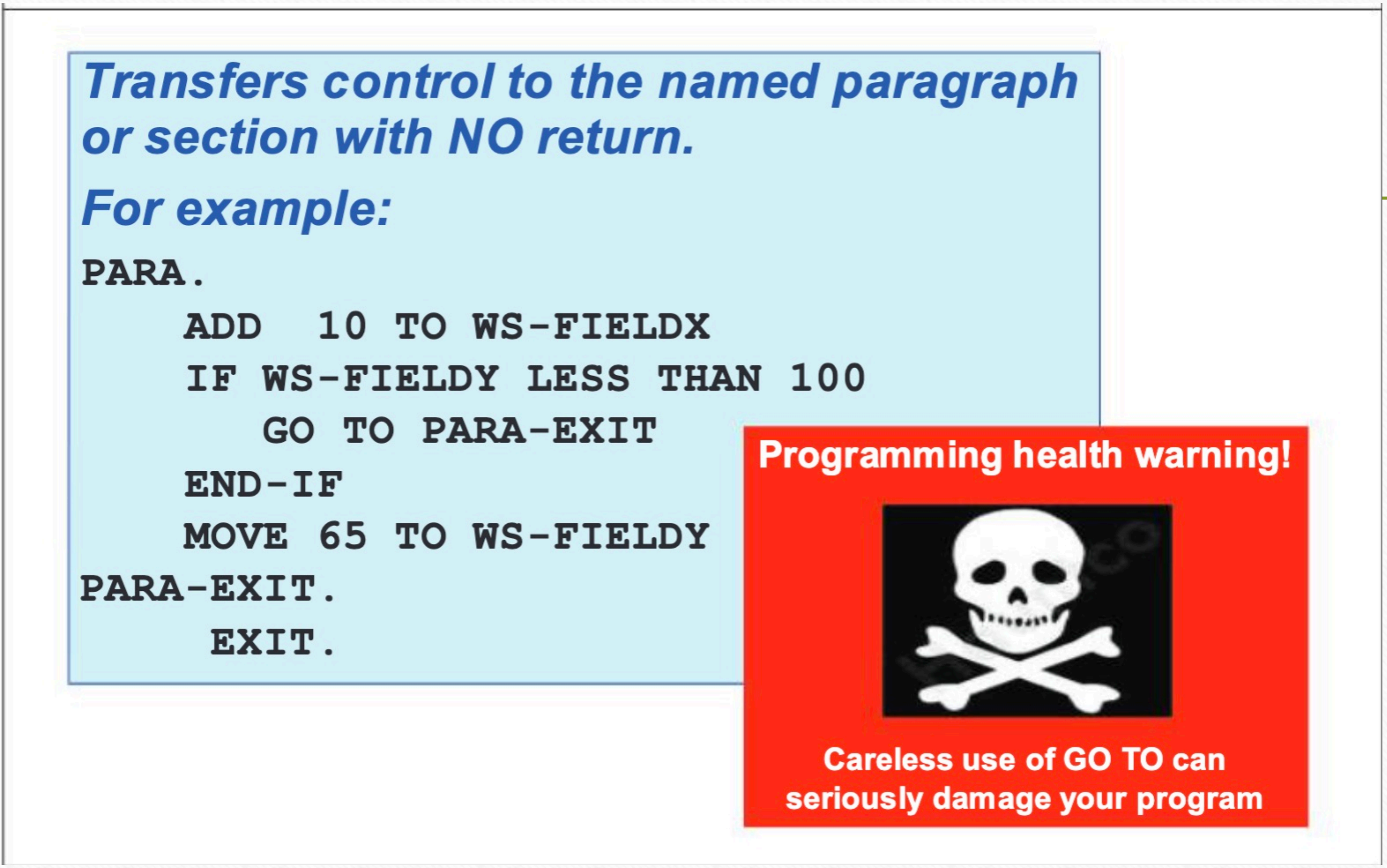
No paragraphs.

Using paragraphs.

```
PROCEDURE DIVISION.
INITIALISE-PARA.
    Statement1.
    Statement2.
    Statement3.
INPUT-PARA.
    Statement4.
    Statement5.
    Statement6.
PROCESS-PARA.
    Statement7.
    Statement8.
OUTPUT-PARA.
    Statement9.
    Statement10.
       :
    Statementn.
    STOP RUN.
```

Paragraphs allow the structuring of the code into logical groups or processes that may be independently performed as subroutines or procedures.

# GO TO statement (mostly discouraged)



***Transfers control to the named paragraph or section with NO return.***

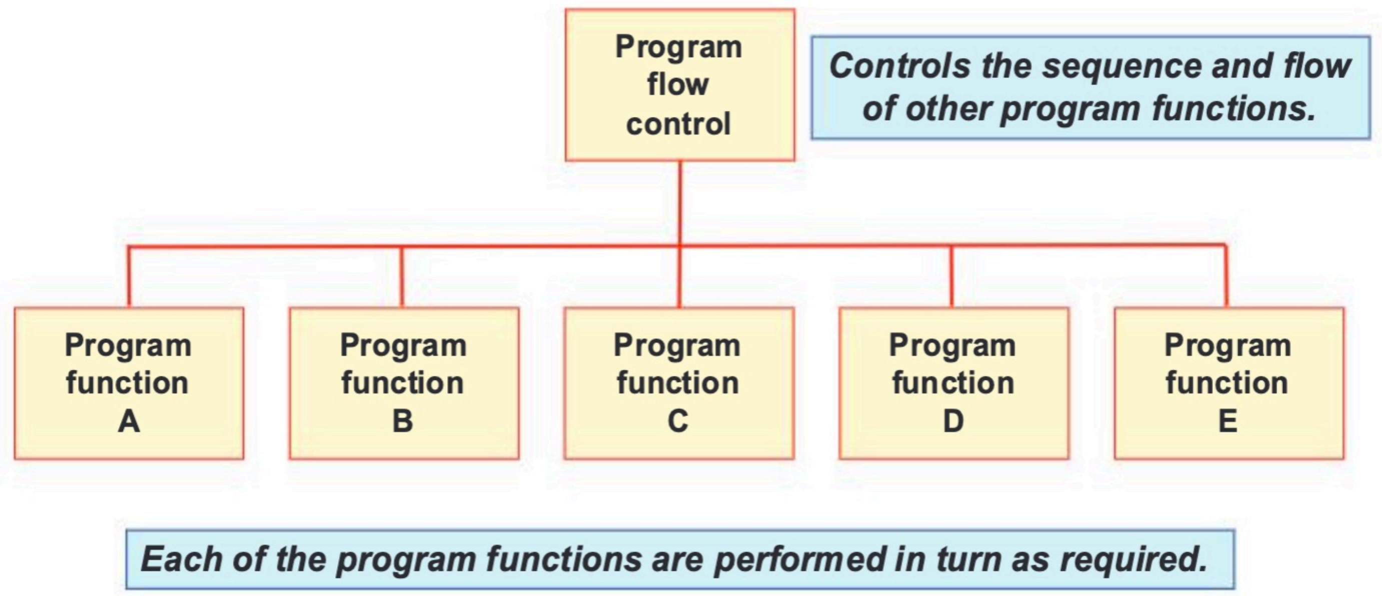
***For example:***

```
PARA.
    ADD  10 TO WS-FIELDX
    IF WS-FIELDY LESS THAN 100
       GO TO PARA-EXIT
    END-IF
    MOVE 65 TO WS-FIELDY
PARA-EXIT.
     EXIT.
```

**Programming health warning!**

**Careless use of GO TO can seriously damage your program**

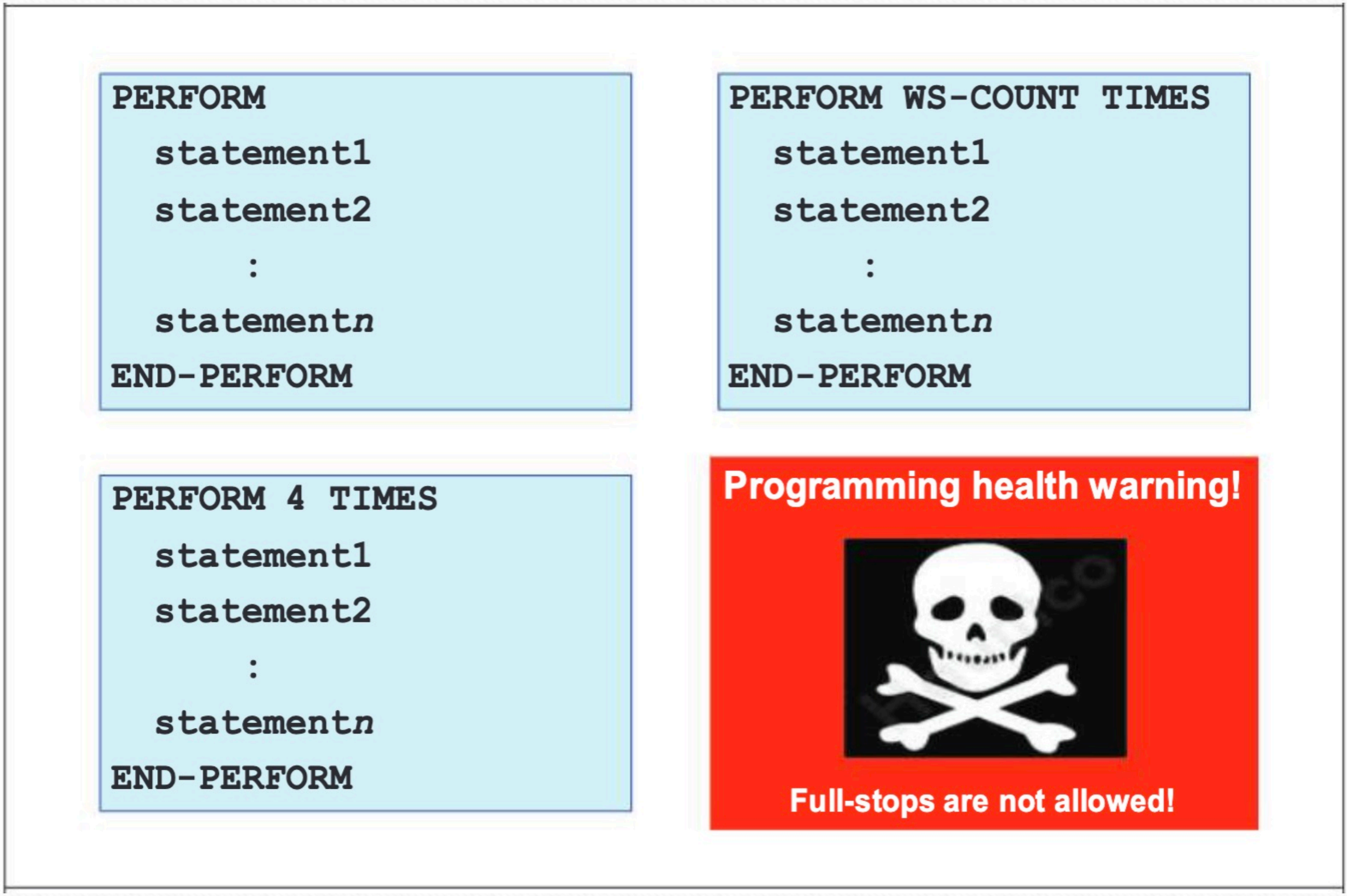
# Structured programming paradigm



*Controls the sequence and flow of other program functions.*

*Each of the program functions are performed in turn as required.*

# In-line PERFORM



```
PERFORM
  statement1
  statement2
      :
  statementn
END-PERFORM
```

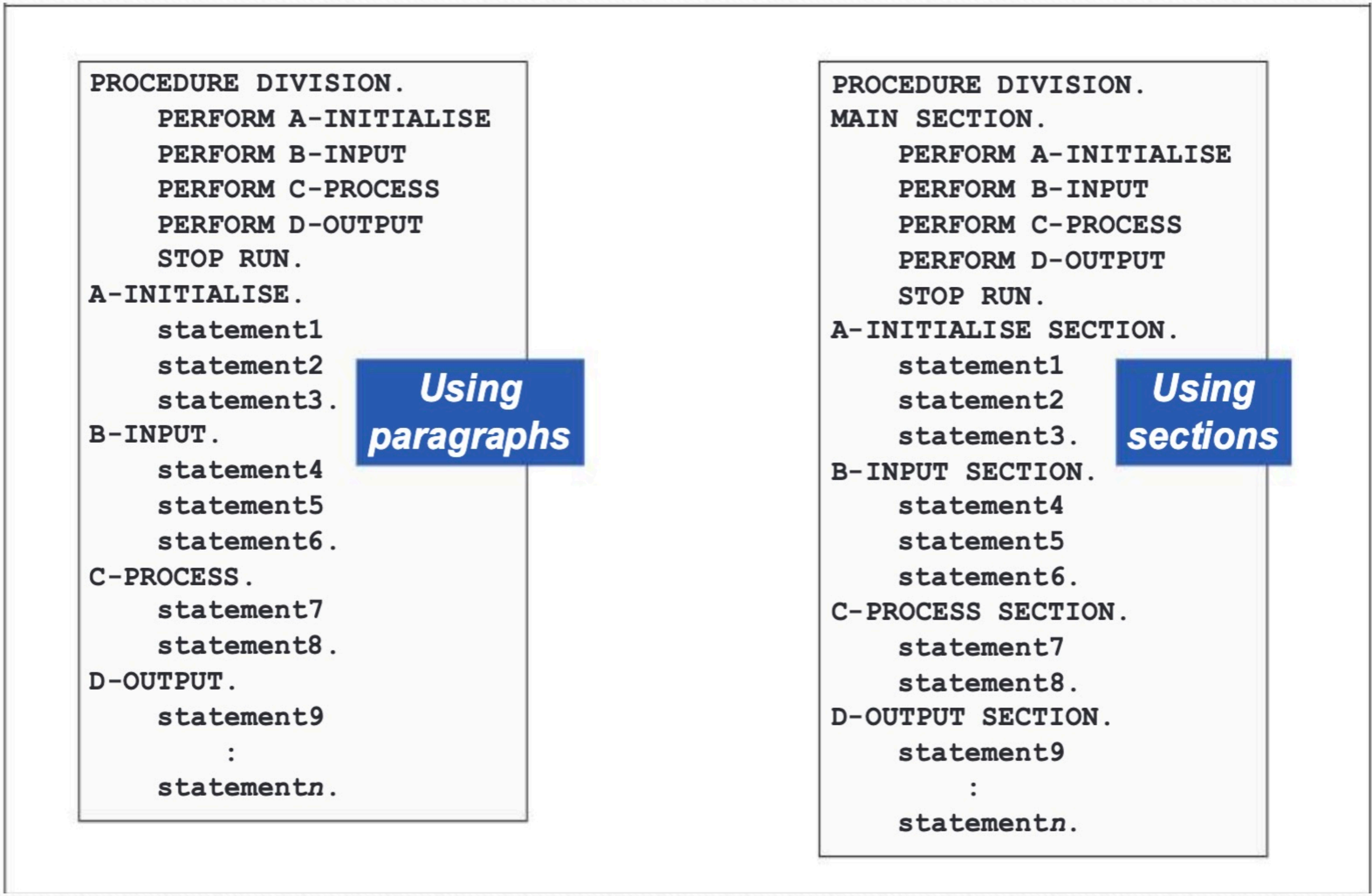
```
PERFORM WS-COUNT TIMES
  statement1
  statement2
      :
  statementn
END-PERFORM
```

```
PERFORM 4 TIMES
  statement1
  statement2
      :
  statementn
END-PERFORM
```

**Programming health warning!**

**Full-stops are not allowed!**

# Paragraphs and sections



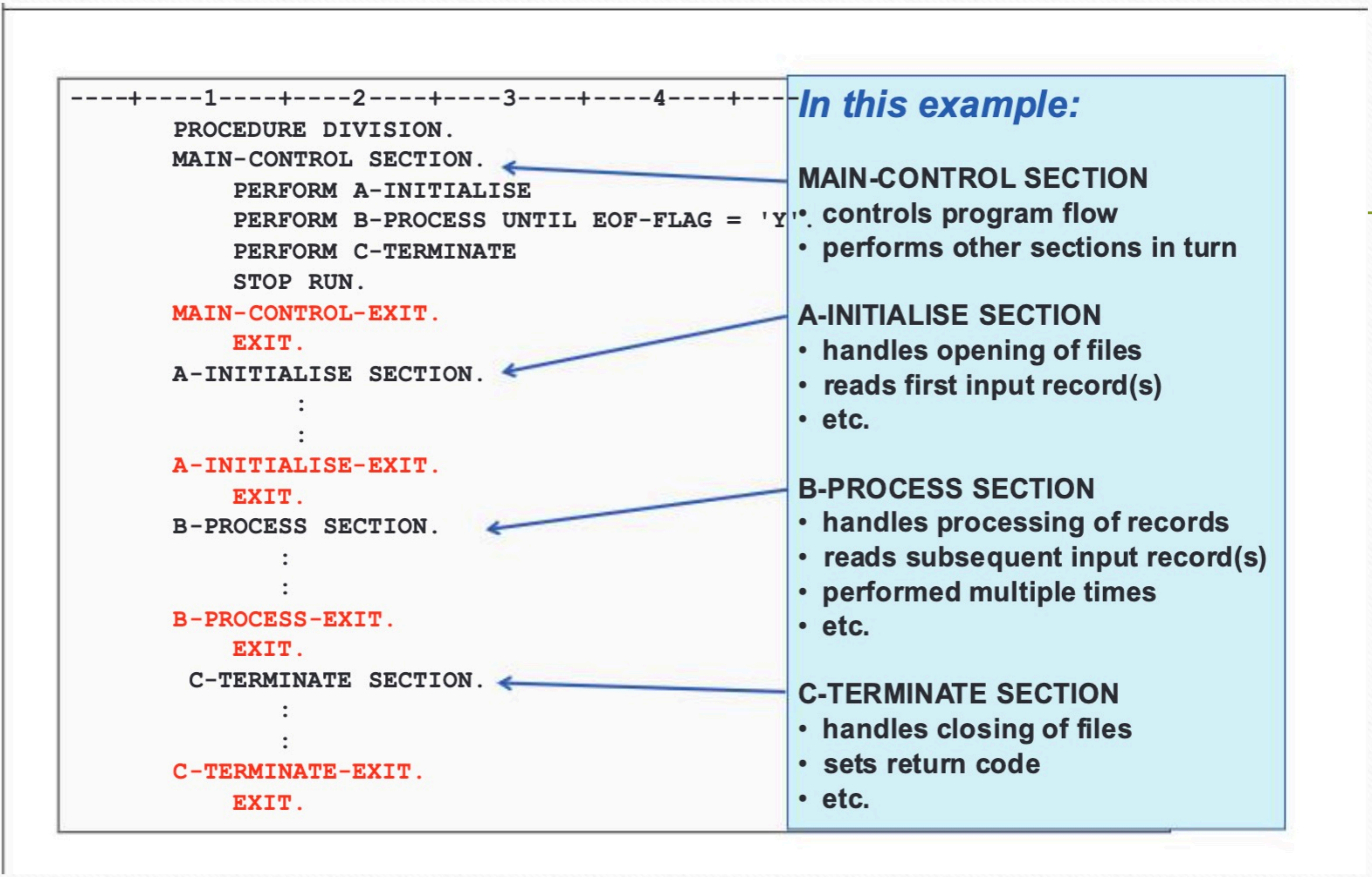
Using paragraphs

```
PROCEDURE DIVISION.
    PERFORM A-INITIALISE
    PERFORM B-INPUT
    PERFORM C-PROCESS
    PERFORM D-OUTPUT
    STOP RUN.
A-INITIALISE.
    statement1
    statement2
    statement3.
B-INPUT.
    statement4
    statement5
    statement6.
C-PROCESS.
    statement7
    statement8.
D-OUTPUT.
    statement9
        :
    statementn.
```

Using sections

```
PROCEDURE DIVISION.
MAIN SECTION.
    PERFORM A-INITIALISE
    PERFORM B-INPUT
    PERFORM C-PROCESS
    PERFORM D-OUTPUT
    STOP RUN.
A-INITIALISE SECTION.
    statement1
    statement2
    statement3.
B-INPUT SECTION.
    statement4
    statement5
    statement6.
C-PROCESS SECTION.
    statement7
    statement8.
D-OUTPUT SECTION.
    statement9
        :
    statementn.
```

# Using section



```
----+----1----+----2----+----3----+----4----+---
       PROCEDURE DIVISION.
       MAIN-CONTROL SECTION.
           PERFORM A-INITIALISE
           PERFORM B-PROCESS UNTIL EOF-FLAG = 'Y'.
           PERFORM C-TERMINATE
           STOP RUN.
       MAIN-CONTROL-EXIT.
           EXIT.
       A-INITIALISE SECTION.
               :
               :
       A-INITIALISE-EXIT.
           EXIT.
       B-PROCESS SECTION.
              :
              :
       B-PROCESS-EXIT.
           EXIT.
        C-TERMINATE SECTION.
              :
              :
       C-TERMINATE-EXIT.
           EXIT.
```

***In this example:***

**MAIN-CONTROL SECTION**

- **controls program flow**
- **performs other sections in turn**

**A-INITIALISE SECTION**

- **handles opening of files**
- **reads first input record(s)**
- **etc.**

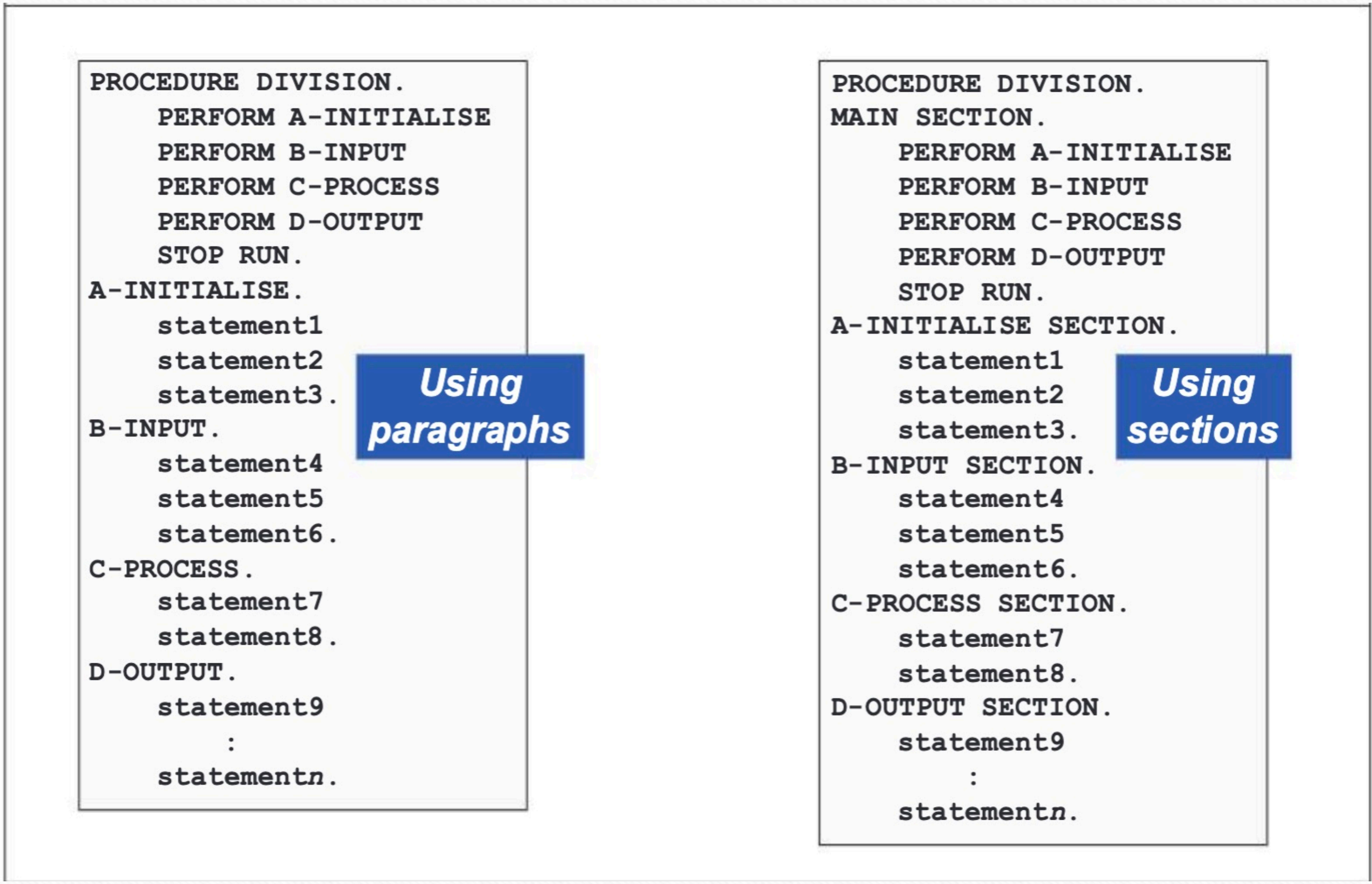
**B-PROCESS SECTION**

- **handles processing of records**
- **reads subsequent input record(s)**
- **performed multiple times**
- **etc.**

**C-TERMINATE SECTION**

- **handles closing of files**
- **sets return code**
- **etc.**

# The basic PERFORM statemnt



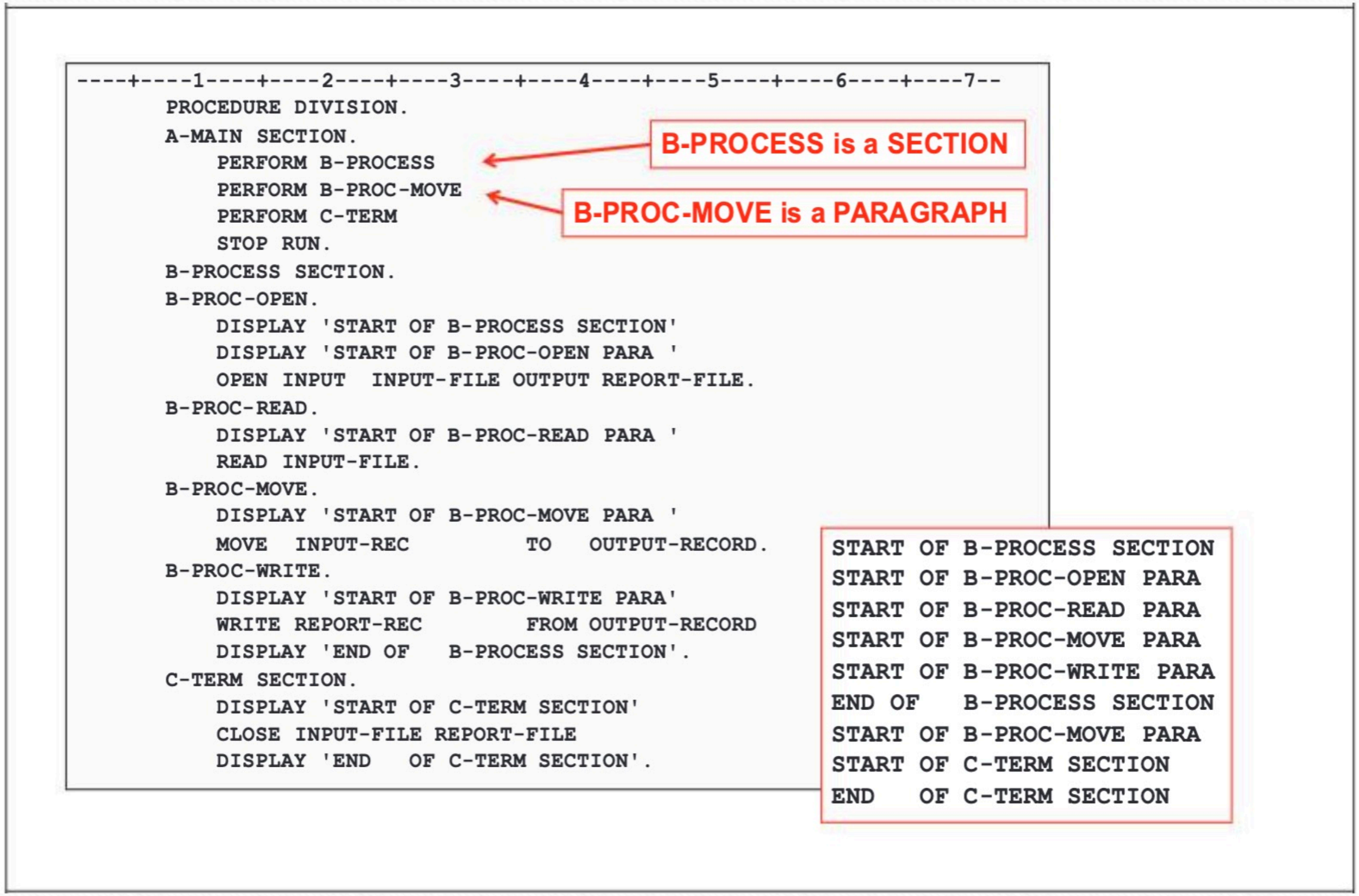
```
PROCEDURE DIVISION.
    PERFORM A-INITIALISE
    PERFORM B-INPUT
    PERFORM C-PROCESS
    PERFORM D-OUTPUT
    STOP RUN.
A-INITIALISE.
    statement1
    statement2
    statement3.
B-INPUT.
    statement4
    statement5
    statement6.
C-PROCESS.
    statement7
    statement8.
D-OUTPUT.
    statement9
        :
    statementn.
```

Using paragraphs

```
PROCEDURE DIVISION.
MAIN SECTION.
    PERFORM A-INITIALISE
    PERFORM B-INPUT
    PERFORM C-PROCESS
    PERFORM D-OUTPUT
    STOP RUN.
A-INITIALISE SECTION.
    statement1
    statement2
    statement3.
B-INPUT SECTION.
    statement4
    statement5
    statement6.
C-PROCESS SECTION.
    statement7
    statement8.
D-OUTPUT SECTION.
    statement9
        :
    statementn.
```

Using sections

# PERFORM example



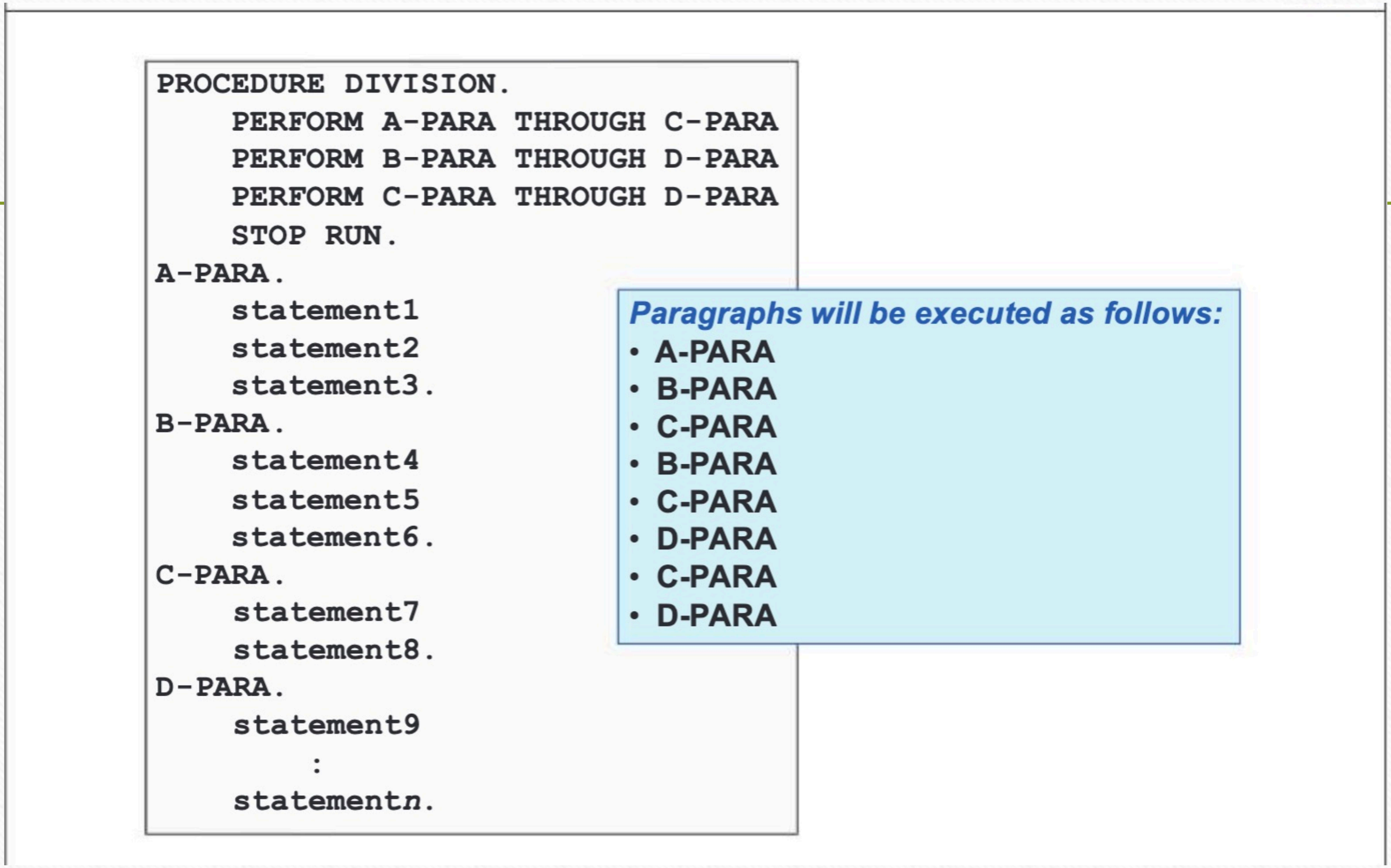
```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
       A-MAIN SECTION.
           PERFORM B-PROCESS
           PERFORM B-PROC-MOVE
           PERFORM C-TERM
           STOP RUN.
       B-PROCESS SECTION.
       B-PROC-OPEN.
           DISPLAY 'START OF B-PROCESS SECTION'
           DISPLAY 'START OF B-PROC-OPEN PARA '
           OPEN INPUT  INPUT-FILE OUTPUT REPORT-FILE.
       B-PROC-READ.
           DISPLAY 'START OF B-PROC-READ PARA '
           READ INPUT-FILE.
       B-PROC-MOVE.
           DISPLAY 'START OF B-PROC-MOVE PARA '
           MOVE  INPUT-REC         TO   OUTPUT-RECORD.
       B-PROC-WRITE.
           DISPLAY 'START OF B-PROC-WRITE PARA'
           WRITE REPORT-REC        FROM OUTPUT-RECORD
           DISPLAY 'END OF   B-PROCESS SECTION'.
       C-TERM SECTION.
           DISPLAY 'START OF C-TERM SECTION'
           CLOSE INPUT-FILE REPORT-FILE
           DISPLAY 'END   OF C-TERM SECTION'.
```

B-PROCESS is a SECTION

B-PROC-MOVE is a PARAGRAPH

```
START OF B-PROCESS SECTION
START OF B-PROC-OPEN PARA
START OF B-PROC-READ PARA
START OF B-PROC-MOVE PARA
START OF B-PROC-WRITE PARA
END OF   B-PROCESS SECTION
START OF B-PROC-MOVE PARA
START OF C-TERM SECTION
END   OF C-TERM SECTION
```

# PERFORM … THROUGH .

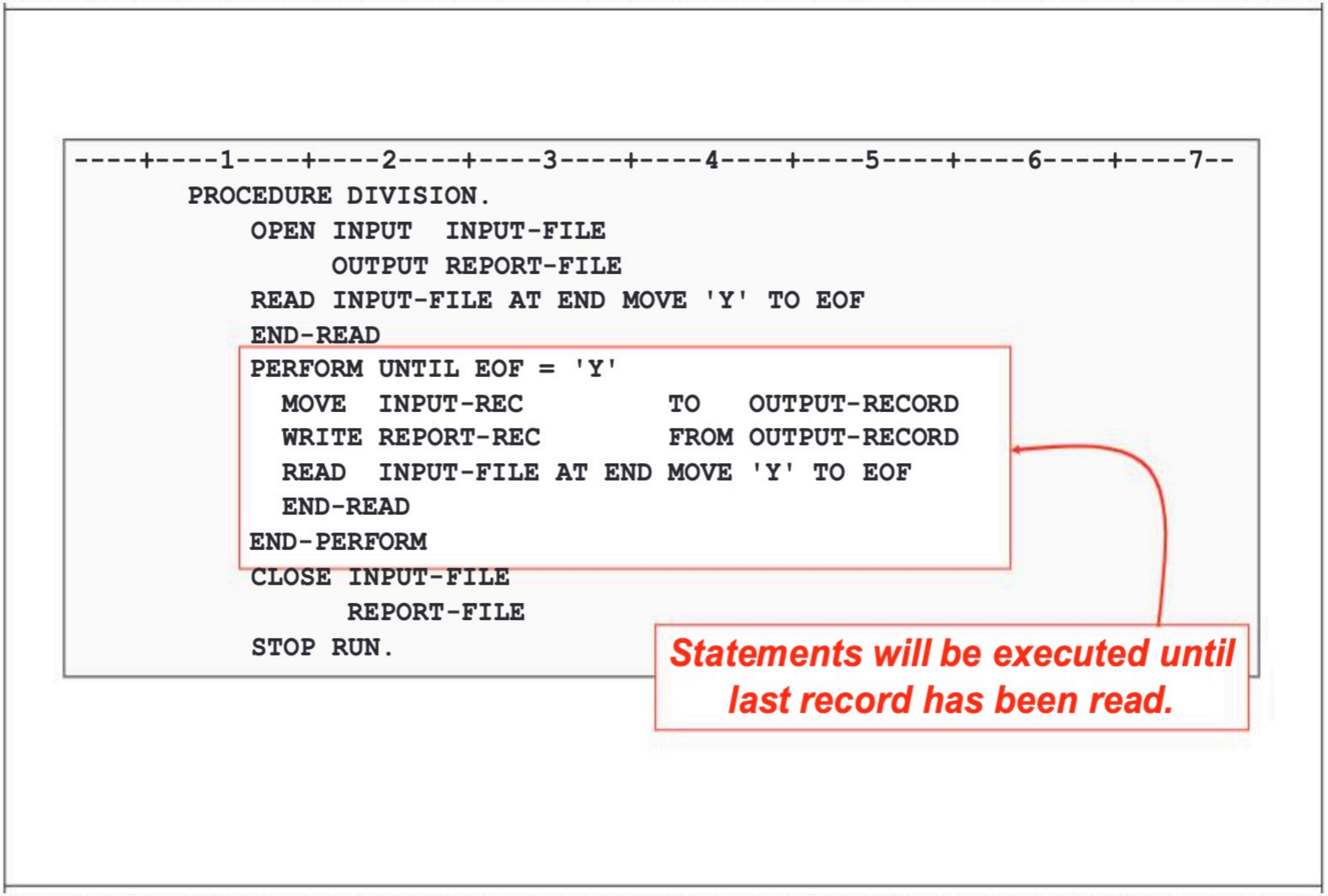


```
PROCEDURE DIVISION.
    PERFORM A-PARA THROUGH C-PARA
    PERFORM B-PARA THROUGH D-PARA
    PERFORM C-PARA THROUGH D-PARA
    STOP RUN.
A-PARA.
    statement1
    statement2
    statement3.
B-PARA.
    statement4
    statement5
    statement6.
C-PARA.
    statement7
    statement8.
D-PARA.
    statement9
        :
    statementn.
```

***Paragraphs will be executed as follows:***

- **A-PARA**
- **B-PARA**
- **C-PARA**
- **B-PARA**
- **C-PARA**
- **D-PARA**
- **C-PARA**
- **D-PARA**

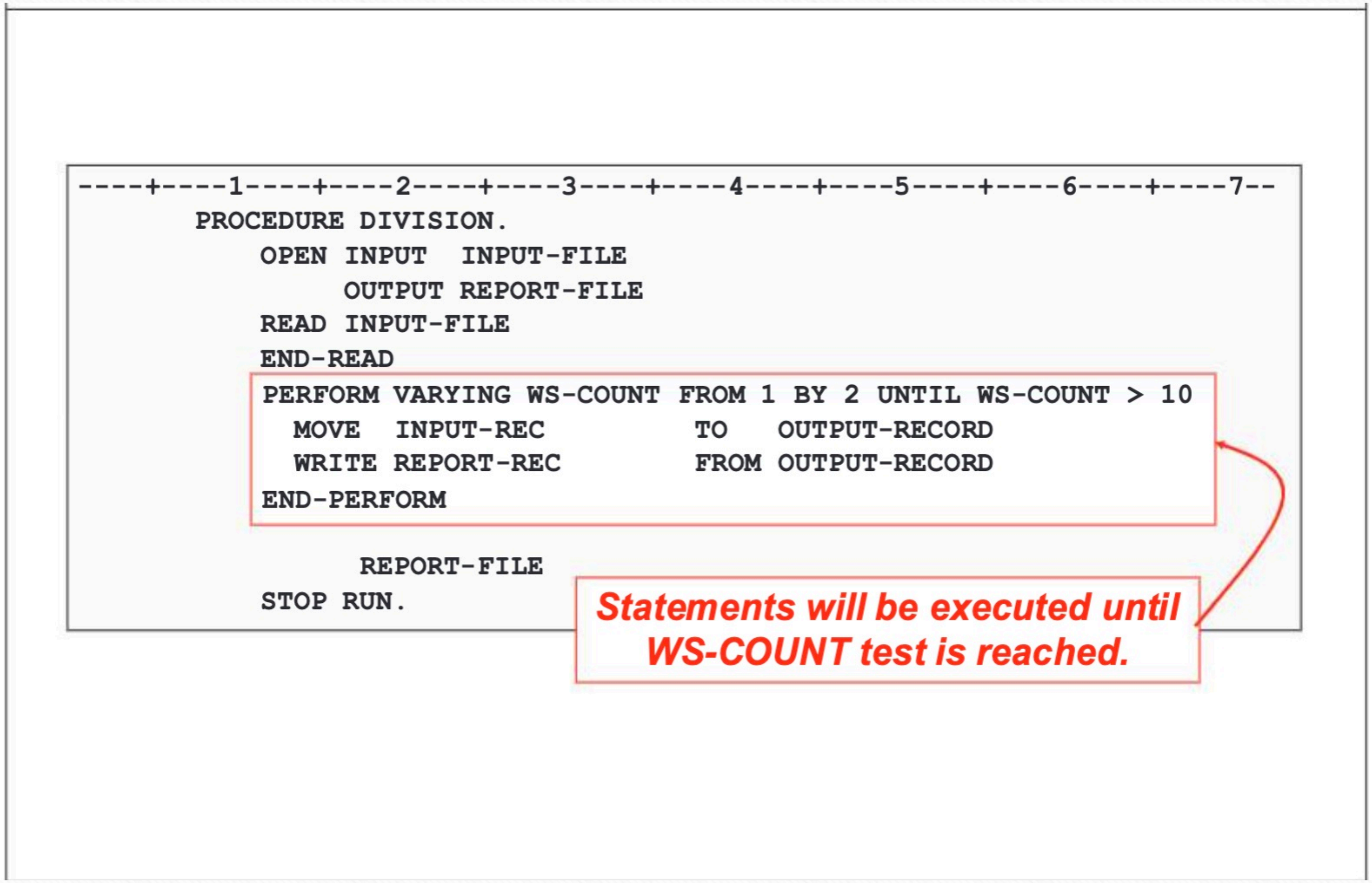
# PERFORM UNTIL



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
           OPEN INPUT  INPUT-FILE
                OUTPUT REPORT-FILE
           READ INPUT-FILE AT END MOVE 'Y' TO EOF
           END-READ
           PERFORM UNTIL EOF = 'Y'
             MOVE  INPUT-REC         TO   OUTPUT-RECORD
             WRITE REPORT-REC        FROM OUTPUT-RECORD
             READ  INPUT-FILE AT END MOVE 'Y' TO EOF
             END-READ
           END-PERFORM
           CLOSE INPUT-FILE
                 REPORT-FILE
           STOP RUN.
```

*Statements will be executed until last record has been read.*

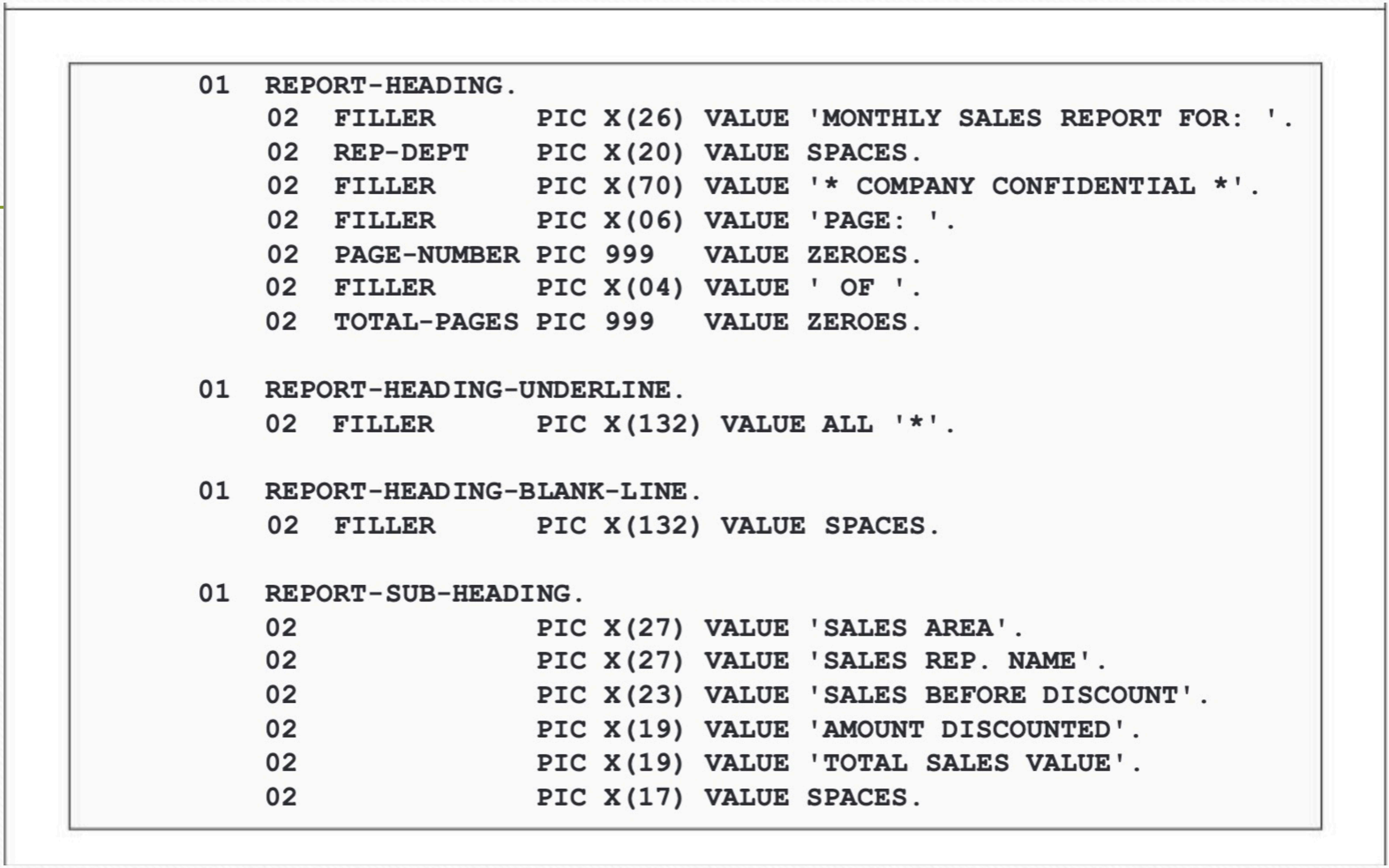
# PERFORM VARYING … UNTIL



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       PROCEDURE DIVISION.
           OPEN INPUT  INPUT-FILE
                OUTPUT REPORT-FILE
           READ INPUT-FILE
           END-READ
           PERFORM VARYING WS-COUNT FROM 1 BY 2 UNTIL WS-COUNT > 10
             MOVE  INPUT-REC          TO   OUTPUT-RECORD
             WRITE REPORT-REC         FROM OUTPUT-RECORD
           END-PERFORM

                 REPORT-FILE
           STOP RUN.
```

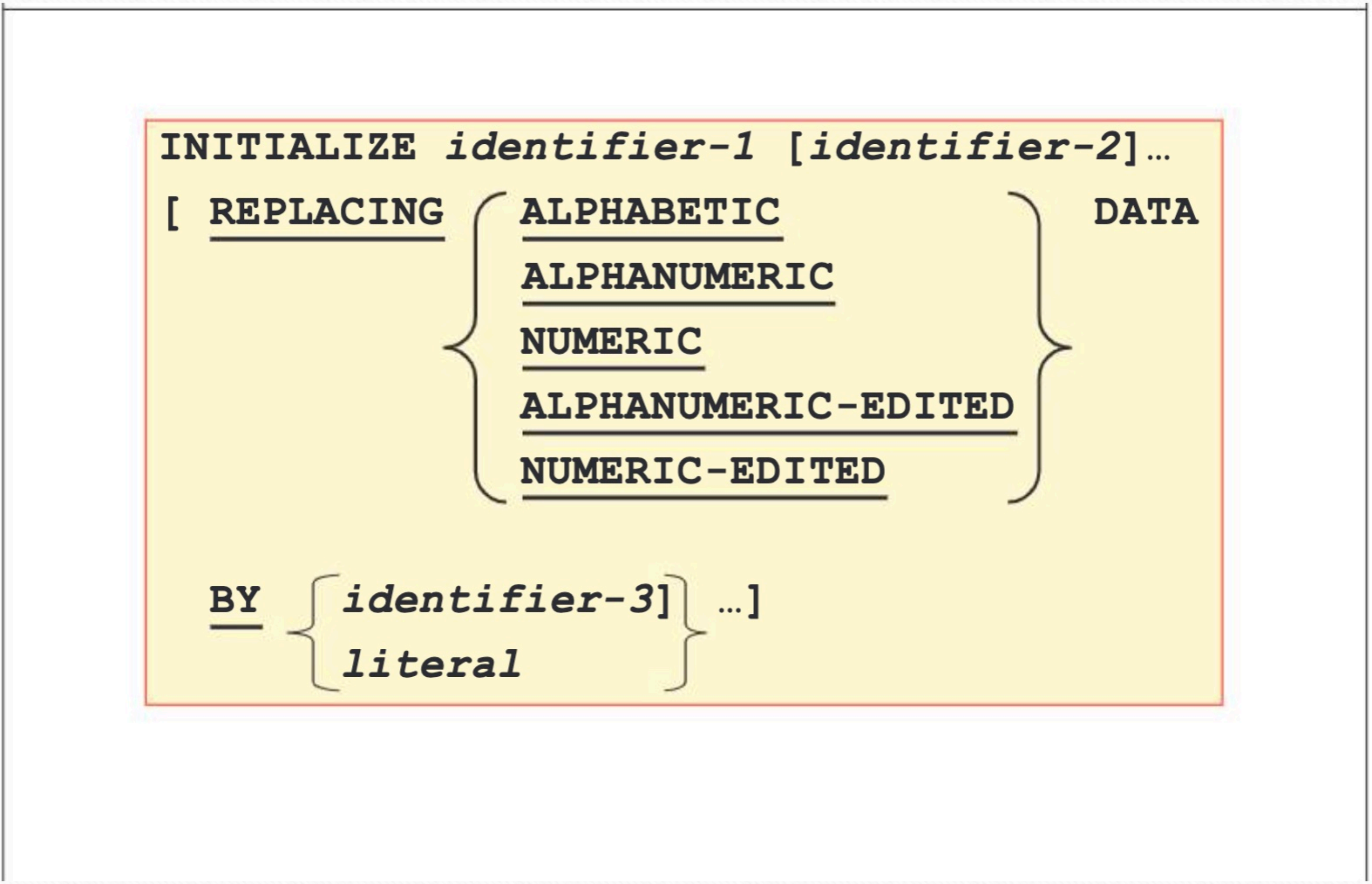
*Statements will be executed until WS-COUNT test is reached.*

# Using the VALUE clause for initial values



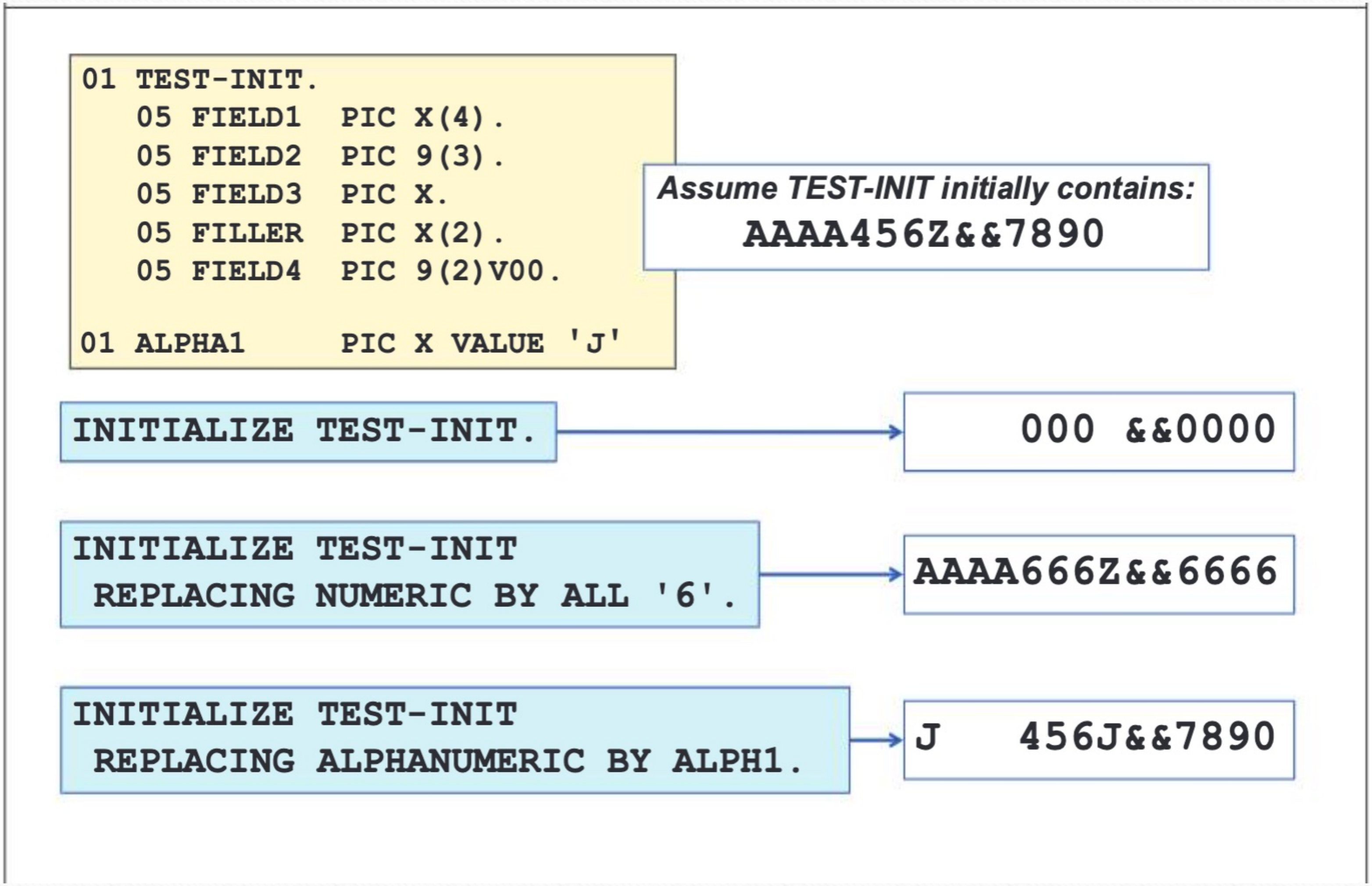
```
01  REPORT-HEADING.
    02  FILLER      PIC X(26) VALUE 'MONTHLY SALES REPORT FOR: '.
    02  REP-DEPT    PIC X(20) VALUE SPACES.
    02  FILLER      PIC X(70) VALUE '* COMPANY CONFIDENTIAL *'.
    02  FILLER      PIC X(06) VALUE 'PAGE: '.
    02  PAGE-NUMBER PIC 999   VALUE ZEROES.
    02  FILLER      PIC X(04) VALUE ' OF '.
    02  TOTAL-PAGES PIC 999   VALUE ZEROES.

01  REPORT-HEADING-UNDERLINE.
    02  FILLER      PIC X(132) VALUE ALL '*'.

01  REPORT-HEADING-BLANK-LINE.
    02  FILLER      PIC X(132) VALUE SPACES.

01  REPORT-SUB-HEADING.
    02              PIC X(27) VALUE 'SALES AREA'.
    02              PIC X(27) VALUE 'SALES REP. NAME'.
    02              PIC X(23) VALUE 'SALES BEFORE DISCOUNT'.
    02              PIC X(19) VALUE 'AMOUNT DISCOUNTED'.
    02              PIC X(19) VALUE 'TOTAL SALES VALUE'.
    02              PIC X(17) VALUE SPACES.
```

# INITIALIZE command with REPLACING



```
INITIALIZE identifier-1 [identifier-2]...

[ REPLACING ⎧ ALPHABETIC             ⎫ DATA
            ⎪ ALPHANUMERIC           ⎪
            ⎨ NUMERIC                ⎬
            ⎪ ALPHANUMERIC-EDITED    ⎪
            ⎩ NUMERIC-EDITED         ⎭

   BY ⎧ identifier-3] ⎫ ...]
      ⎩ literal       ⎭
```

# Examples of INITIALIZE



```
01 TEST-INIT.
   05 FIELD1   PIC X(4).
   05 FIELD2   PIC 9(3).
   05 FIELD3   PIC X.
   05 FILLER   PIC X(2).
   05 FIELD4   PIC 9(2)V00.

01 ALPHA1      PIC X VALUE 'J'
```

*Assume TEST-INIT initially contains:*

```
AAAA456Z&&7890
```

```
INITIALIZE TEST-INIT.
```

→

```
    000 &&0000
```

```
INITIALIZE TEST-INIT
 REPLACING NUMERIC BY ALL '6'.
```

→

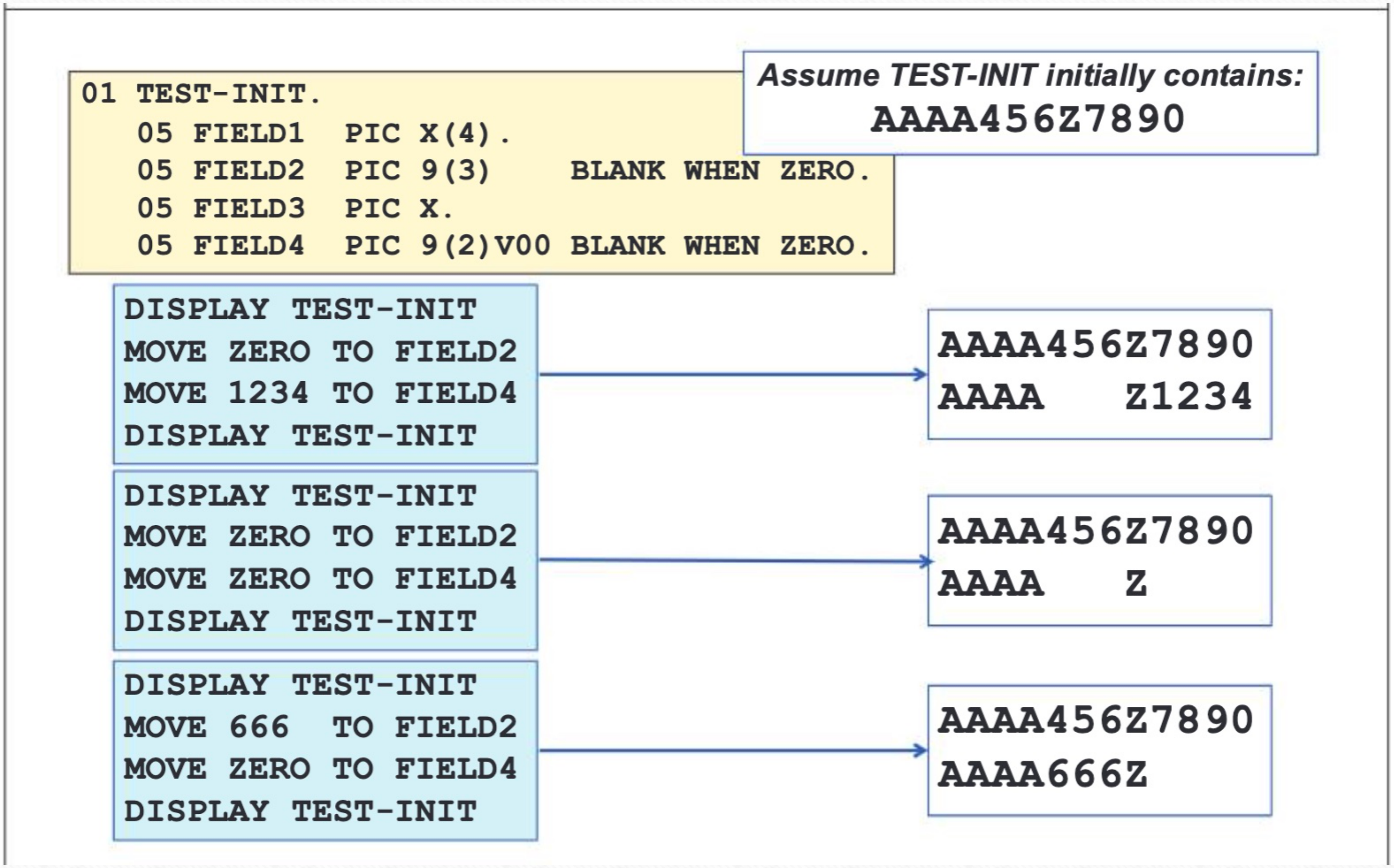
```
AAAA666Z&&6666
```

```
INITIALIZE TEST-INIT
 REPLACING ALPHANUMERIC BY ALPH1.
```

→

```
J   456J&&7890
```

# BLANK when ZERO clause



```
01 TEST-INIT.
   05 FIELD1  PIC X(4).
   05 FIELD2  PIC 9(3)    BLANK WHEN ZERO.
   05 FIELD3  PIC X.
   05 FIELD4  PIC 9(2)V00 BLANK WHEN ZERO.
```

*Assume TEST-INIT initially contains:*

```
AAAA456Z7890
```

```
DISPLAY TEST-INIT
MOVE ZERO TO FIELD2
MOVE 1234 TO FIELD4
DISPLAY TEST-INIT
```

→

```
AAAA456Z7890
AAAA   Z1234
```

```
DISPLAY TEST-INIT
MOVE ZERO TO FIELD2
MOVE ZERO TO FIELD4
DISPLAY TEST-INIT
```

→

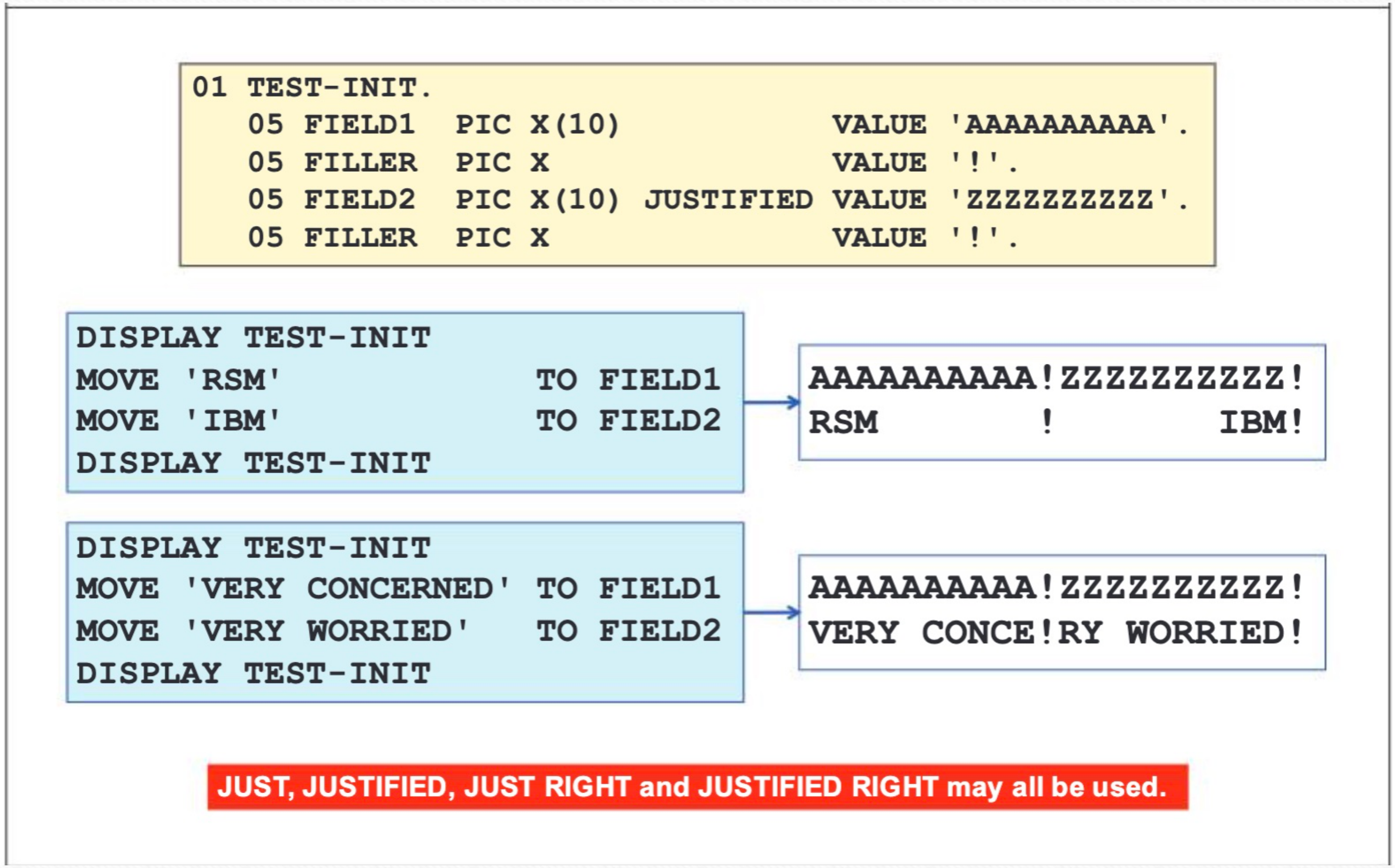
```
AAAA456Z7890
AAAA   Z
```

```
DISPLAY TEST-INIT
MOVE 666  TO FIELD2
MOVE ZERO TO FIELD4
DISPLAY TEST-INIT
```

→

```
AAAA456Z7890
AAAA666Z
```

# JUSTIFIED clause effect



```
01 TEST-INIT.
   05 FIELD1  PIC X(10)           VALUE 'AAAAAAAAAA'.
   05 FILLER  PIC X               VALUE '!'.
   05 FIELD2  PIC X(10) JUSTIFIED VALUE 'ZZZZZZZZZZ'.
   05 FILLER  PIC X               VALUE '!'.
```

```
DISPLAY TEST-INIT
MOVE 'RSM'             TO FIELD1
MOVE 'IBM'             TO FIELD2
DISPLAY TEST-INIT
```

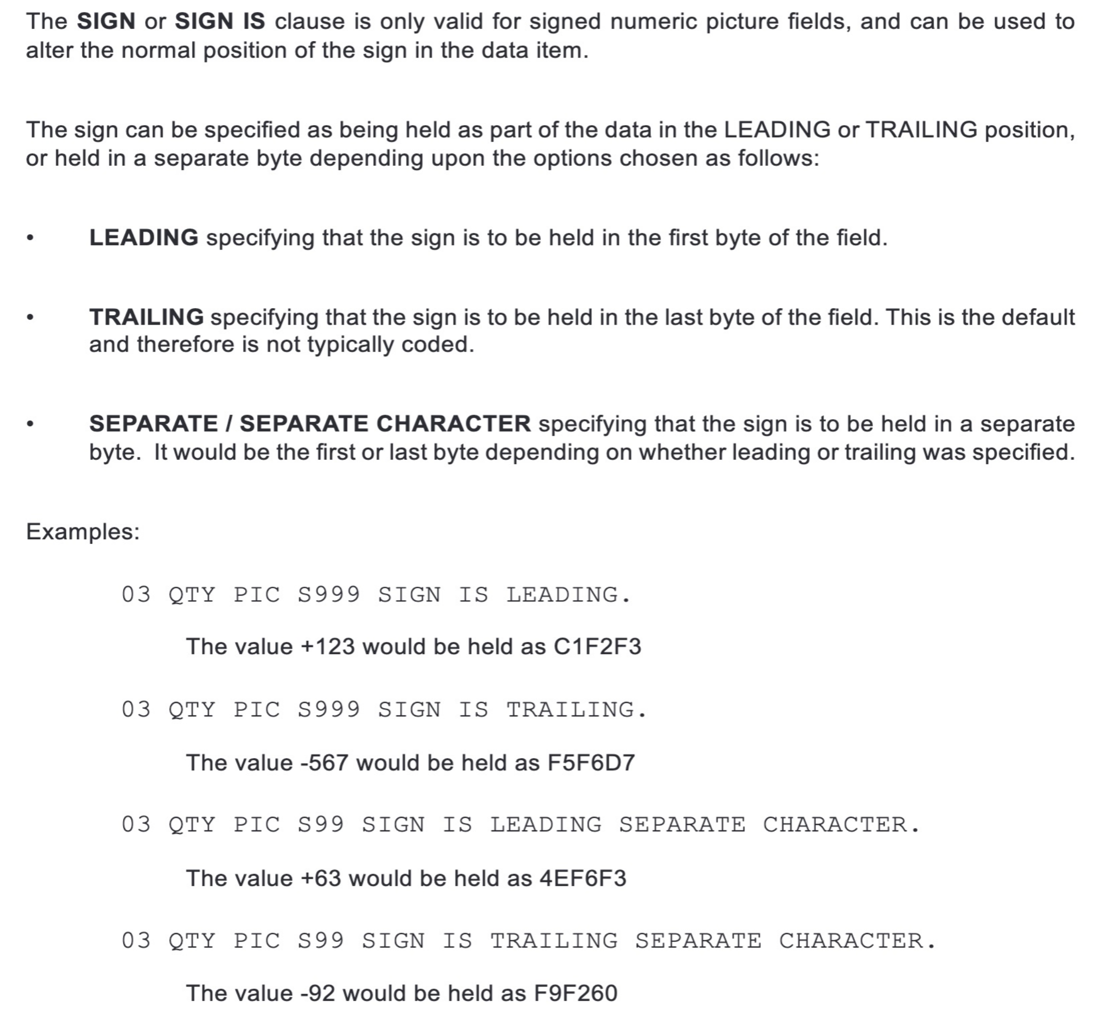
```
AAAAAAAAAA!ZZZZZZZZZZ!
RSM       !       IBM!
```

```
DISPLAY TEST-INIT
MOVE 'VERY CONCERNED'  TO FIELD1
MOVE 'VERY WORRIED'    TO FIELD2
DISPLAY TEST-INIT
```

```
AAAAAAAAAA!ZZZZZZZZZZ!
VERY CONCE!RY WORRIED!
```

JUST, JUSTIFIED, JUST RIGHT and JUSTIFIED RIGHT may all be used.

# SIGN IS



The **SIGN** or **SIGN IS** clause is only valid for signed numeric picture fields, and can be used to alter the normal position of the sign in the data item.

The sign can be specified as being held as part of the data in the LEADING or TRAILING position, or held in a separate byte depending upon the options chosen as follows:

- **LEADING** specifying that the sign is to be held in the first byte of the field.
- **TRAILING** specifying that the sign is to be held in the last byte of the field. This is the default and therefore is not typically coded.
- **SEPARATE / SEPARATE CHARACTER** specifying that the sign is to be held in a separate byte. It would be the first or last byte depending on whether leading or trailing was specified.

Examples:

```
03 QTY PIC S999 SIGN IS LEADING.
```

The value +123 would be held as C1F2F3

```
03 QTY PIC S999 SIGN IS TRAILING.
```

The value -567 would be held as F5F6D7

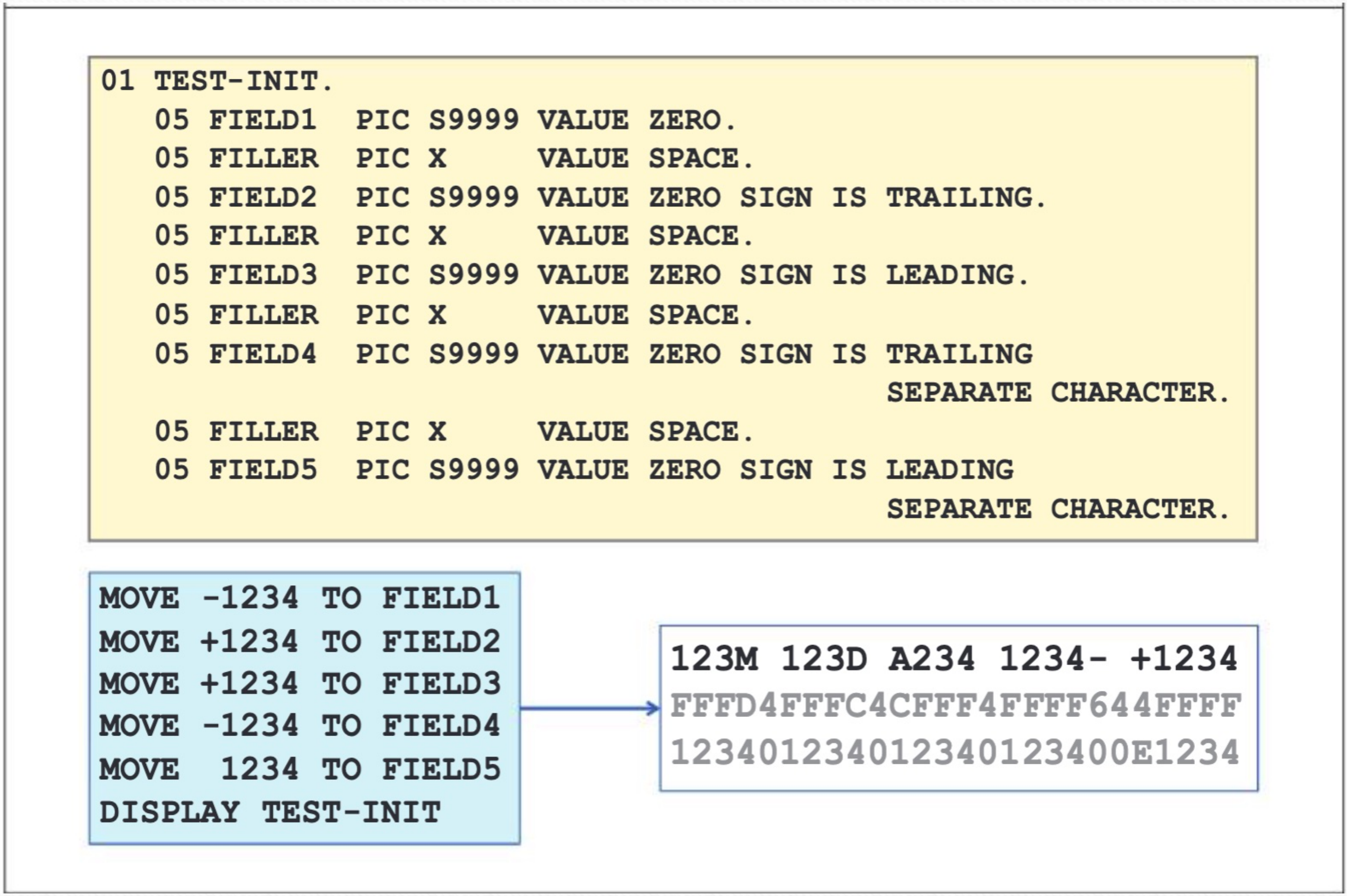
```
03 QTY PIC S99 SIGN IS LEADING SEPARATE CHARACTER.
```

The value +63 would be held as 4EF6F3

```
03 QTY PIC S99 SIGN IS TRAILING SEPARATE CHARACTER.
```

The value -92 would be held as F9F260

# SIGN IS clause

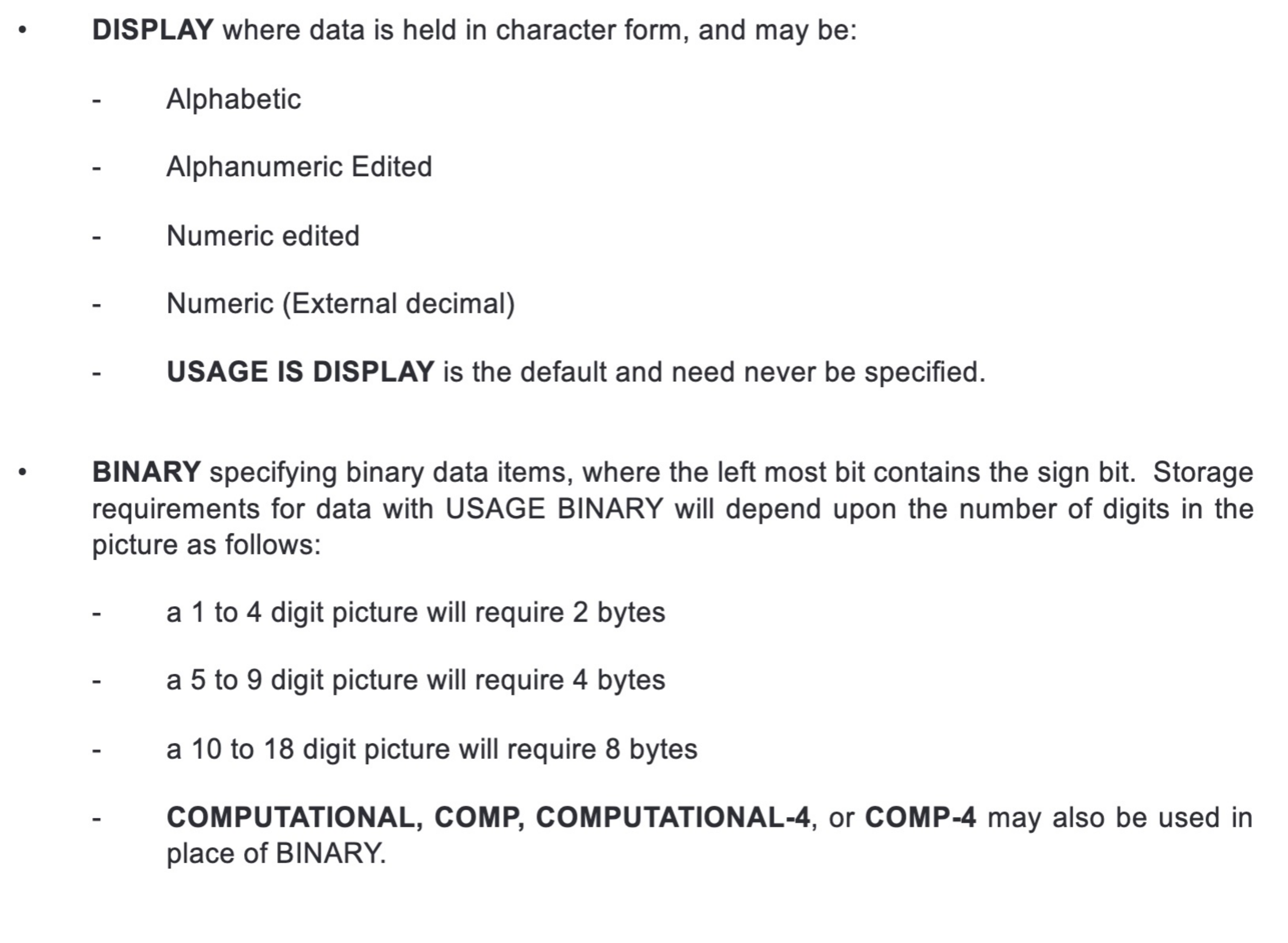


```
01 TEST-INIT.
   05 FIELD1  PIC S9999 VALUE ZERO.
   05 FILLER  PIC X     VALUE SPACE.
   05 FIELD2  PIC S9999 VALUE ZERO SIGN IS TRAILING.
   05 FILLER  PIC X     VALUE SPACE.
   05 FIELD3  PIC S9999 VALUE ZERO SIGN IS LEADING.
   05 FILLER  PIC X     VALUE SPACE.
   05 FIELD4  PIC S9999 VALUE ZERO SIGN IS TRAILING
                                    SEPARATE CHARACTER.
   05 FILLER  PIC X     VALUE SPACE.
   05 FIELD5  PIC S9999 VALUE ZERO SIGN IS LEADING
                                    SEPARATE CHARACTER.
```

```
MOVE -1234 TO FIELD1
MOVE +1234 TO FIELD2
MOVE +1234 TO FIELD3
MOVE -1234 TO FIELD4
MOVE  1234 TO FIELD5
DISPLAY TEST-INIT
```

```
123M 123D A234 1234- +1234
FFFD4FFFC4CFFF4FFFF644FFFF
123401234012340123400E1234
```

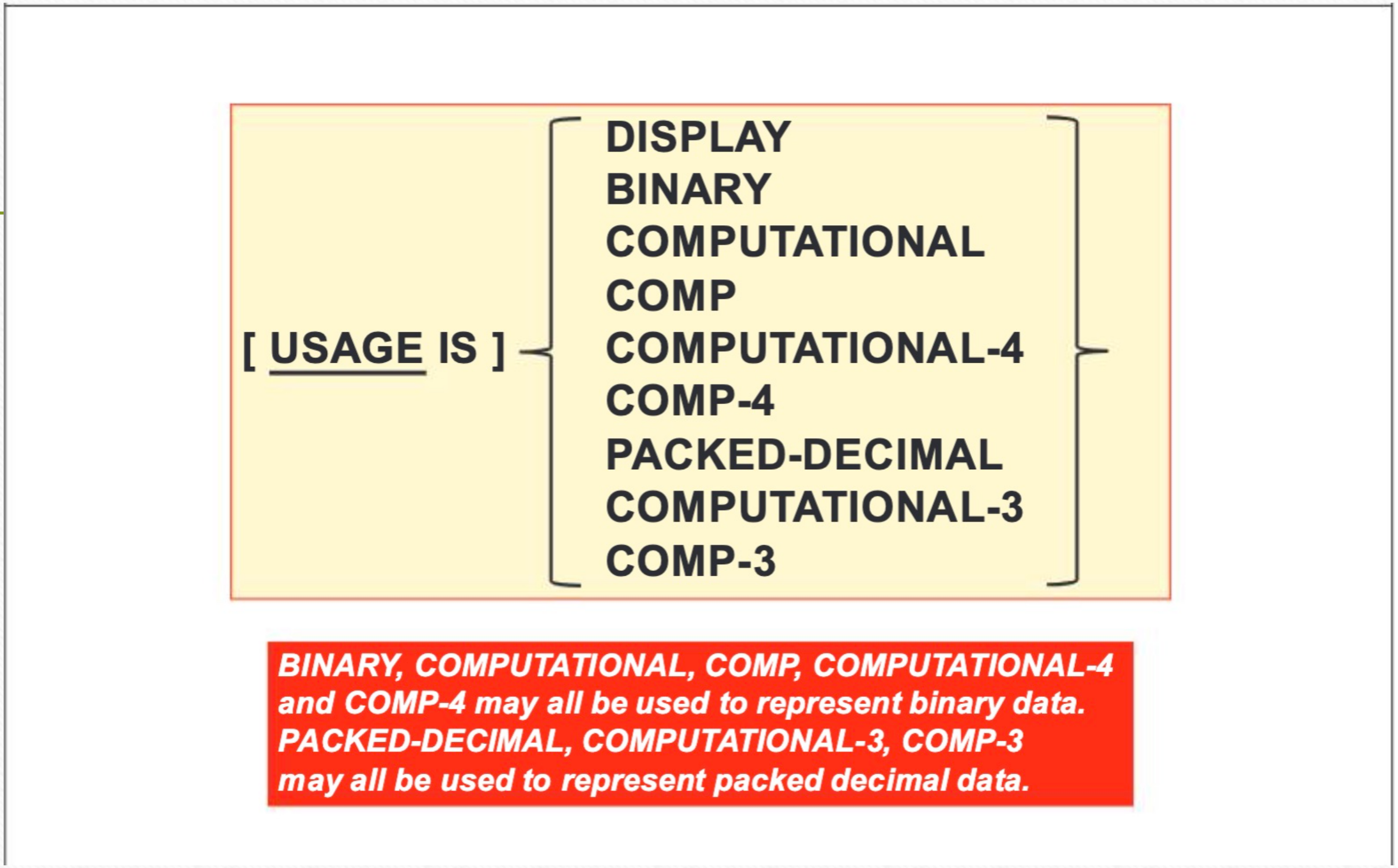
# USAGE IS



- **DISPLAY** where data is held in character form, and may be:
  - Alphabetic
  - Alphanumeric Edited
  - Numeric edited
  - Numeric (External decimal)
  - **USAGE IS DISPLAY** is the default and need never be specified.

- **BINARY** specifying binary data items, where the left most bit contains the sign bit. Storage requirements for data with USAGE BINARY will depend upon the number of digits in the picture as follows:
  - a 1 to 4 digit picture will require 2 bytes
  - a 5 to 9 digit picture will require 4 bytes
  - a 10 to 18 digit picture will require 8 bytes
  - **COMPUTATIONAL, COMP, COMPUTATIONAL-4**, or **COMP-4** may also be used in place of BINARY.

# USAGE IS clause (data types) - 1



```
              ⎧ DISPLAY                ⎫
              ⎪ BINARY                 ⎪
              ⎪ COMPUTATIONAL          ⎪
              ⎪ COMP                   ⎪
[ USAGE IS ]  ⎨ COMPUTATIONAL-4        ⎬
              ⎪ COMP-4                 ⎪
              ⎪ PACKED-DECIMAL         ⎪
              ⎪ COMPUTATIONAL-3        ⎪
              ⎩ COMP-3                 ⎭
```

***BINARY, COMPUTATIONAL, COMP, COMPUTATIONAL-4 and COMP-4 may all be used to represent binary data. PACKED-DECIMAL, COMPUTATIONAL-3, COMP-3 may all be used to represent packed decimal data.***

# COMP-1, COMP-2 USAGE

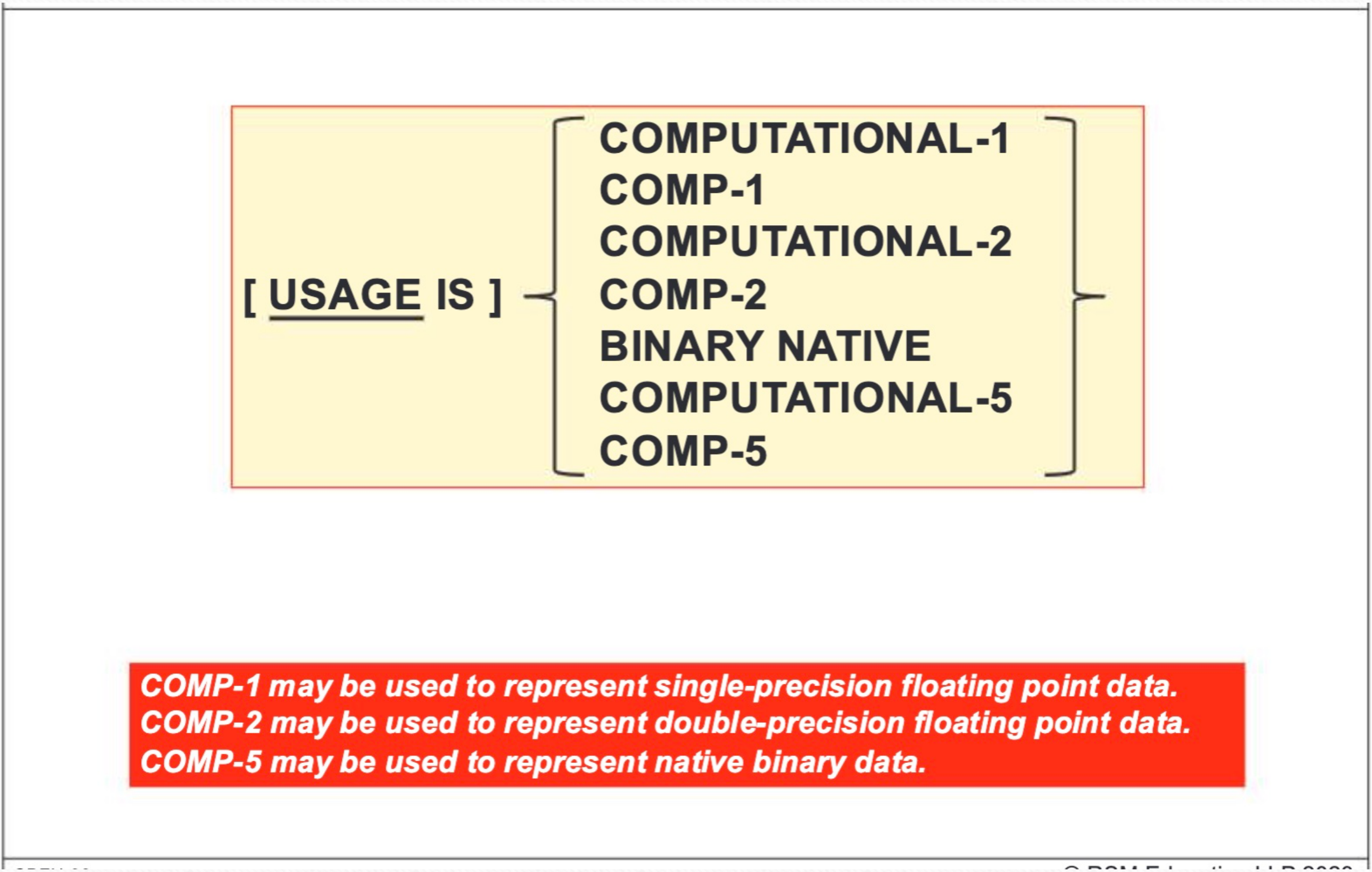
COMP-1 refers to short (single-precision) floating-point format, and COMP-2 refers to long (double-precision) floating-point format, which occupy 4 and 8 bytes of storage, respectively.

The leftmost bit contains the sign; the next seven bits contain the exponent; the remaining 3 or 7 bytes contain the mantissa.

Example:

05 COMPUTE-RESULT USAGE COMP-1 VALUE 06.23E-24.

# USAGE IS clause (data types) - 2



```
                 ┌ COMPUTATIONAL-1 ┐
                 │ COMP-1          │
                 │ COMPUTATIONAL-2 │
[ USAGE IS ]  ───┤ COMP-2          ├───
                 │ BINARY NATIVE   │
                 │ COMPUTATIONAL-5 │
                 └ COMP-5          ┘
```

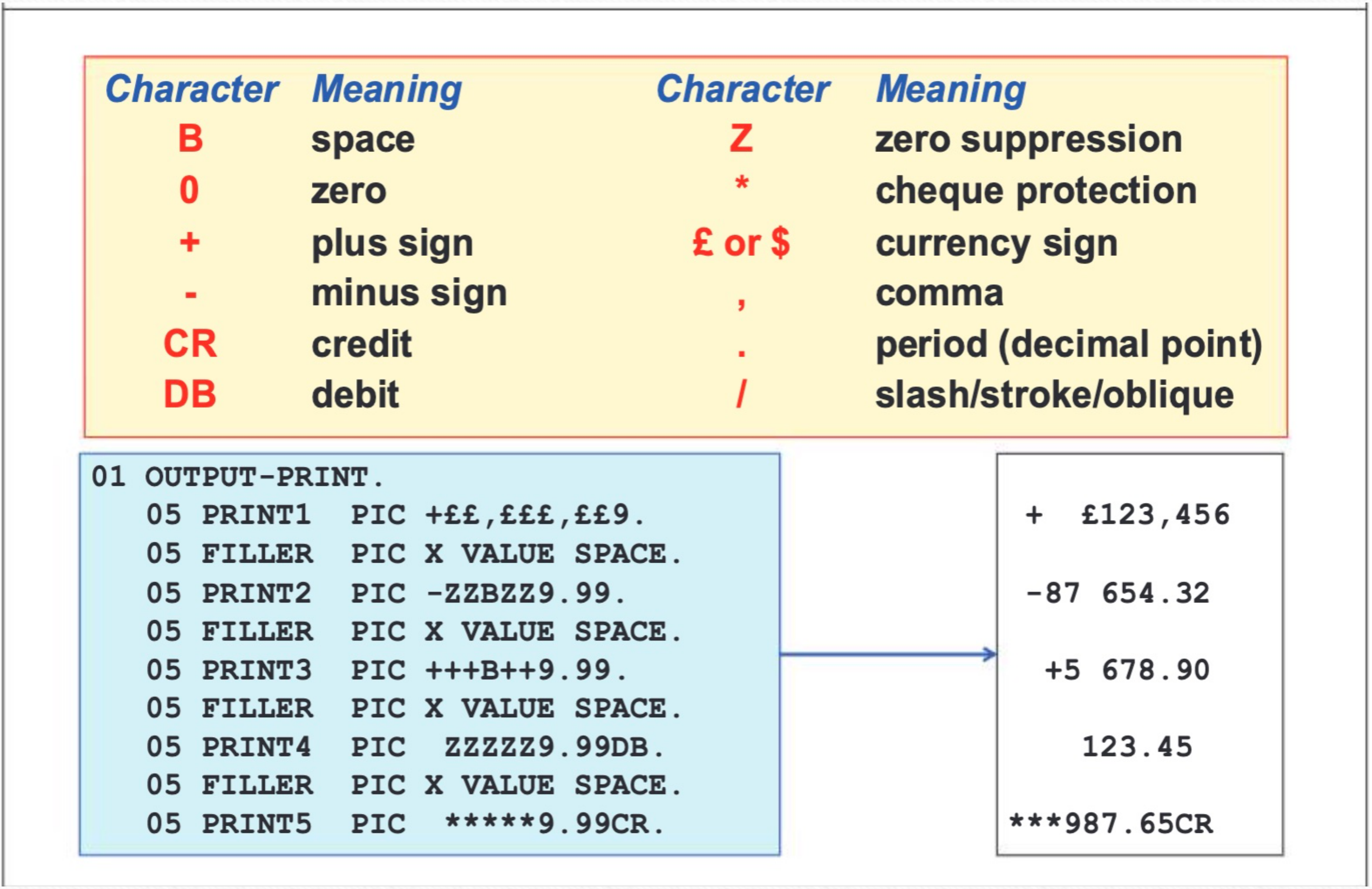
***COMP-1 may be used to represent single-precision floating point data.***
***COMP-2 may be used to represent double-precision floating point data.***
***COMP-5 may be used to represent native binary data.***

# Binary native COMP-5

The highest value is 2 to the power of 64 minus 1 (8 bytes) – S9(18)

| | | |
|---|---|---|
| S9(1) through S9(4) | Binary halfword (2 bytes) | -32768 through +32767 |
| S9(5) through S9(9) | Binary fullword (4 bytes) | -2,147,483,648 through +2,147,483,647 |

# Editing (inserted) characters

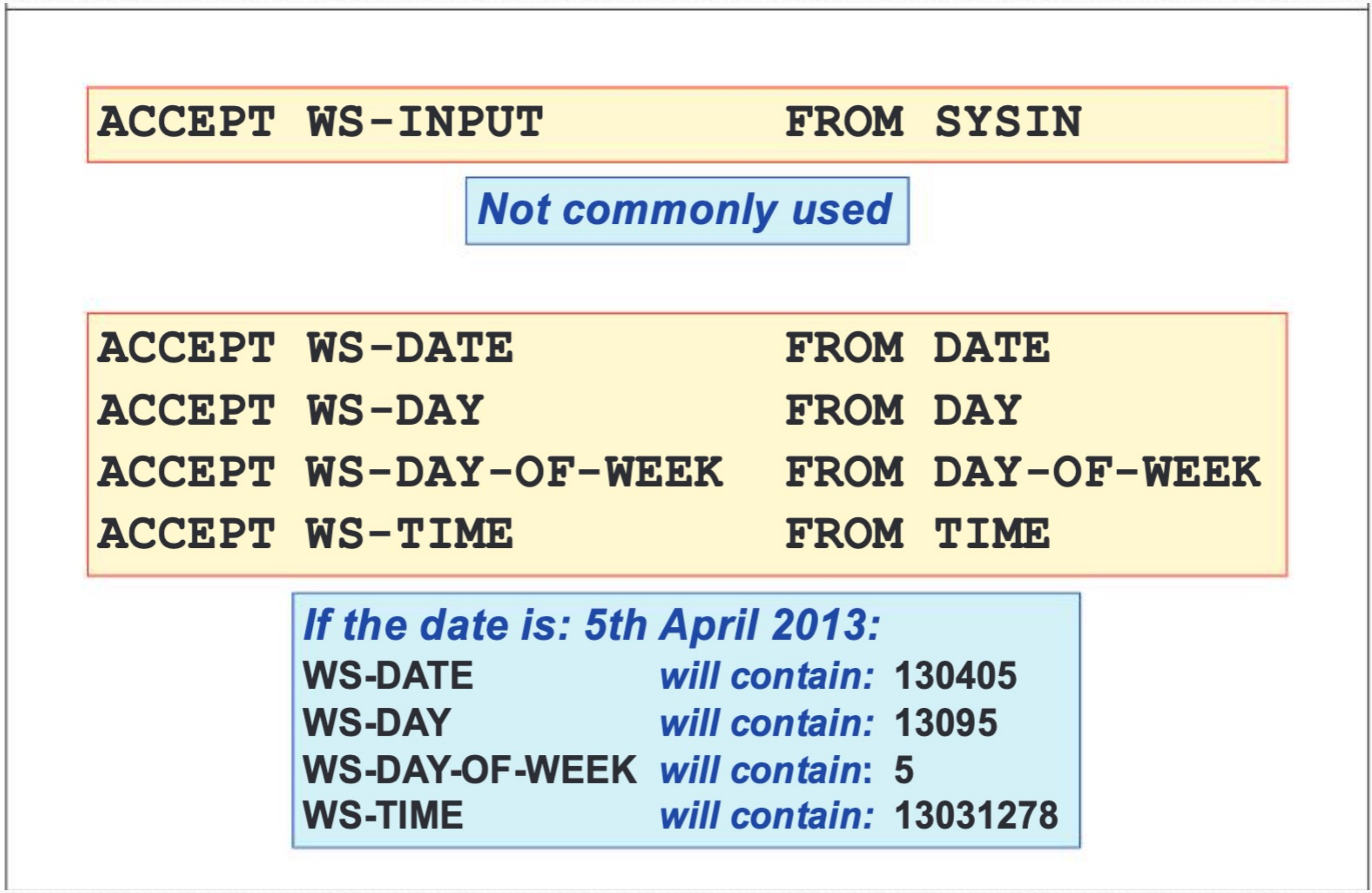


| Character | Meaning | Character | Meaning |
|---|---|---|---|
| B | space | Z | zero suppression |
| 0 | zero | * | cheque protection |
| + | plus sign | £ or $ | currency sign |
| - | minus sign | , | comma |
| CR | credit | . | period (decimal point) |
| DB | debit | / | slash/stroke/oblique |

```
01 OUTPUT-PRINT.
   05 PRINT1   PIC +££,£££,££9.
   05 FILLER   PIC X VALUE SPACE.
   05 PRINT2   PIC -ZZBZZ9.99.
   05 FILLER   PIC X VALUE SPACE.
   05 PRINT3   PIC +++B++9.99.
   05 FILLER   PIC X VALUE SPACE.
   05 PRINT4   PIC  ZZZZZ9.99DB.
   05 FILLER   PIC X VALUE SPACE.
   05 PRINT5   PIC  *****9.99CR.
```

```
 +  £123,456

 -87 654.32

  +5 678.90

    123.45

***987.65CR
```

# ACCEPT



```
ACCEPT WS-INPUT          FROM SYSIN
```

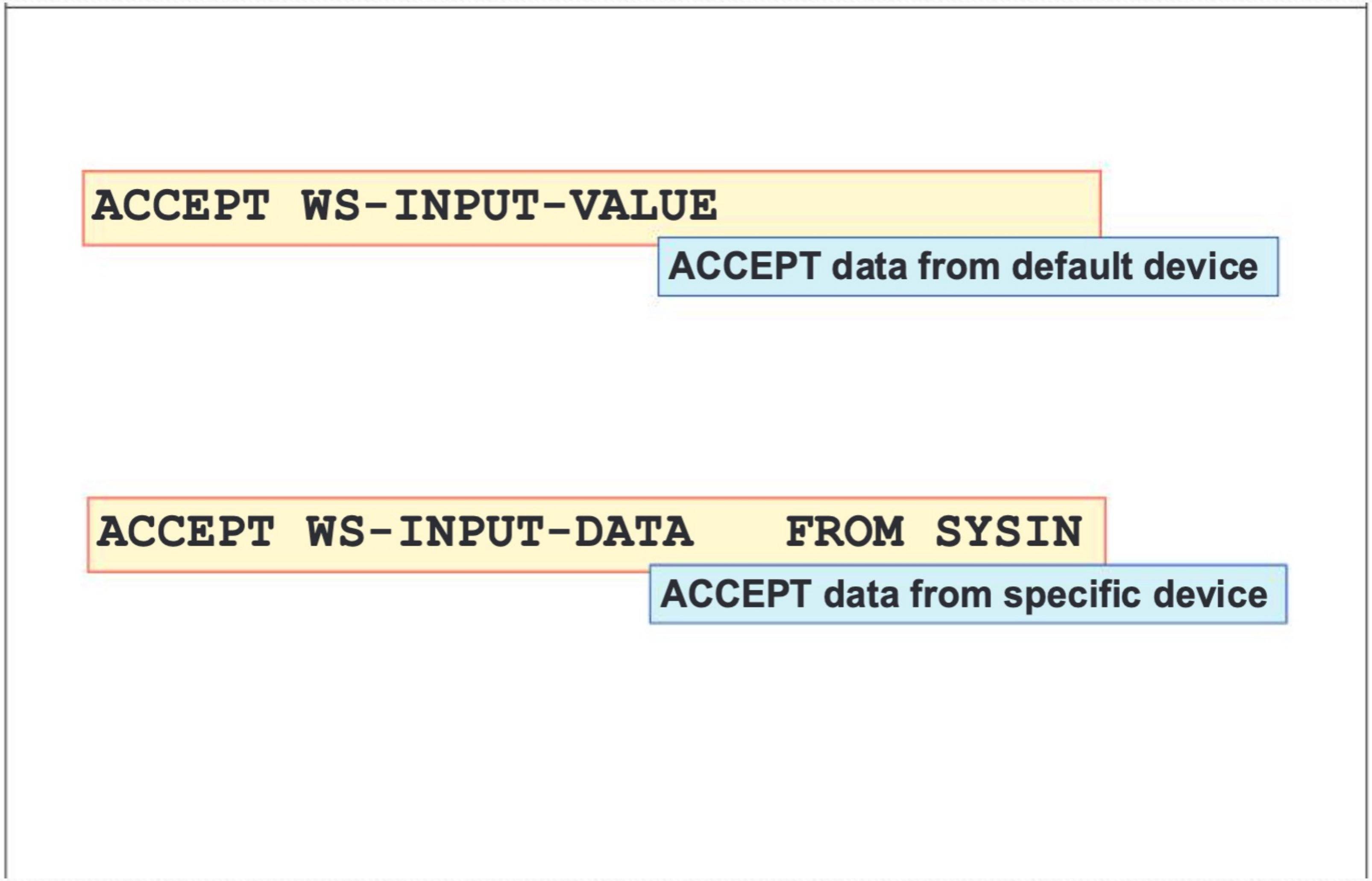
*Not commonly used*

```
ACCEPT WS-DATE           FROM DATE
ACCEPT WS-DAY            FROM DAY
ACCEPT WS-DAY-OF-WEEK    FROM DAY-OF-WEEK
ACCEPT WS-TIME           FROM TIME
```

*If the date is: 5th April 2013:*

| | | |
|---|---|---|
| WS-DATE | *will contain:* | 130405 |
| WS-DAY | *will contain:* | 13095 |
| WS-DAY-OF-WEEK | *will contain:* | 5 |
| WS-TIME | *will contain:* | 13031278 |

# Using ACCEPT for entering data



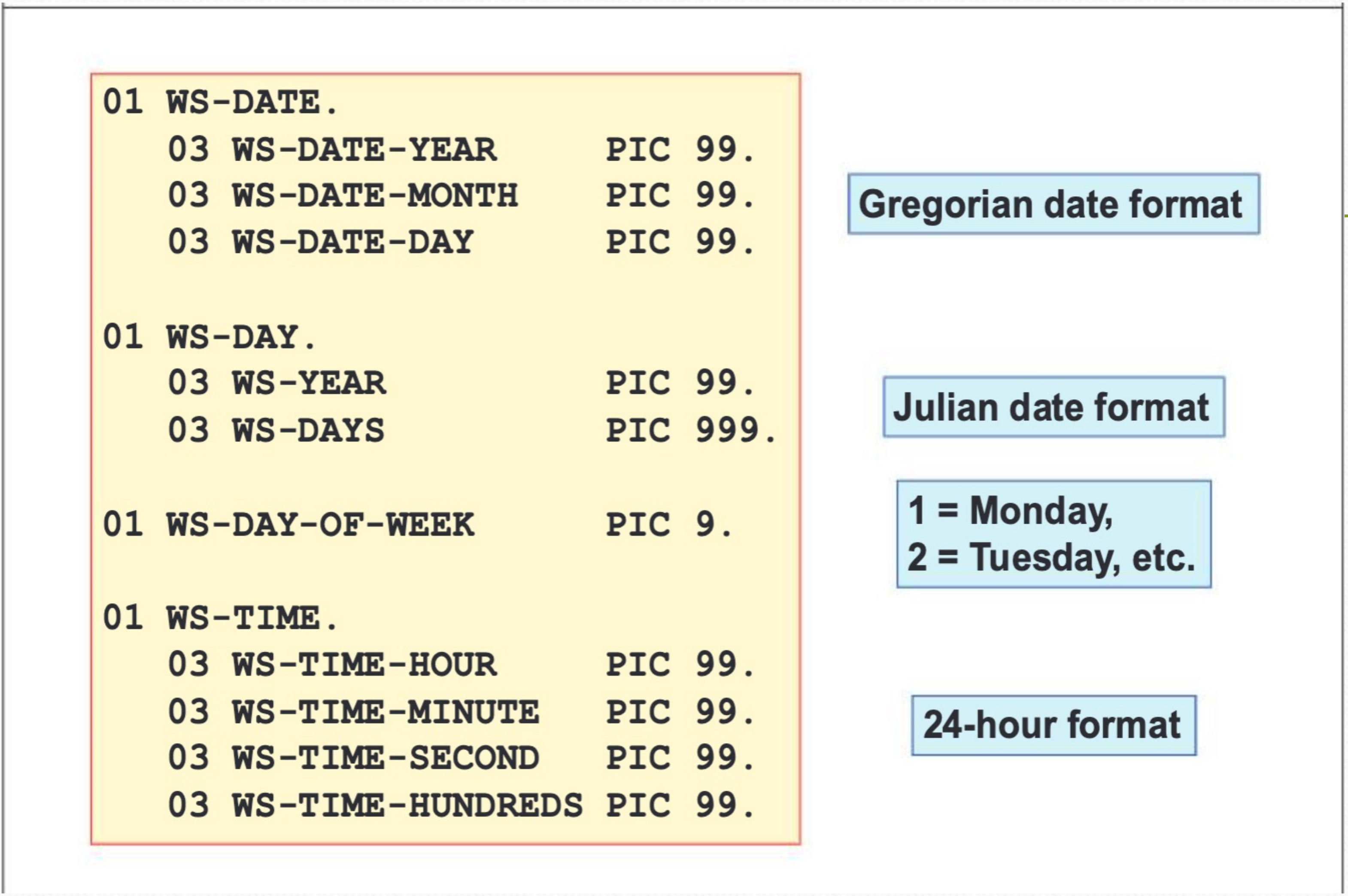
```
ACCEPT WS-INPUT-VALUE
```

ACCEPT data from default device

```
ACCEPT WS-INPUT-DATA   FROM SYSIN
```

ACCEPT data from specific device

# Data formats for ACCEPT



```
01 WS-DATE.
   03 WS-DATE-YEAR      PIC 99.
   03 WS-DATE-MONTH     PIC 99.
   03 WS-DATE-DAY       PIC 99.

01 WS-DAY.
   03 WS-YEAR           PIC 99.
   03 WS-DAYS           PIC 999.

01 WS-DAY-OF-WEEK       PIC 9.

01 WS-TIME.
   03 WS-TIME-HOUR      PIC 99.
   03 WS-TIME-MINUTE    PIC 99.
   03 WS-TIME-SECOND    PIC 99.
   03 WS-TIME-HUNDREDS  PIC 99.
```

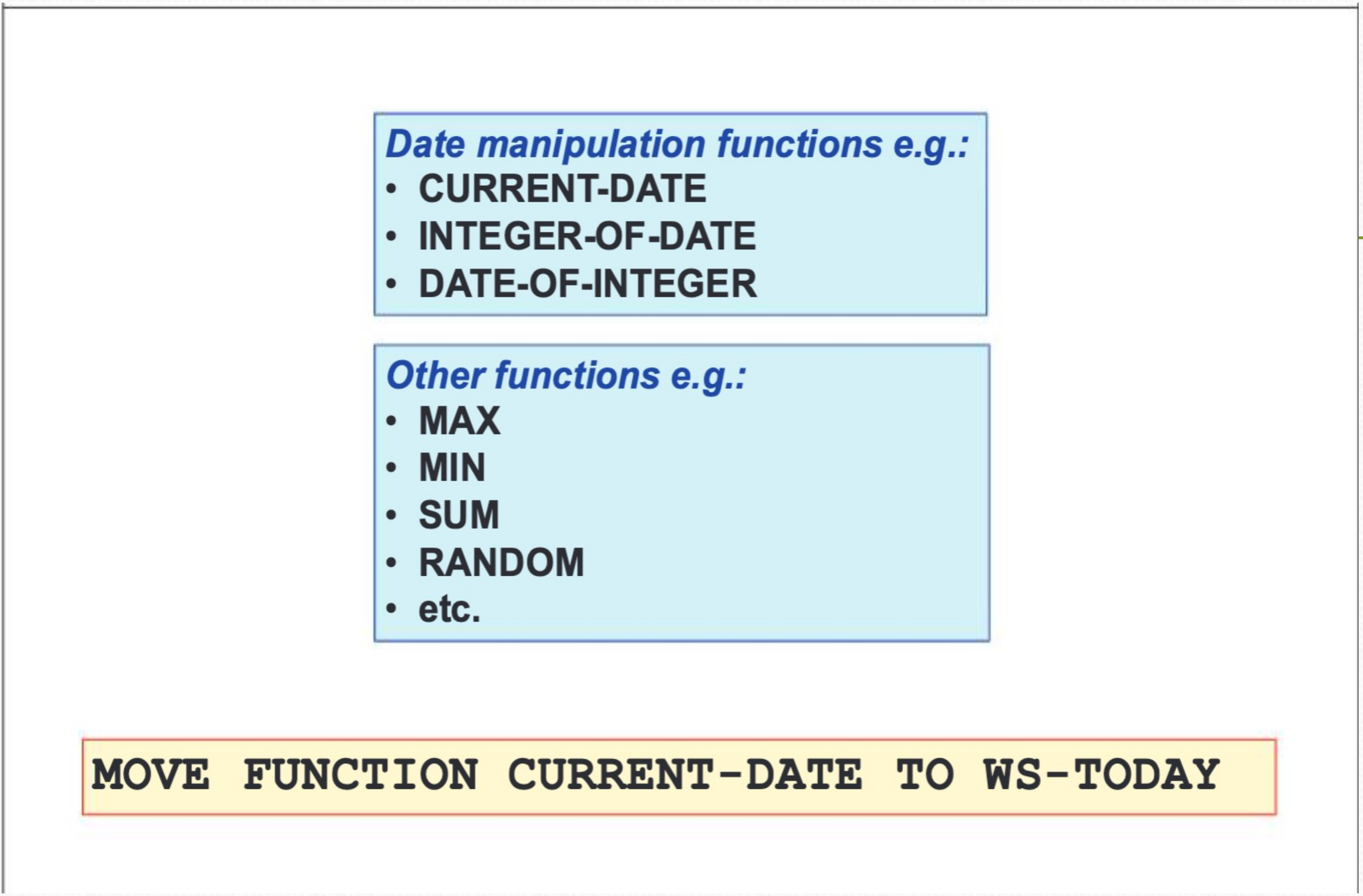
Gregorian date format

Julian date format

1 = Monday,
2 = Tuesday, etc.

24-hour format

# Intrinsic functions sample



*Date manipulation functions e.g.:*

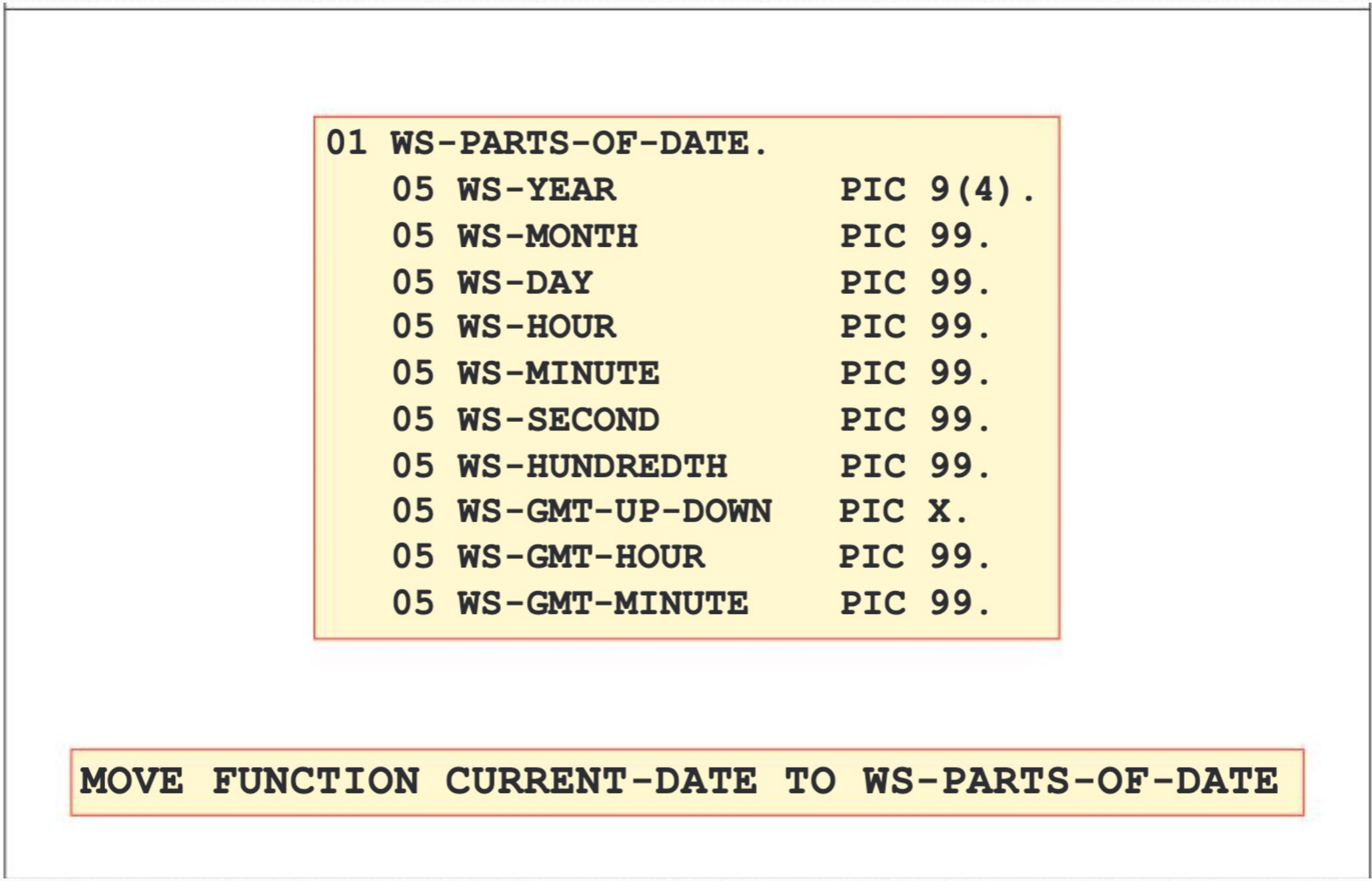
- CURRENT-DATE
- INTEGER-OF-DATE
- DATE-OF-INTEGER

*Other functions e.g.:*

- MAX
- MIN
- SUM
- RANDOM
- etc.

```
MOVE FUNCTION CURRENT-DATE TO WS-TODAY
```

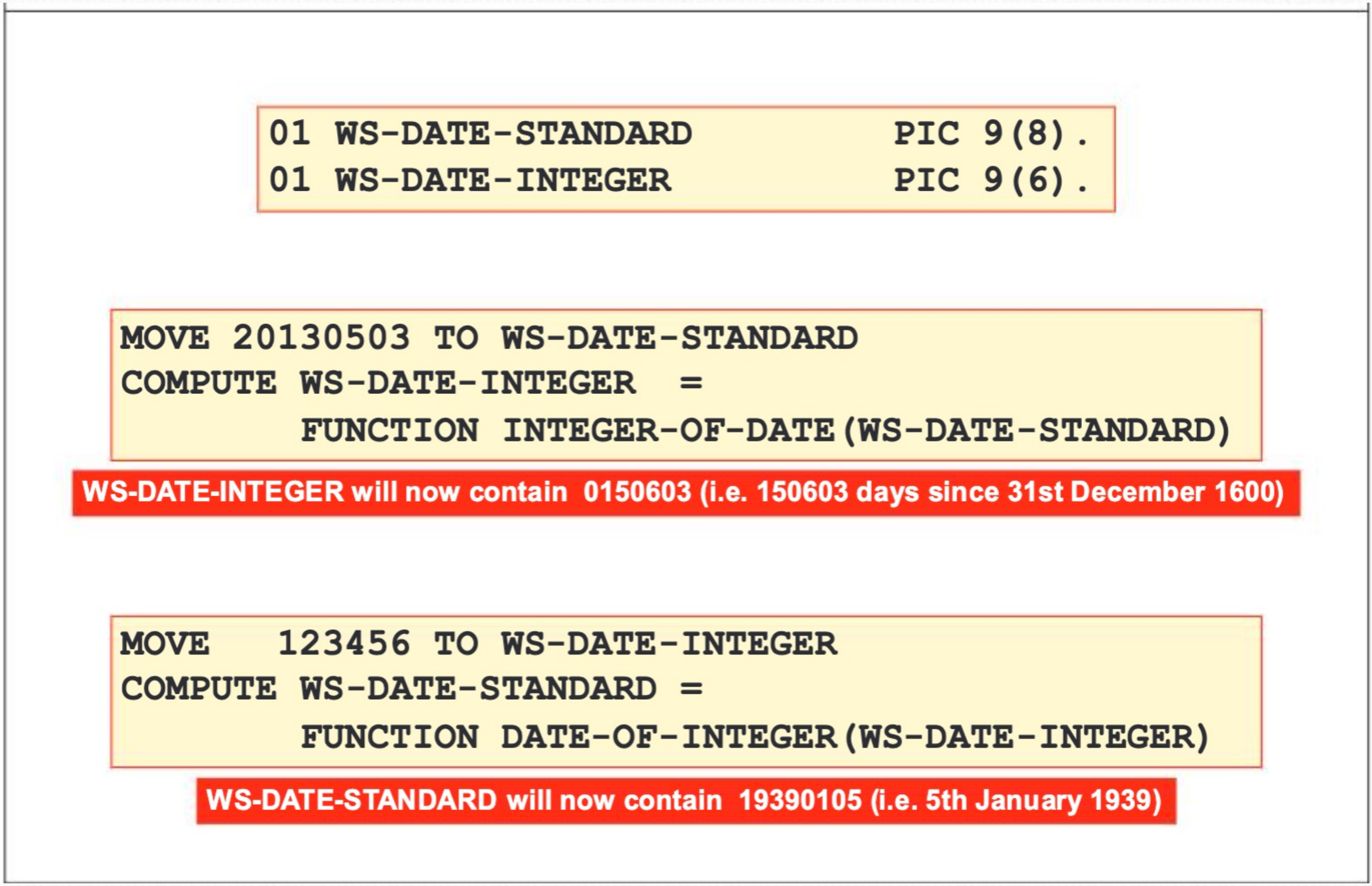
# CURRENT-DATE function



```
01 WS-PARTS-OF-DATE.
   05 WS-YEAR            PIC 9(4).
   05 WS-MONTH           PIC 99.
   05 WS-DAY             PIC 99.
   05 WS-HOUR            PIC 99.
   05 WS-MINUTE          PIC 99.
   05 WS-SECOND          PIC 99.
   05 WS-HUNDREDTH       PIC 99.
   05 WS-GMT-UP-DOWN     PIC X.
   05 WS-GMT-HOUR        PIC 99.
   05 WS-GMT-MINUTE      PIC 99.
```

```
MOVE FUNCTION CURRENT-DATE TO WS-PARTS-OF-DATE
```

# More DATE functions, YYYYMMDD format



```
01 WS-DATE-STANDARD          PIC 9(8).
01 WS-DATE-INTEGER           PIC 9(6).
```

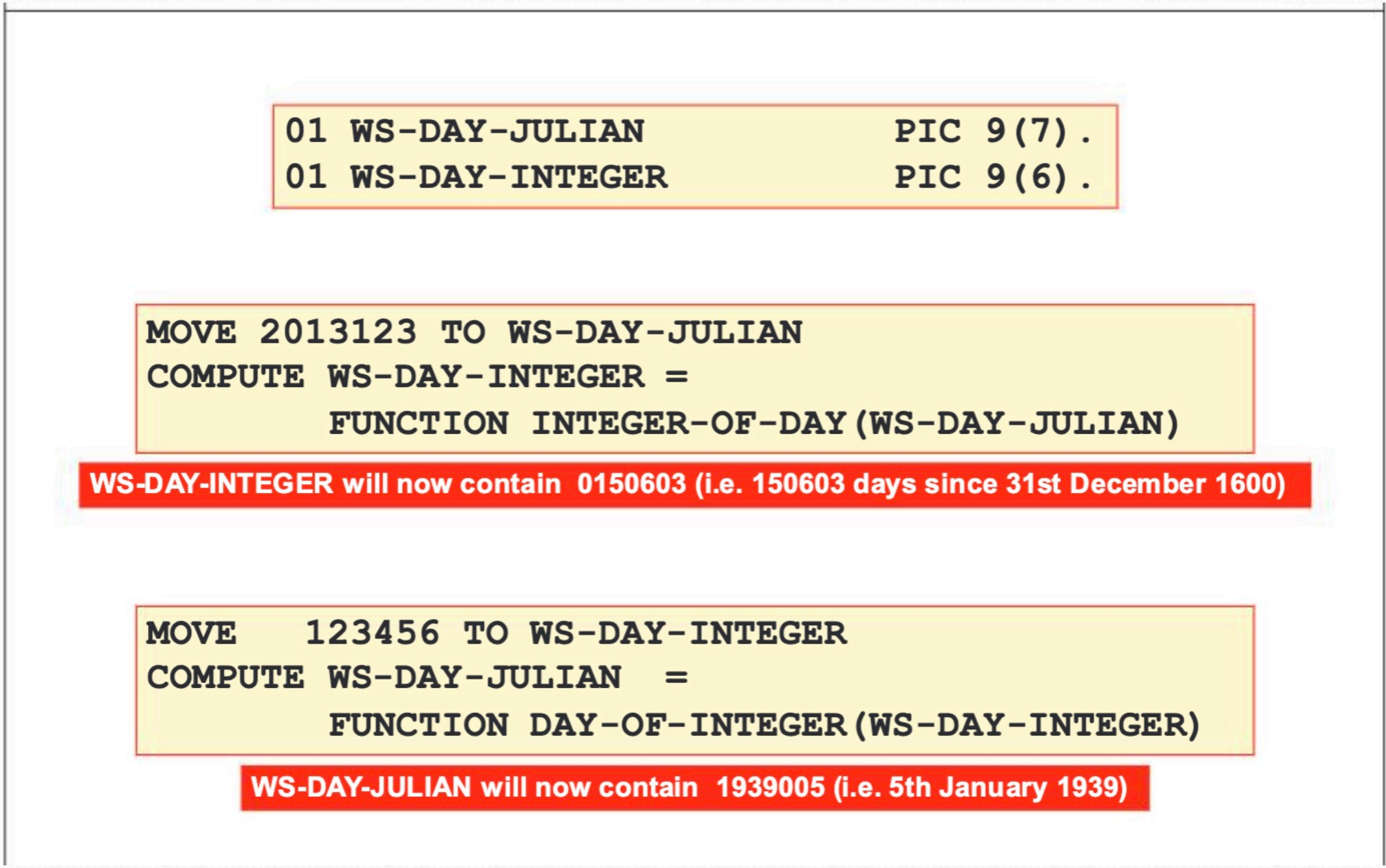
```
MOVE 20130503 TO WS-DATE-STANDARD
COMPUTE WS-DATE-INTEGER  =
        FUNCTION INTEGER-OF-DATE(WS-DATE-STANDARD)
```

**WS-DATE-INTEGER will now contain 0150603 (i.e. 150603 days since 31st December 1600)**

```
MOVE   123456 TO WS-DATE-INTEGER
COMPUTE WS-DATE-STANDARD =
        FUNCTION DATE-OF-INTEGER(WS-DATE-INTEGER)
```

**WS-DATE-STANDARD will now contain 19390105 (i.e. 5th January 1939)**

# DAY / JULIAN function



```
01 WS-DAY-JULIAN                PIC 9(7).
01 WS-DAY-INTEGER               PIC 9(6).
```

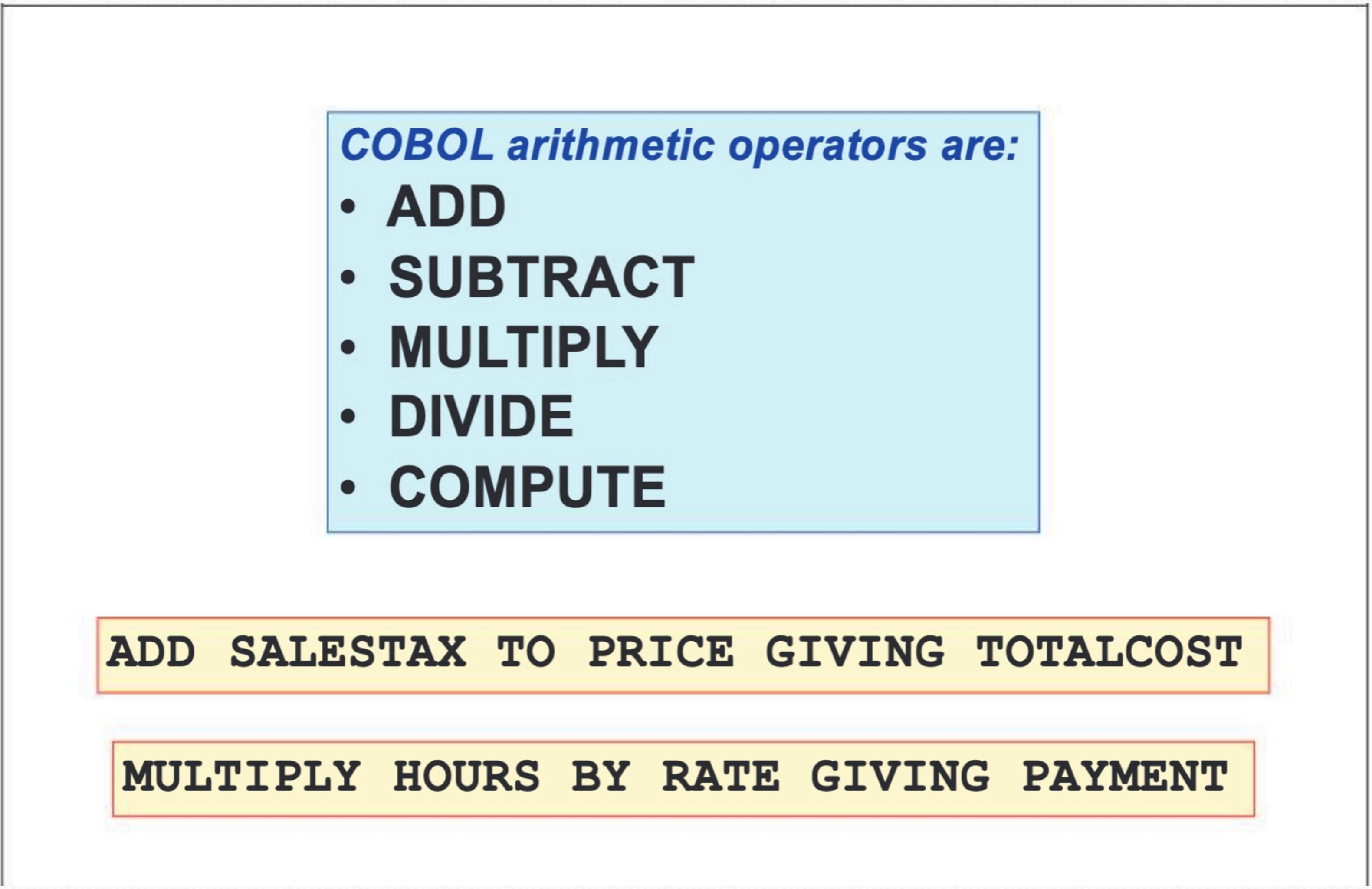
```
MOVE 2013123 TO WS-DAY-JULIAN
COMPUTE WS-DAY-INTEGER =
        FUNCTION INTEGER-OF-DAY(WS-DAY-JULIAN)
```

WS-DAY-INTEGER will now contain  0150603 (i.e. 150603 days since 31st December 1600)

```
MOVE   123456 TO WS-DAY-INTEGER
COMPUTE WS-DAY-JULIAN  =
        FUNCTION DAY-OF-INTEGER(WS-DAY-INTEGER)
```

WS-DAY-JULIAN will now contain  1939005 (i.e. 5th January 1939)

# Arithmetic operations



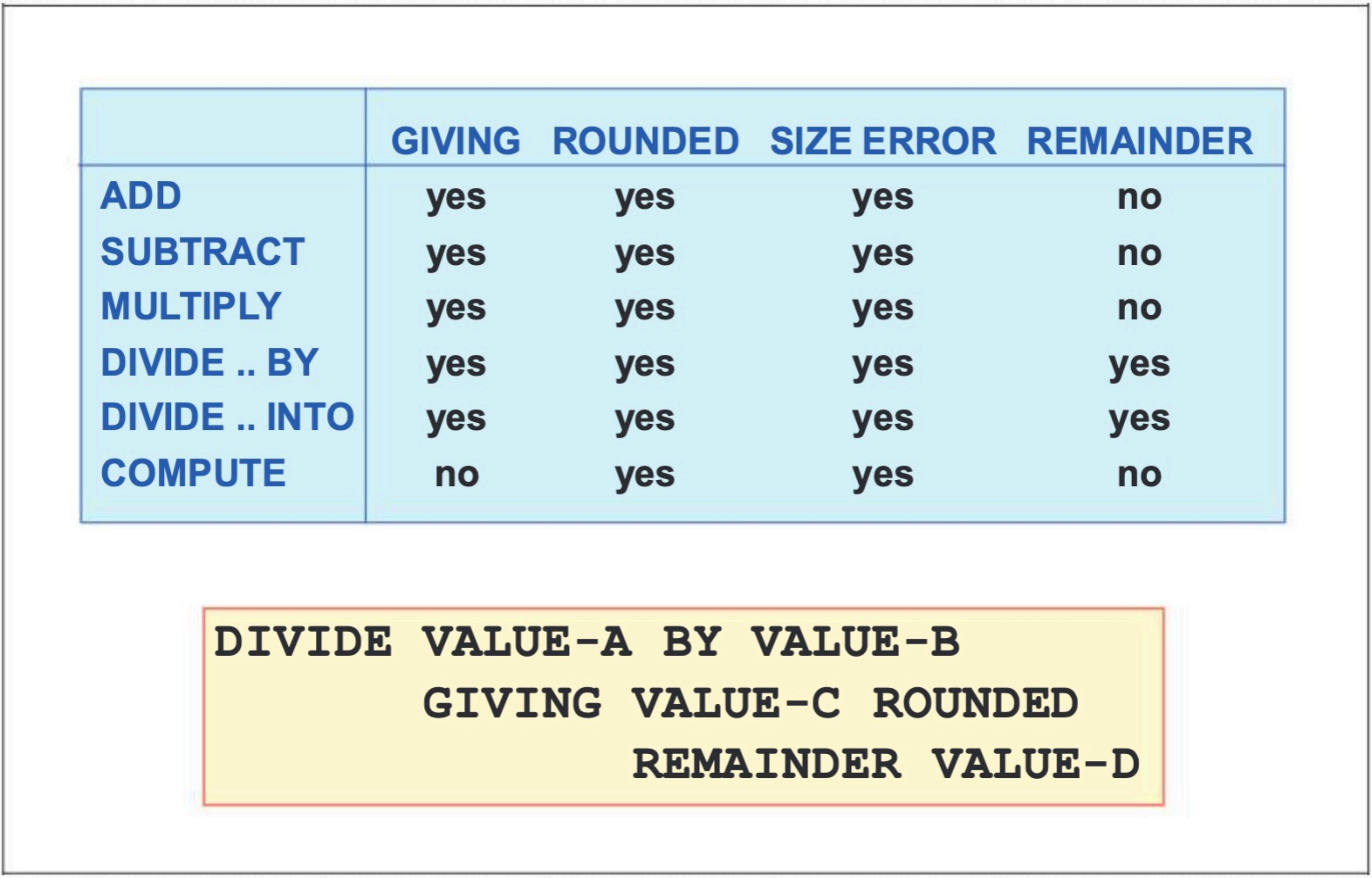
*COBOL arithmetic operators are:*

- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE
- COMPUTE

```
ADD SALESTAX TO PRICE GIVING TOTALCOST
```

```
MULTIPLY HOURS BY RATE GIVING PAYMENT
```

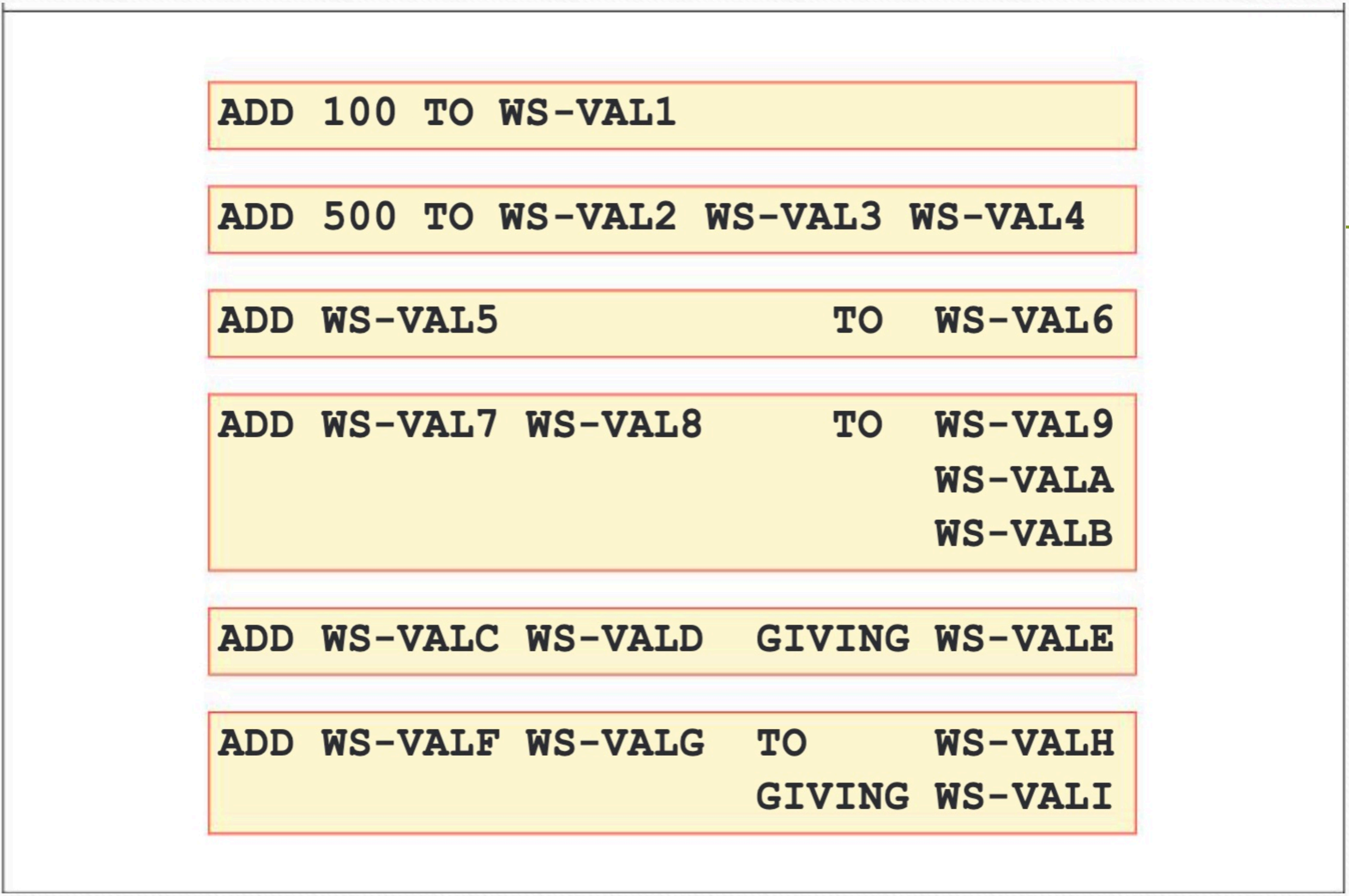
# More common options for those operations



| | GIVING | ROUNDED | SIZE ERROR | REMAINDER |
|---|---|---|---|---|
| ADD | yes | yes | yes | no |
| SUBTRACT | yes | yes | yes | no |
| MULTIPLY | yes | yes | yes | no |
| DIVIDE .. BY | yes | yes | yes | yes |
| DIVIDE .. INTO | yes | yes | yes | yes |
| COMPUTE | no | yes | yes | no |

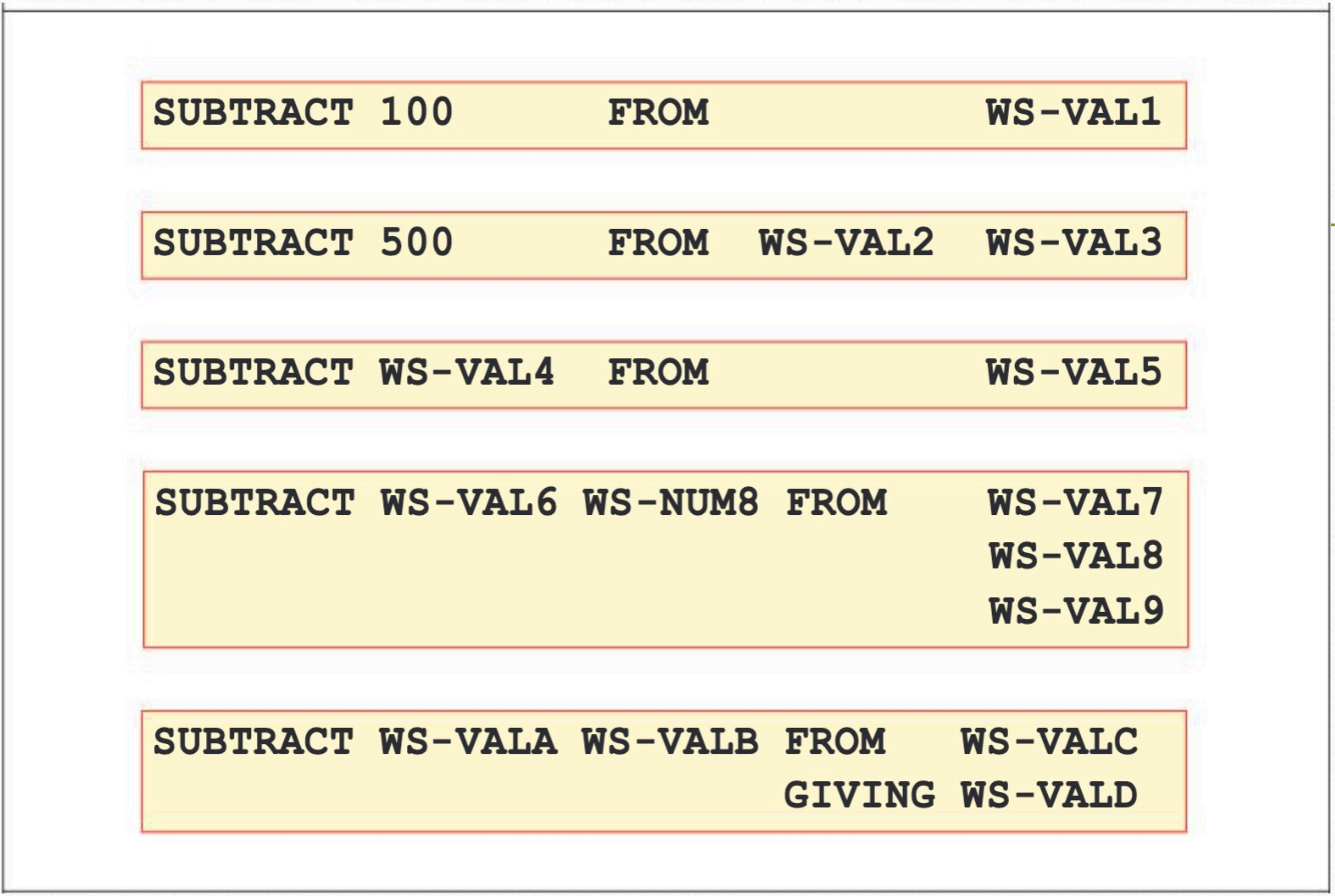
```
DIVIDE VALUE-A BY VALUE-B
       GIVING VALUE-C ROUNDED
              REMAINDER VALUE-D
```

# ADD options



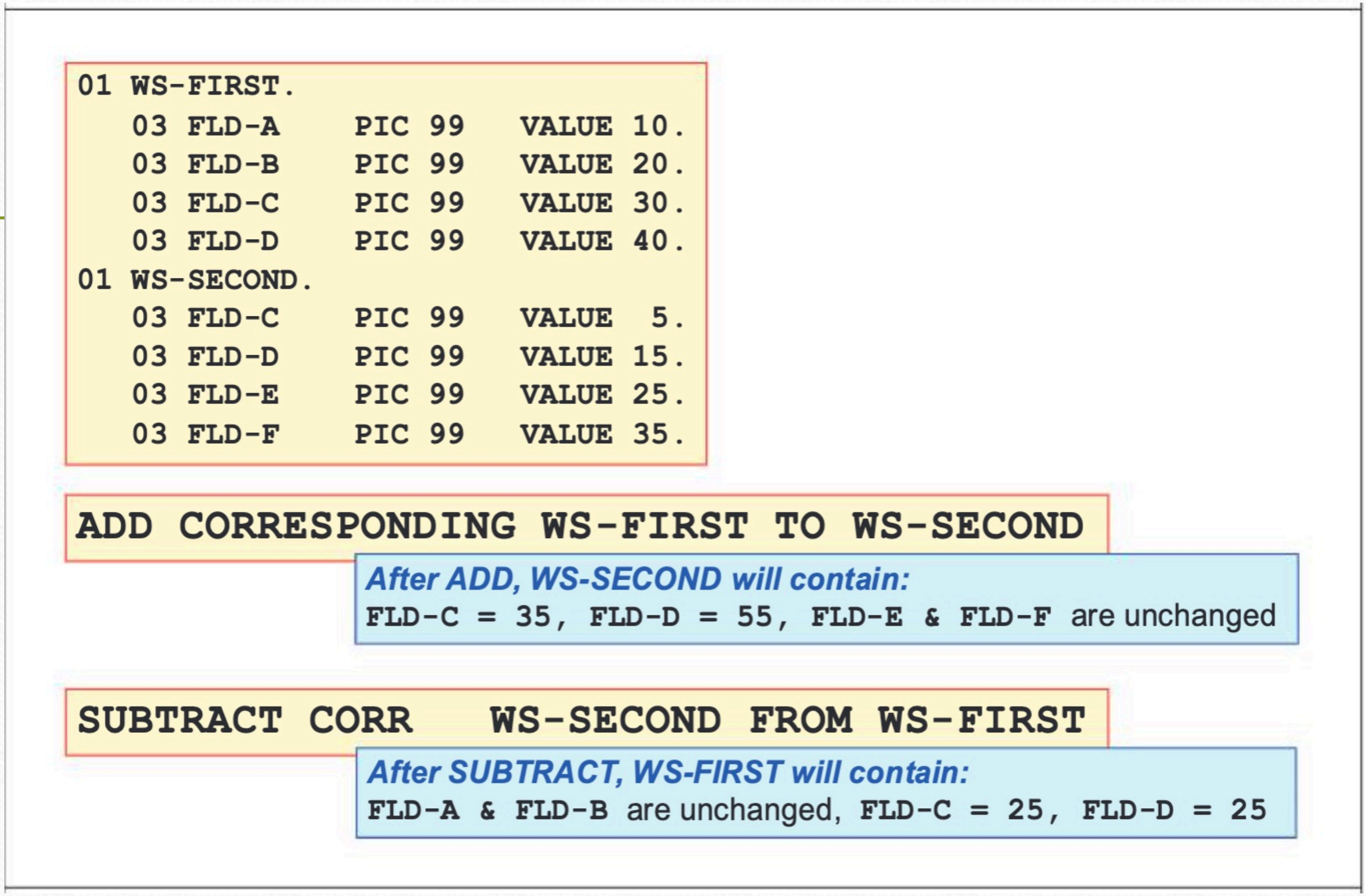
```
ADD 100 TO WS-VAL1

ADD 500 TO WS-VAL2 WS-VAL3 WS-VAL4

ADD WS-VAL5             TO  WS-VAL6

ADD WS-VAL7 WS-VAL8     TO  WS-VAL9
                            WS-VALA
                            WS-VALB

ADD WS-VALC WS-VALD  GIVING WS-VALE

ADD WS-VALF WS-VALG  TO     WS-VALH
                     GIVING WS-VALI
```

# SUBTRACT



```
SUBTRACT 100      FROM           WS-VAL1

SUBTRACT 500      FROM  WS-VAL2  WS-VAL3

SUBTRACT WS-VAL4  FROM           WS-VAL5

SUBTRACT WS-VAL6 WS-NUM8 FROM    WS-VAL7
                                 WS-VAL8
                                 WS-VAL9

SUBTRACT WS-VALA WS-VALB FROM   WS-VALC
                         GIVING WS-VALD
```

# The CORRESPONDING key word



```
01 WS-FIRST.
   03 FLD-A     PIC 99   VALUE 10.
   03 FLD-B     PIC 99   VALUE 20.
   03 FLD-C     PIC 99   VALUE 30.
   03 FLD-D     PIC 99   VALUE 40.
01 WS-SECOND.
   03 FLD-C     PIC 99   VALUE  5.
   03 FLD-D     PIC 99   VALUE 15.
   03 FLD-E     PIC 99   VALUE 25.
   03 FLD-F     PIC 99   VALUE 35.
```

```
ADD CORRESPONDING WS-FIRST TO WS-SECOND
```

***After ADD, WS-SECOND will contain:***

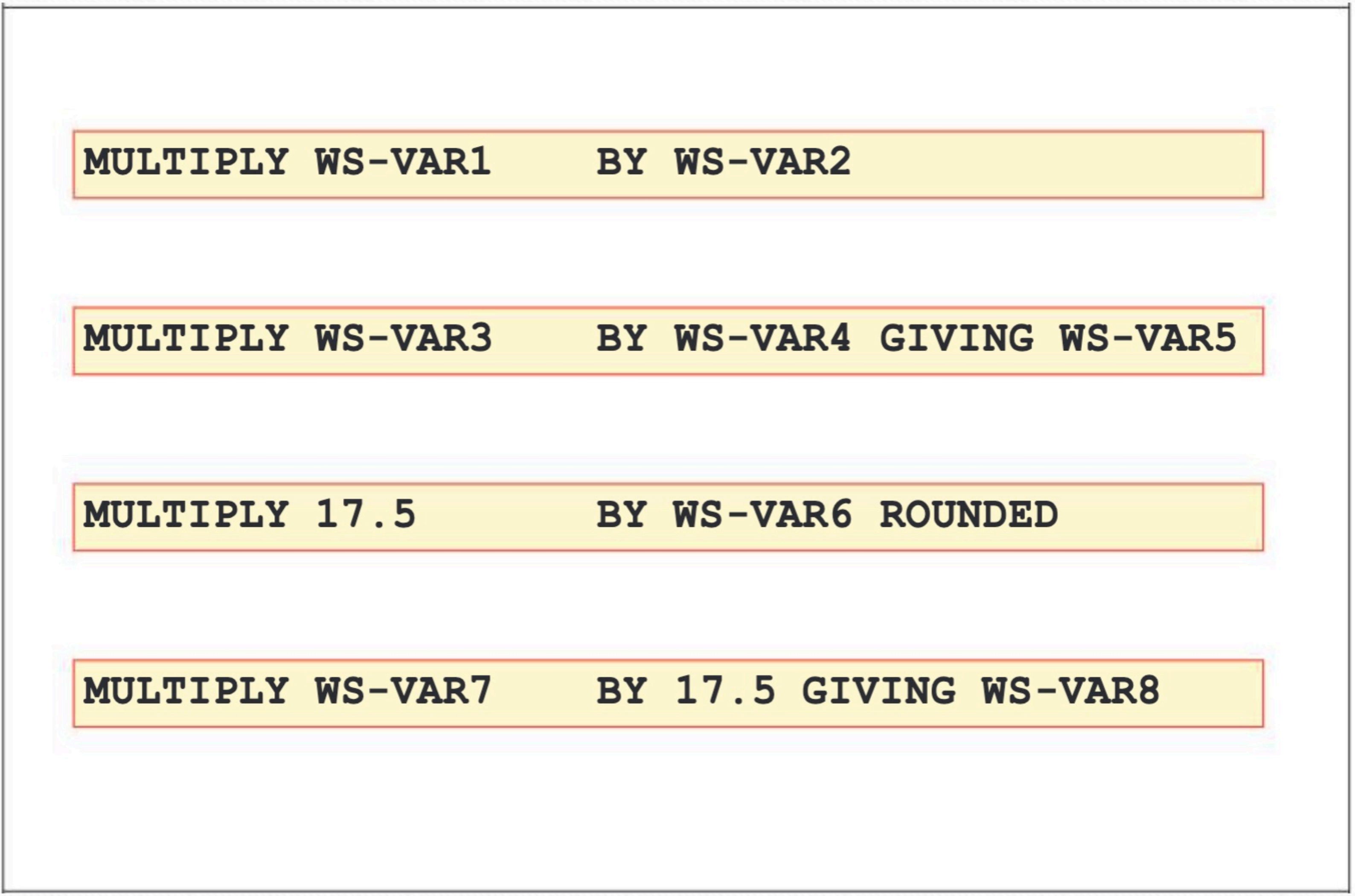
`FLD-C = 35, FLD-D = 55, FLD-E & FLD-F` are unchanged

```
SUBTRACT CORR   WS-SECOND FROM WS-FIRST
```

***After SUBTRACT, WS-FIRST will contain:***

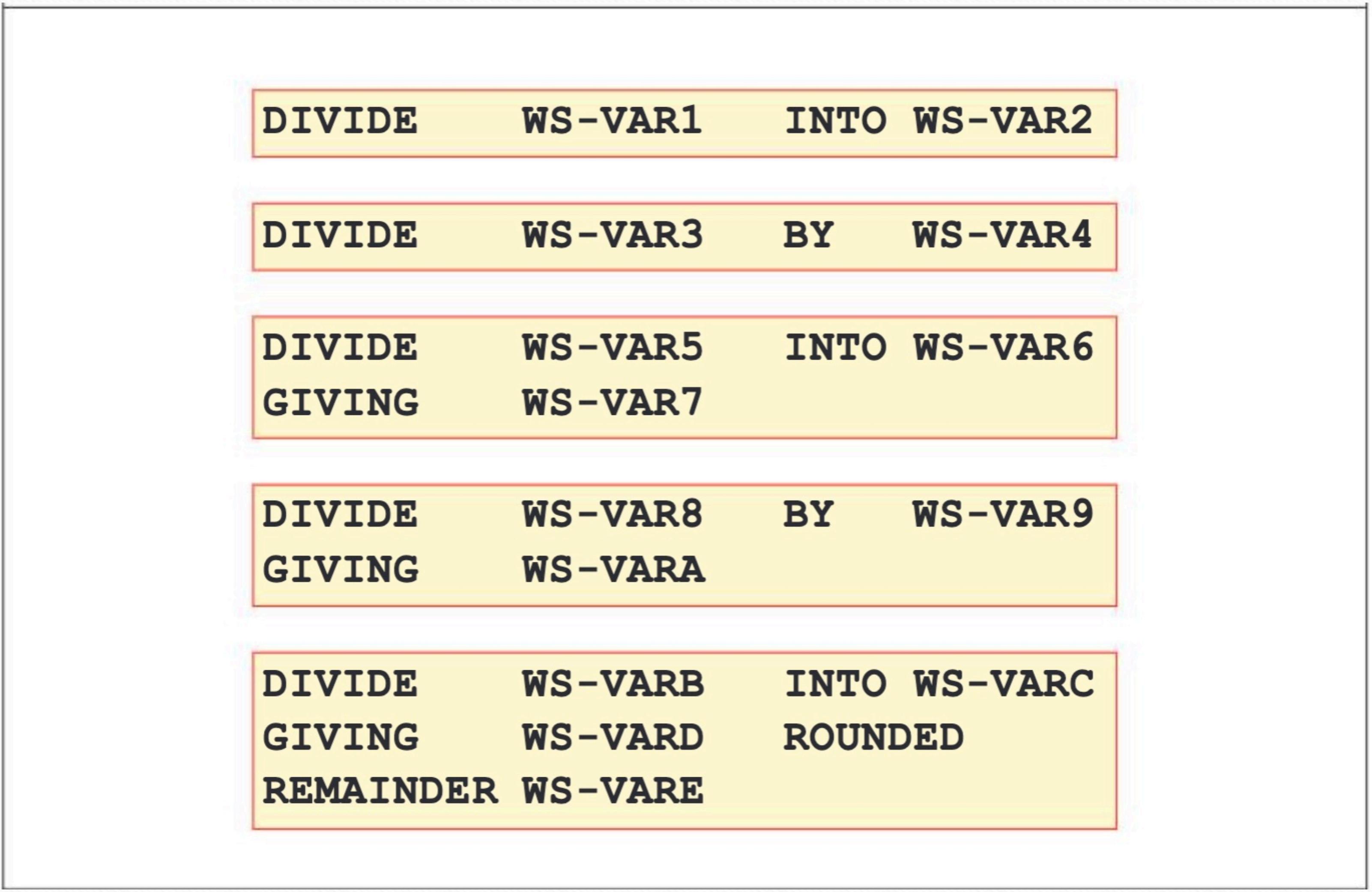
`FLD-A & FLD-B` are unchanged, `FLD-C = 25, FLD-D = 25`

# MULTIPLY



```
MULTIPLY WS-VAR1    BY WS-VAR2

MULTIPLY WS-VAR3    BY WS-VAR4 GIVING WS-VAR5

MULTIPLY 17.5       BY WS-VAR6 ROUNDED

MULTIPLY WS-VAR7    BY 17.5 GIVING WS-VAR8
```

# DIVIDE



```
DIVIDE    WS-VAR1   INTO WS-VAR2
```

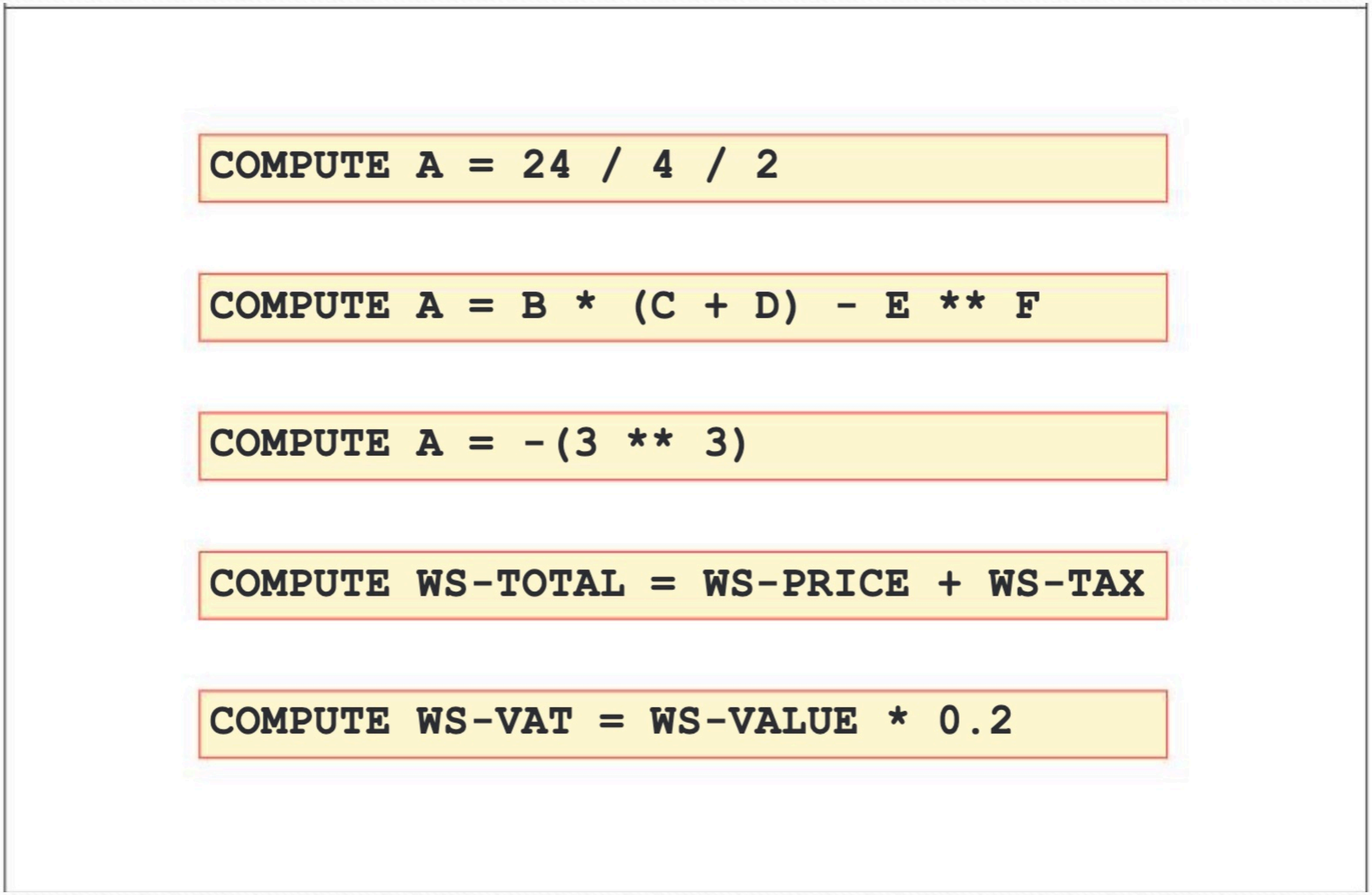
```
DIVIDE    WS-VAR3   BY   WS-VAR4
```

```
DIVIDE    WS-VAR5   INTO WS-VAR6
GIVING    WS-VAR7
```

```
DIVIDE    WS-VAR8   BY   WS-VAR9
GIVING    WS-VARA
```

```
DIVIDE    WS-VARB   INTO WS-VARC
GIVING    WS-VARD   ROUNDED
REMAINDER WS-VARE
```

# COMPUTE – A simpler option



```
COMPUTE A = 24 / 4 / 2
```

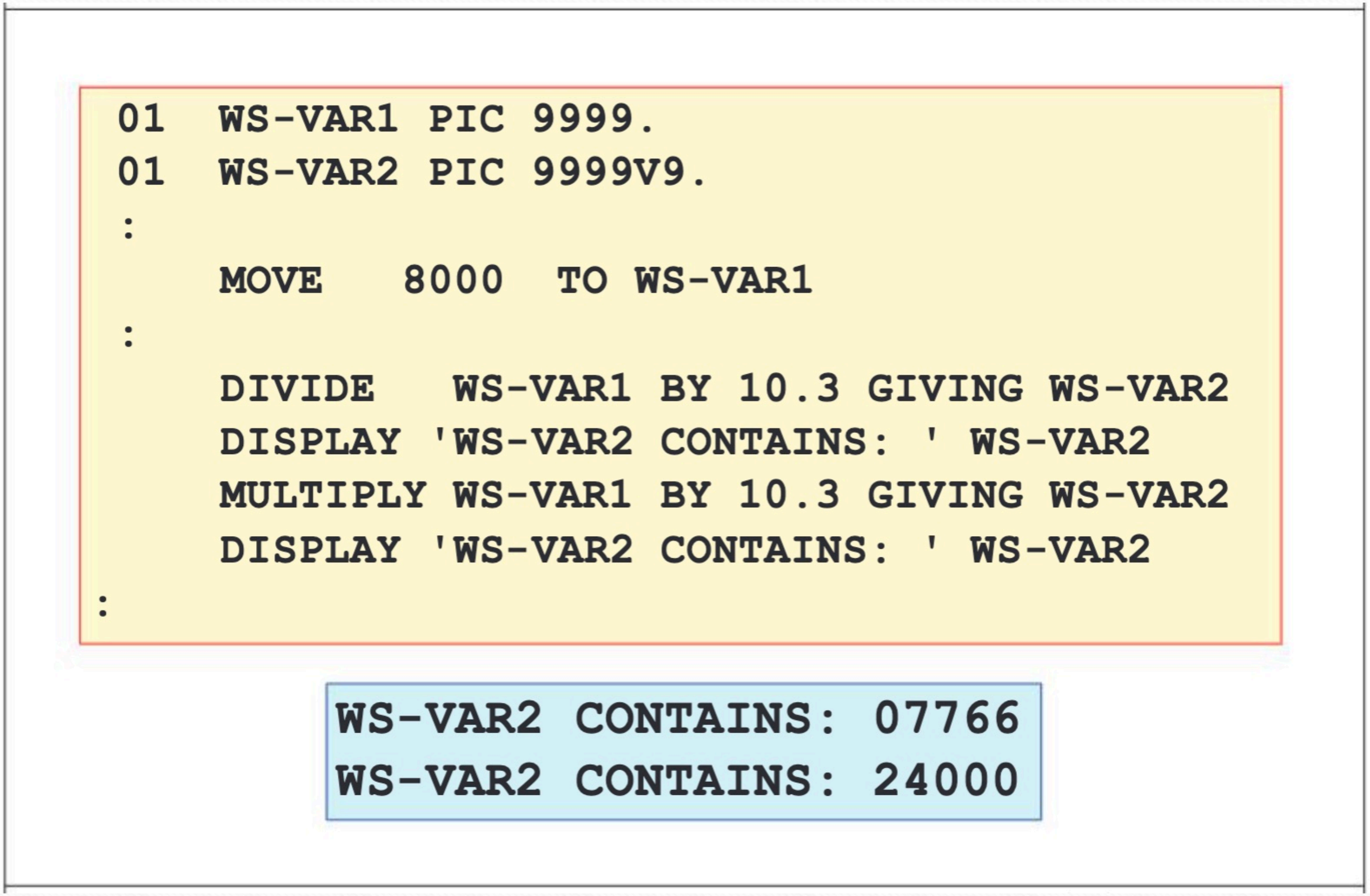
```
COMPUTE A = B * (C + D) - E ** F
```

```
COMPUTE A = -(3 ** 3)
```

```
COMPUTE WS-TOTAL = WS-PRICE + WS-TAX
```

```
COMPUTE WS-VAT = WS-VALUE * 0.2
```

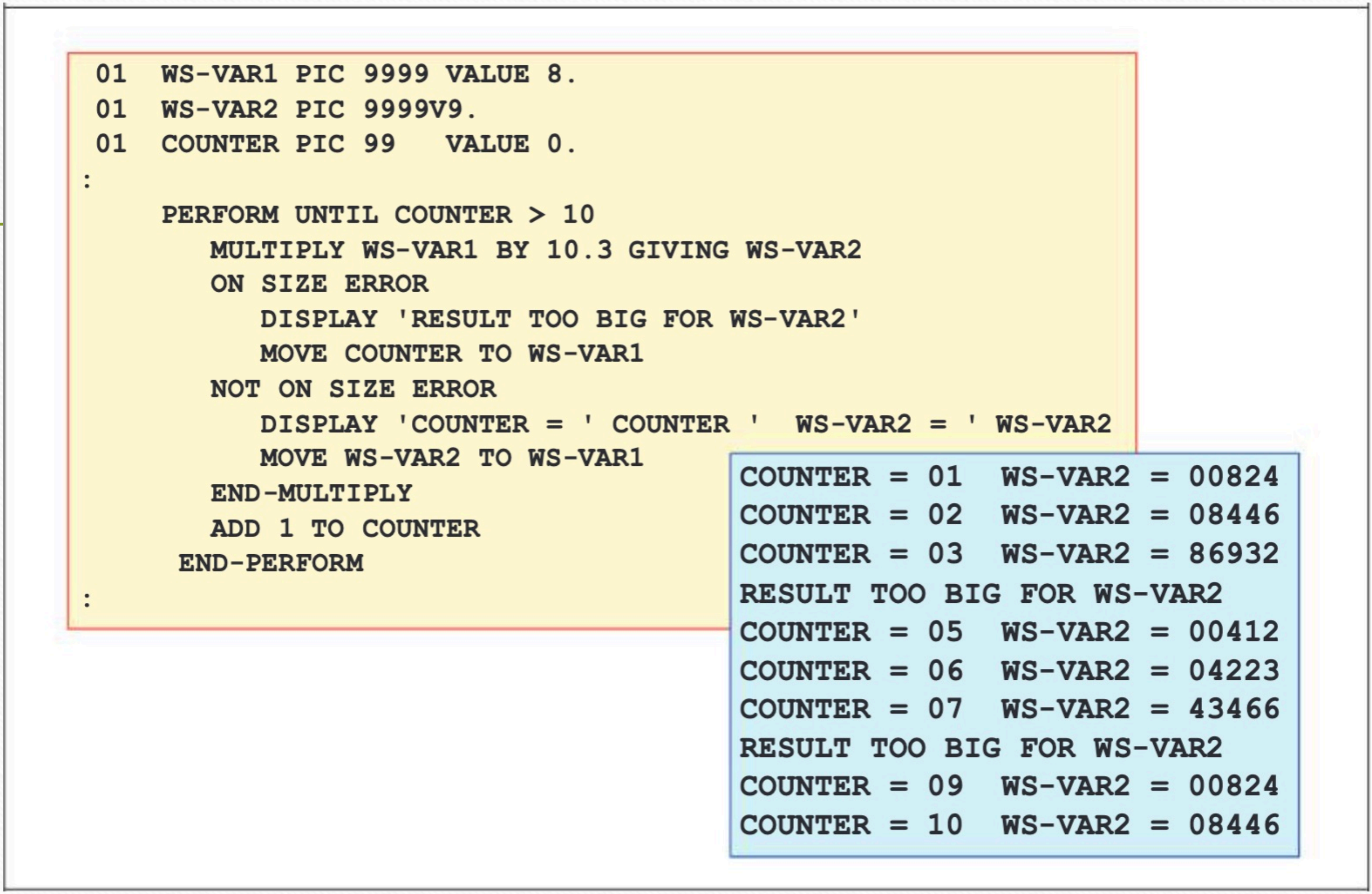
# SIZE errors



```
 01  WS-VAR1 PIC 9999.
 01  WS-VAR2 PIC 9999V9.
 :
     MOVE   8000  TO WS-VAR1
 :
     DIVIDE   WS-VAR1 BY 10.3 GIVING WS-VAR2
     DISPLAY 'WS-VAR2 CONTAINS: ' WS-VAR2
     MULTIPLY WS-VAR1 BY 10.3 GIVING WS-VAR2
     DISPLAY 'WS-VAR2 CONTAINS: ' WS-VAR2
:
```

```
WS-VAR2 CONTAINS: 07766
WS-VAR2 CONTAINS: 24000
```

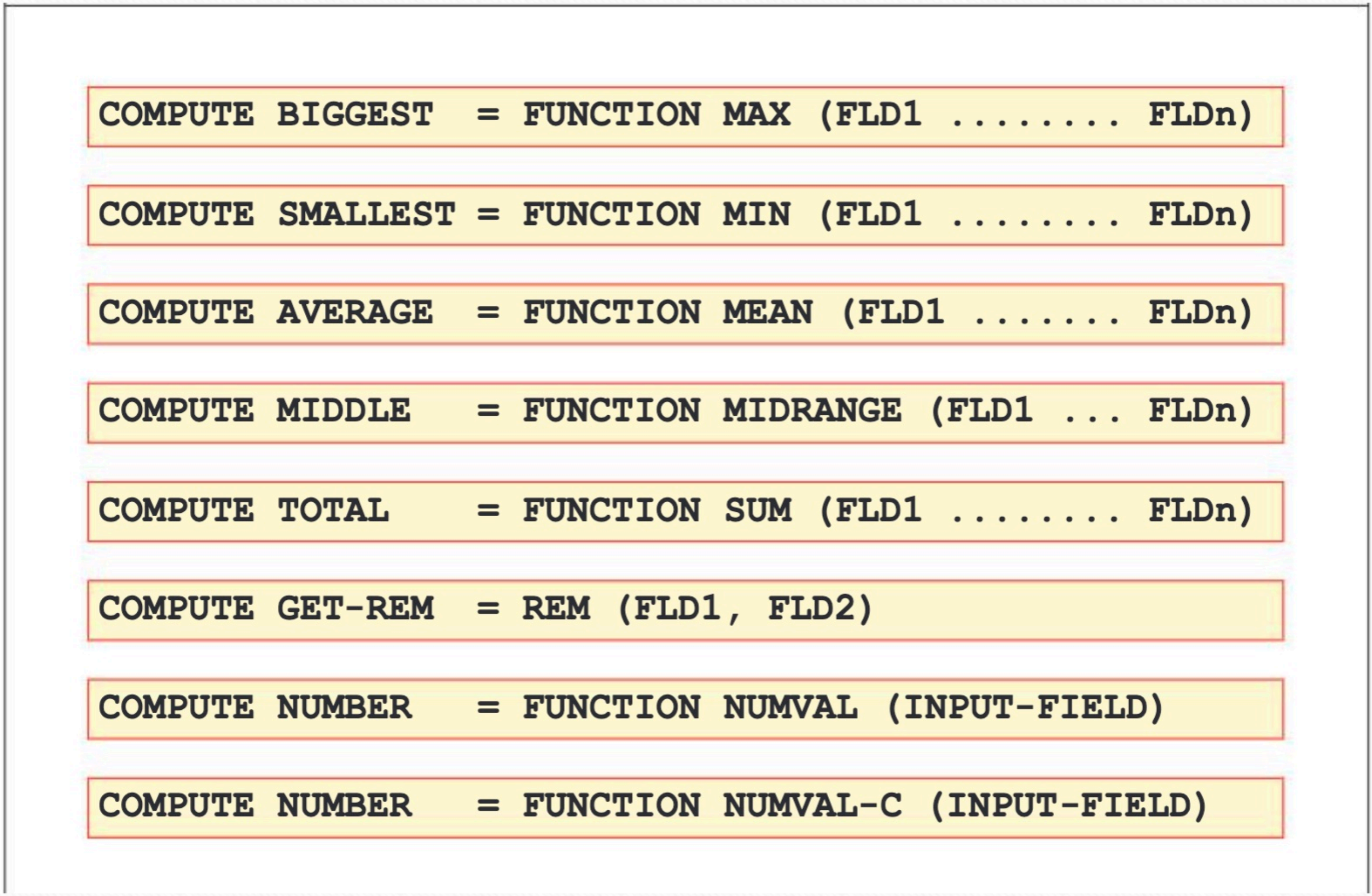
# ON SIZE error options



```
 01  WS-VAR1 PIC 9999 VALUE 8.
 01  WS-VAR2 PIC 9999V9.
 01  COUNTER PIC 99   VALUE 0.
:
     PERFORM UNTIL COUNTER > 10
        MULTIPLY WS-VAR1 BY 10.3 GIVING WS-VAR2
        ON SIZE ERROR
           DISPLAY 'RESULT TOO BIG FOR WS-VAR2'
           MOVE COUNTER TO WS-VAR1
        NOT ON SIZE ERROR
           DISPLAY 'COUNTER = ' COUNTER '  WS-VAR2 = ' WS-VAR2
           MOVE WS-VAR2 TO WS-VAR1
        END-MULTIPLY
        ADD 1 TO COUNTER
      END-PERFORM
:
```

```
COUNTER = 01  WS-VAR2 = 00824
COUNTER = 02  WS-VAR2 = 08446
COUNTER = 03  WS-VAR2 = 86932
RESULT TOO BIG FOR WS-VAR2
COUNTER = 05  WS-VAR2 = 00412
COUNTER = 06  WS-VAR2 = 04223
COUNTER = 07  WS-VAR2 = 43466
RESULT TOO BIG FOR WS-VAR2
COUNTER = 09  WS-VAR2 = 00824
COUNTER = 10  WS-VAR2 = 08446
```

# Other arithmetic functions



```
COMPUTE BIGGEST  = FUNCTION MAX (FLD1 ........ FLDn)
```

```
COMPUTE SMALLEST = FUNCTION MIN (FLD1 ........ FLDn)
```

```
COMPUTE AVERAGE  = FUNCTION MEAN (FLD1 ....... FLDn)
```

```
COMPUTE MIDDLE   = FUNCTION MIDRANGE (FLD1 ... FLDn)
```

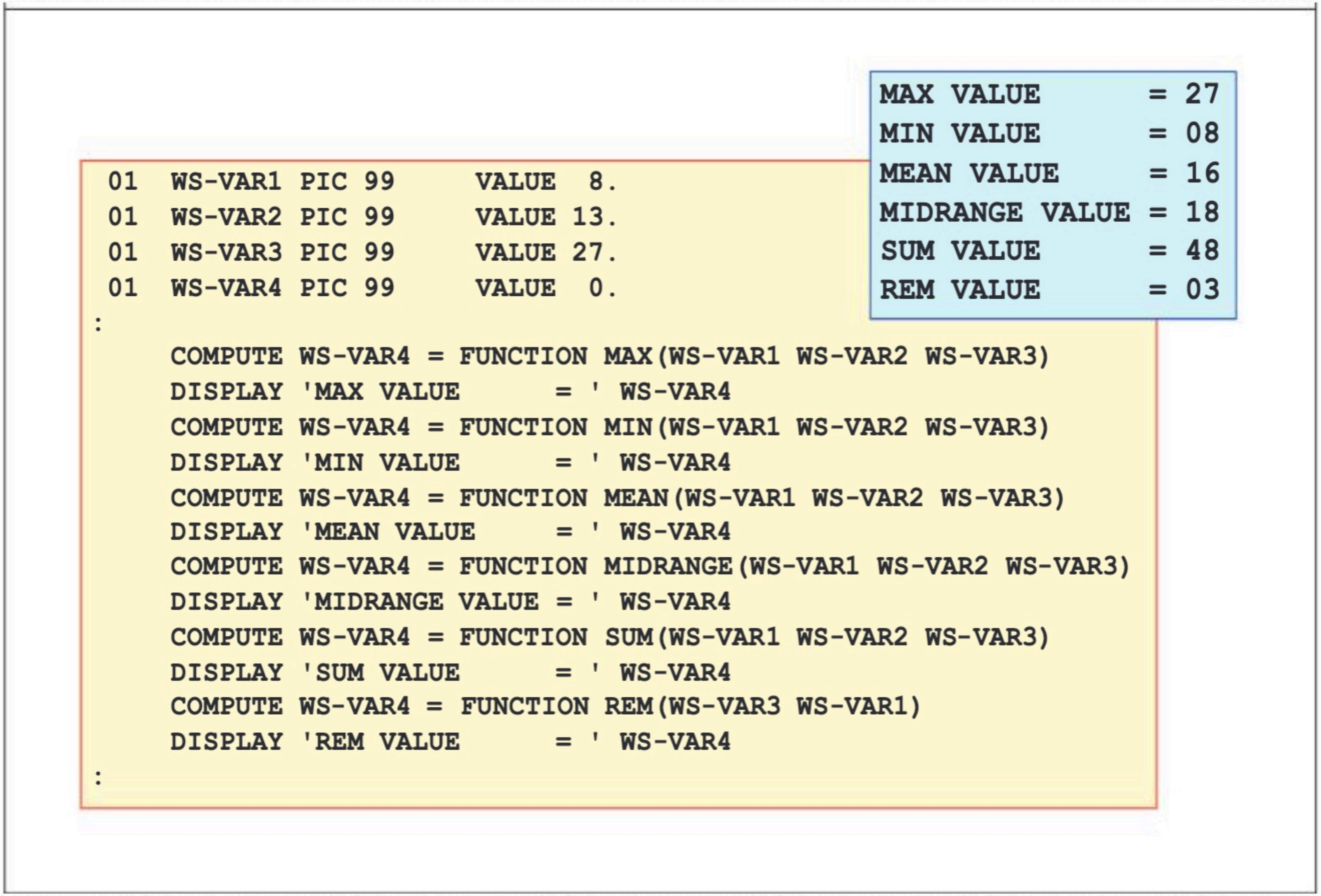
```
COMPUTE TOTAL    = FUNCTION SUM (FLD1 ........ FLDn)
```

```
COMPUTE GET-REM  = REM (FLD1, FLD2)
```

```
COMPUTE NUMBER   = FUNCTION NUMVAL (INPUT-FIELD)
```

```
COMPUTE NUMBER   = FUNCTION NUMVAL-C (INPUT-FIELD)
```

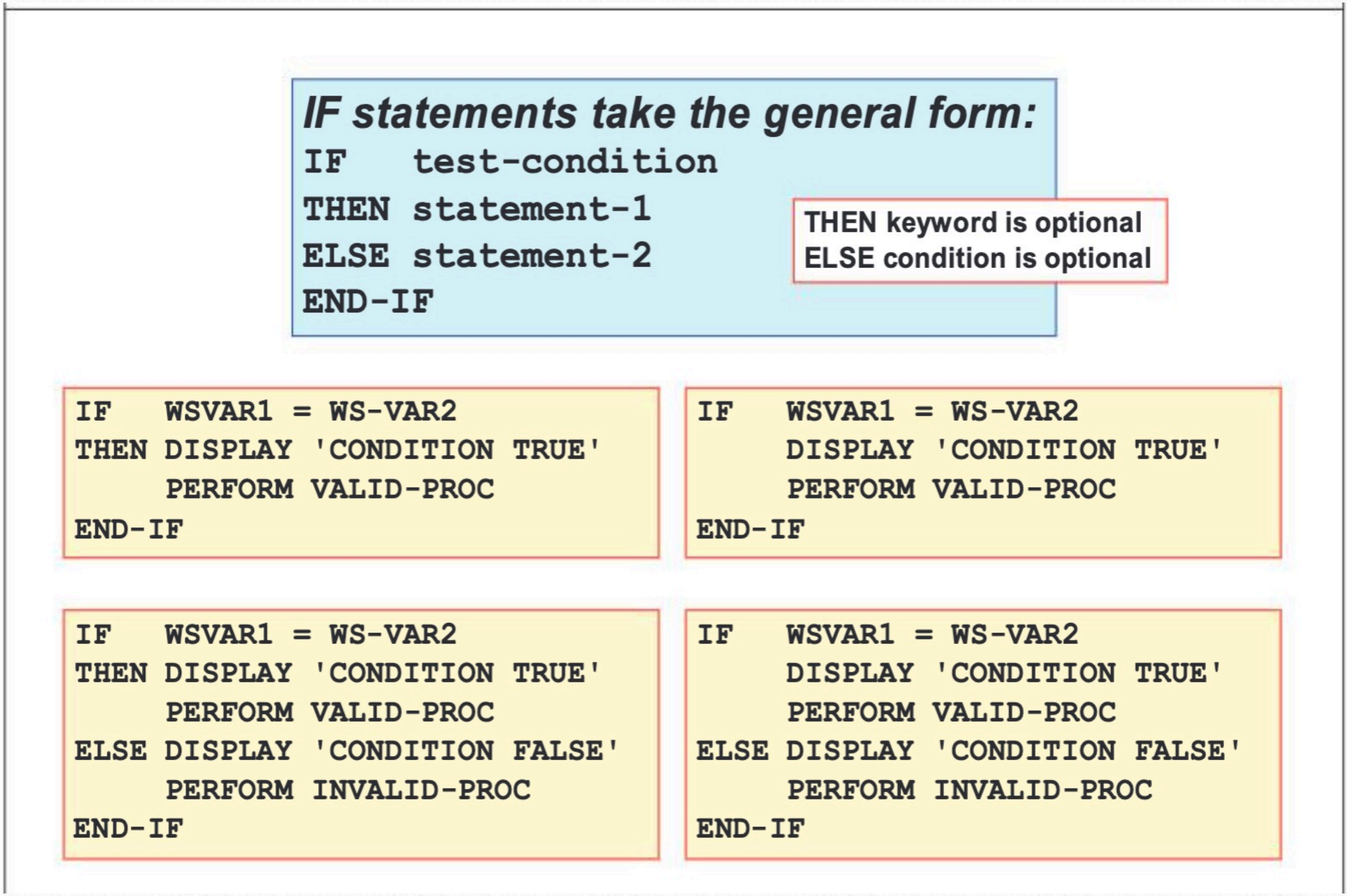
# Arithmetic usage examples



```
MAX VALUE      = 27
MIN VALUE      = 08
MEAN VALUE     = 16
MIDRANGE VALUE = 18
SUM VALUE      = 48
REM VALUE      = 03
```

```
 01  WS-VAR1 PIC 99     VALUE  8.
 01  WS-VAR2 PIC 99     VALUE 13.
 01  WS-VAR3 PIC 99     VALUE 27.
 01  WS-VAR4 PIC 99     VALUE  0.
:
     COMPUTE WS-VAR4 = FUNCTION MAX(WS-VAR1 WS-VAR2 WS-VAR3)
     DISPLAY 'MAX VALUE      = ' WS-VAR4
     COMPUTE WS-VAR4 = FUNCTION MIN(WS-VAR1 WS-VAR2 WS-VAR3)
     DISPLAY 'MIN VALUE      = ' WS-VAR4
     COMPUTE WS-VAR4 = FUNCTION MEAN(WS-VAR1 WS-VAR2 WS-VAR3)
     DISPLAY 'MEAN VALUE     = ' WS-VAR4
     COMPUTE WS-VAR4 = FUNCTION MIDRANGE(WS-VAR1 WS-VAR2 WS-VAR3)
     DISPLAY 'MIDRANGE VALUE = ' WS-VAR4
     COMPUTE WS-VAR4 = FUNCTION SUM(WS-VAR1 WS-VAR2 WS-VAR3)
     DISPLAY 'SUM VALUE      = ' WS-VAR4
     COMPUTE WS-VAR4 = FUNCTION REM(WS-VAR3 WS-VAR1)
     DISPLAY 'REM VALUE      = ' WS-VAR4
:
```

# Conditional - IF statement format



***IF statements take the general form:***

```
IF   test-condition
THEN statement-1
ELSE statement-2
END-IF
```

**THEN keyword is optional**
**ELSE condition is optional**

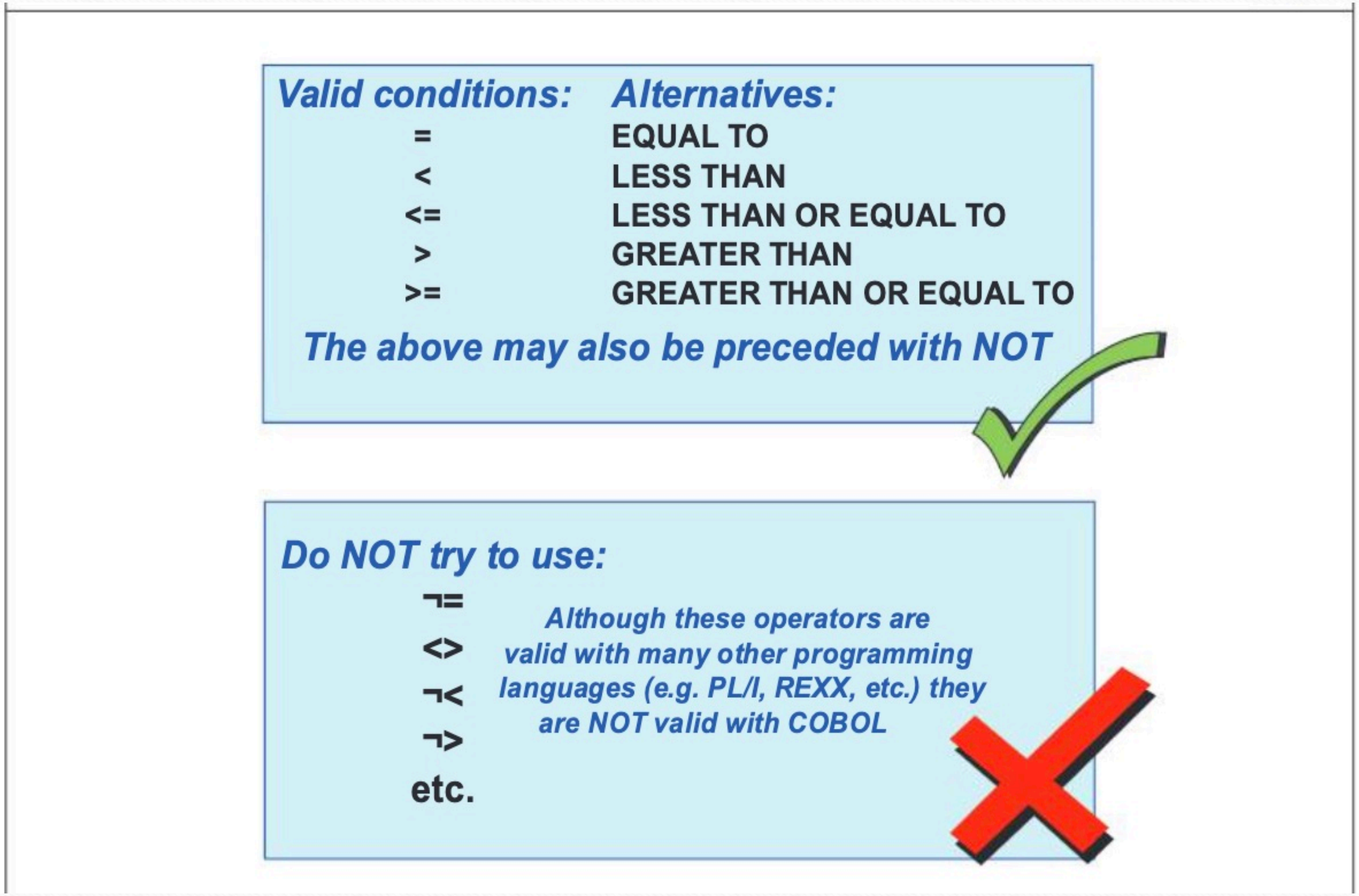
```
IF   WSVAR1 = WS-VAR2
THEN DISPLAY 'CONDITION TRUE'
     PERFORM VALID-PROC
END-IF
```

```
IF   WSVAR1 = WS-VAR2
     DISPLAY 'CONDITION TRUE'
     PERFORM VALID-PROC
END-IF
```

```
IF   WSVAR1 = WS-VAR2
THEN DISPLAY 'CONDITION TRUE'
     PERFORM VALID-PROC
ELSE DISPLAY 'CONDITION FALSE'
     PERFORM INVALID-PROC
END-IF
```

```
IF   WSVAR1 = WS-VAR2
     DISPLAY 'CONDITION TRUE'
     PERFORM VALID-PROC
ELSE DISPLAY 'CONDITION FALSE'
     PERFORM INVALID-PROC
END-IF
```

# Relational operators



| Valid conditions: | Alternatives: |
|---|---|
| = | EQUAL TO |
| < | LESS THAN |
| <= | LESS THAN OR EQUAL TO |
| > | GREATER THAN |
| >= | GREATER THAN OR EQUAL TO |

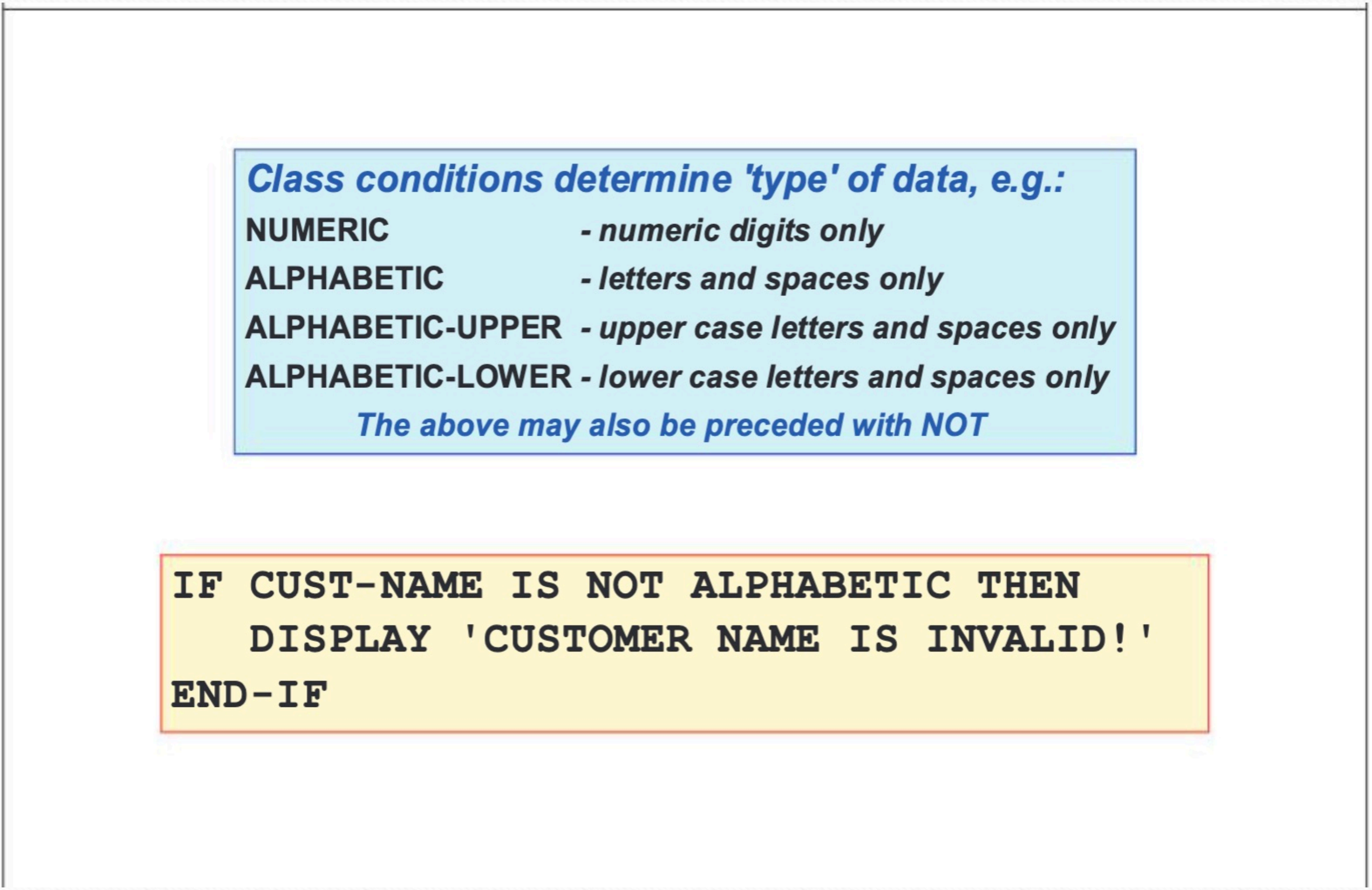
*The above may also be preceded with NOT*

*Do NOT try to use:*

¬=
<>
¬<
¬>
etc.

*Although these operators are valid with many other programming languages (e.g. PL/I, REXX, etc.) they are NOT valid with COBOL*

# Class conditions



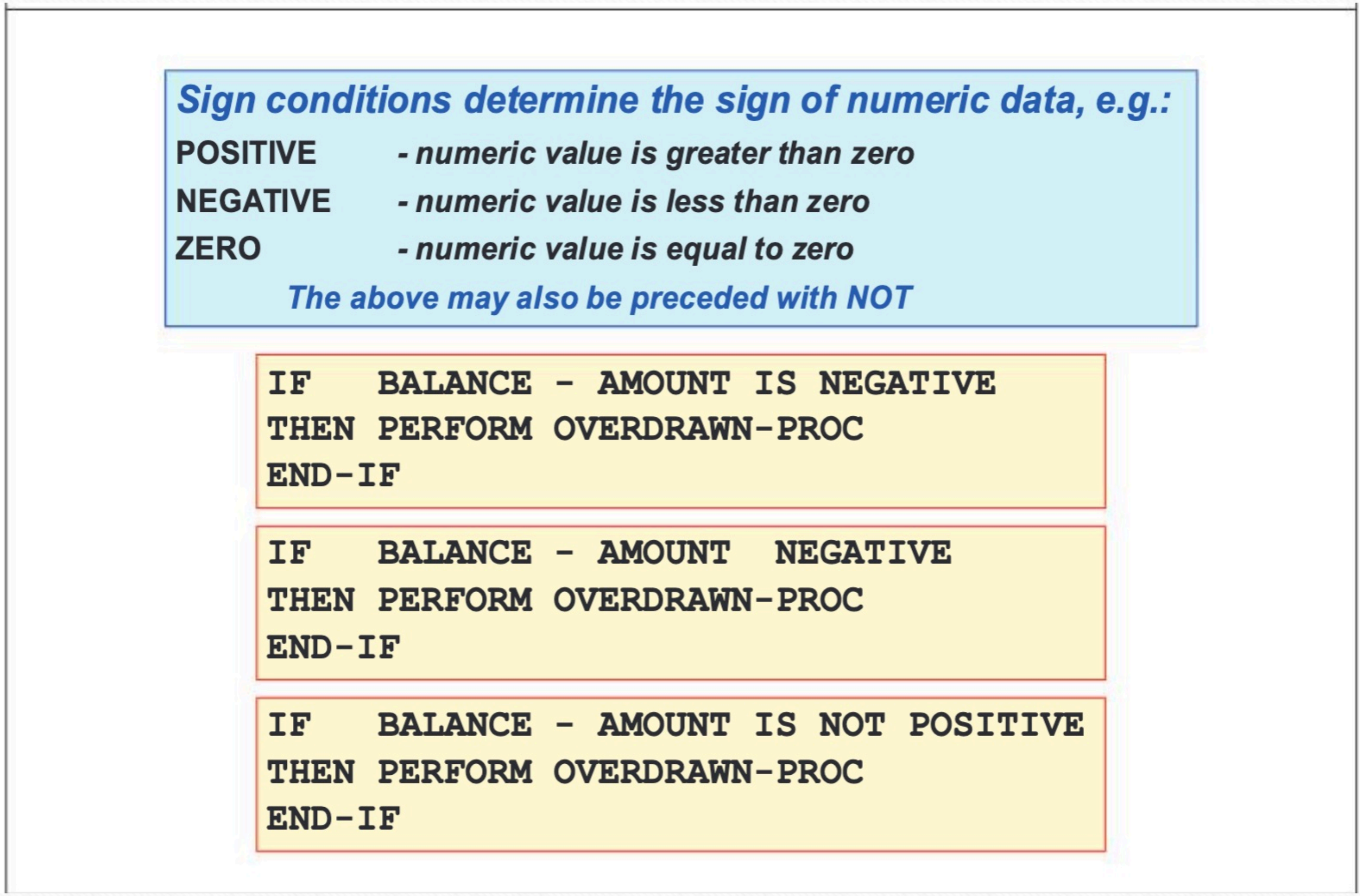
*Class conditions determine 'type' of data, e.g.:*

| | |
|---|---|
| NUMERIC | - *numeric digits only* |
| ALPHABETIC | - *letters and spaces only* |
| ALPHABETIC-UPPER | - *upper case letters and spaces only* |
| ALPHABETIC-LOWER | - *lower case letters and spaces only* |

*The above may also be preceded with NOT*

```
IF CUST-NAME IS NOT ALPHABETIC THEN
   DISPLAY 'CUSTOMER NAME IS INVALID!'
END-IF
```

# Sign conditions



***Sign conditions determine the sign of numeric data, e.g.:***

| | |
|---|---|
| **POSITIVE** | ***- numeric value is greater than zero*** |
| **NEGATIVE** | ***- numeric value is less than zero*** |
| **ZERO** | ***- numeric value is equal to zero*** |

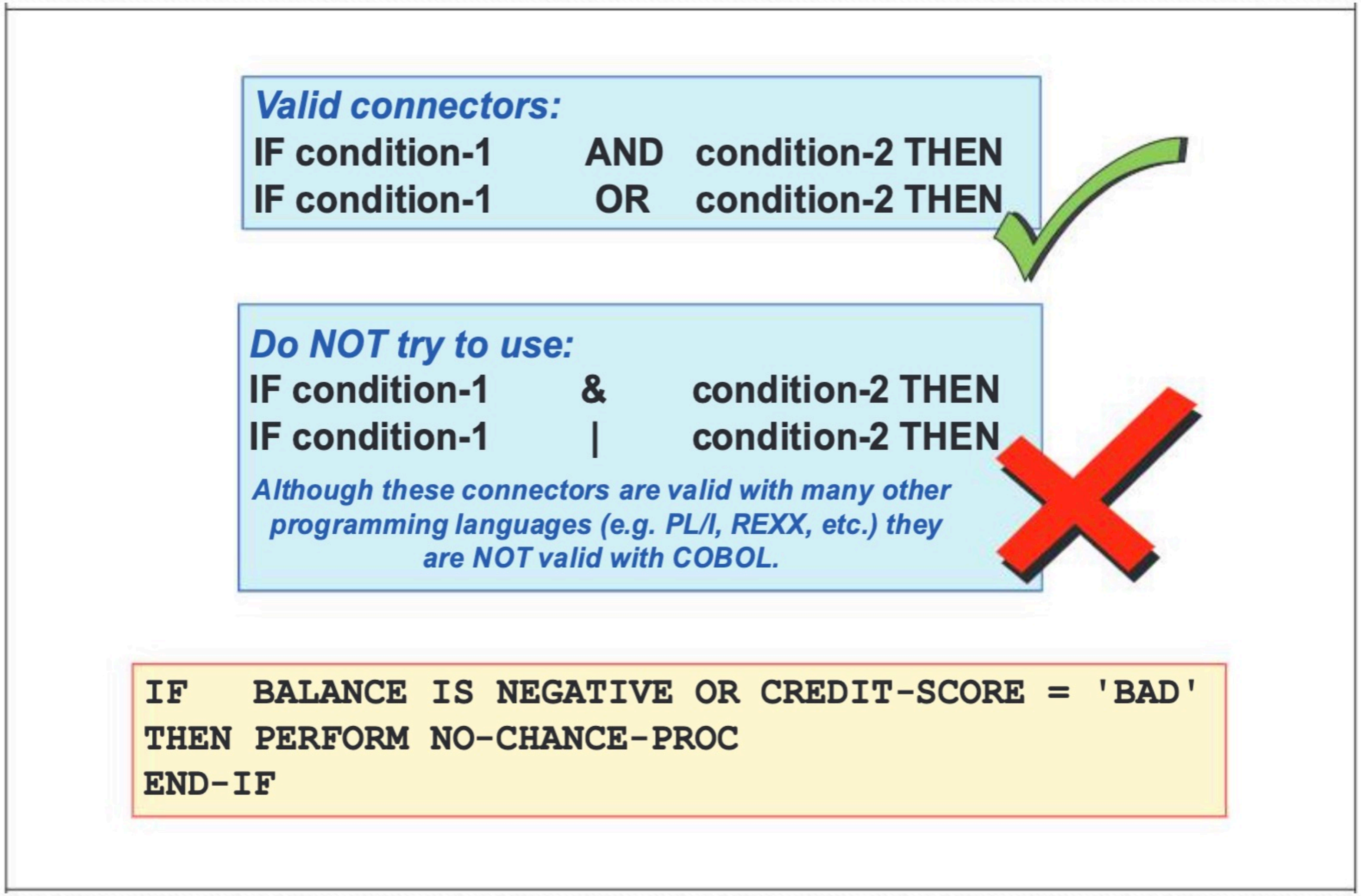
***The above may also be preceded with NOT***

```
IF   BALANCE - AMOUNT IS NEGATIVE
THEN PERFORM OVERDRAWN-PROC
END-IF
```

```
IF   BALANCE - AMOUNT  NEGATIVE
THEN PERFORM OVERDRAWN-PROC
END-IF
```

```
IF   BALANCE - AMOUNT IS NOT POSITIVE
THEN PERFORM OVERDRAWN-PROC
END-IF
```

# Multiple conditions



*Valid connectors:*

```
IF condition-1     AND   condition-2 THEN
IF condition-1      OR   condition-2 THEN
```

✓

*Do NOT try to use:*

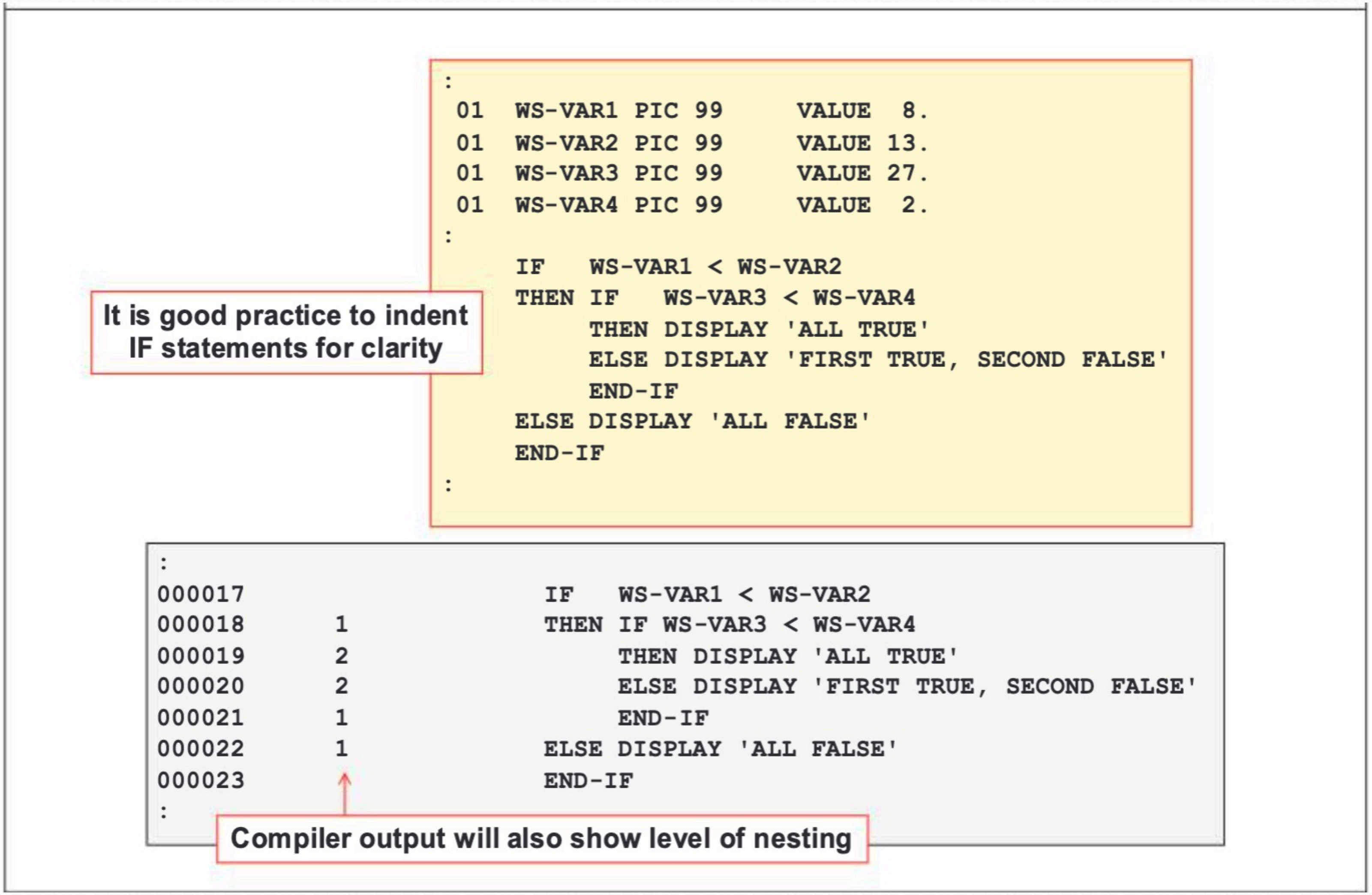
```
IF condition-1      &      condition-2 THEN
IF condition-1      |      condition-2 THEN
```

*Although these connectors are valid with many other programming languages (e.g. PL/I, REXX, etc.) they are NOT valid with COBOL.*

✗

```
IF   BALANCE IS NEGATIVE OR CREDIT-SCORE = 'BAD'
THEN PERFORM NO-CHANCE-PROC
END-IF
```

# Nested IF conditions



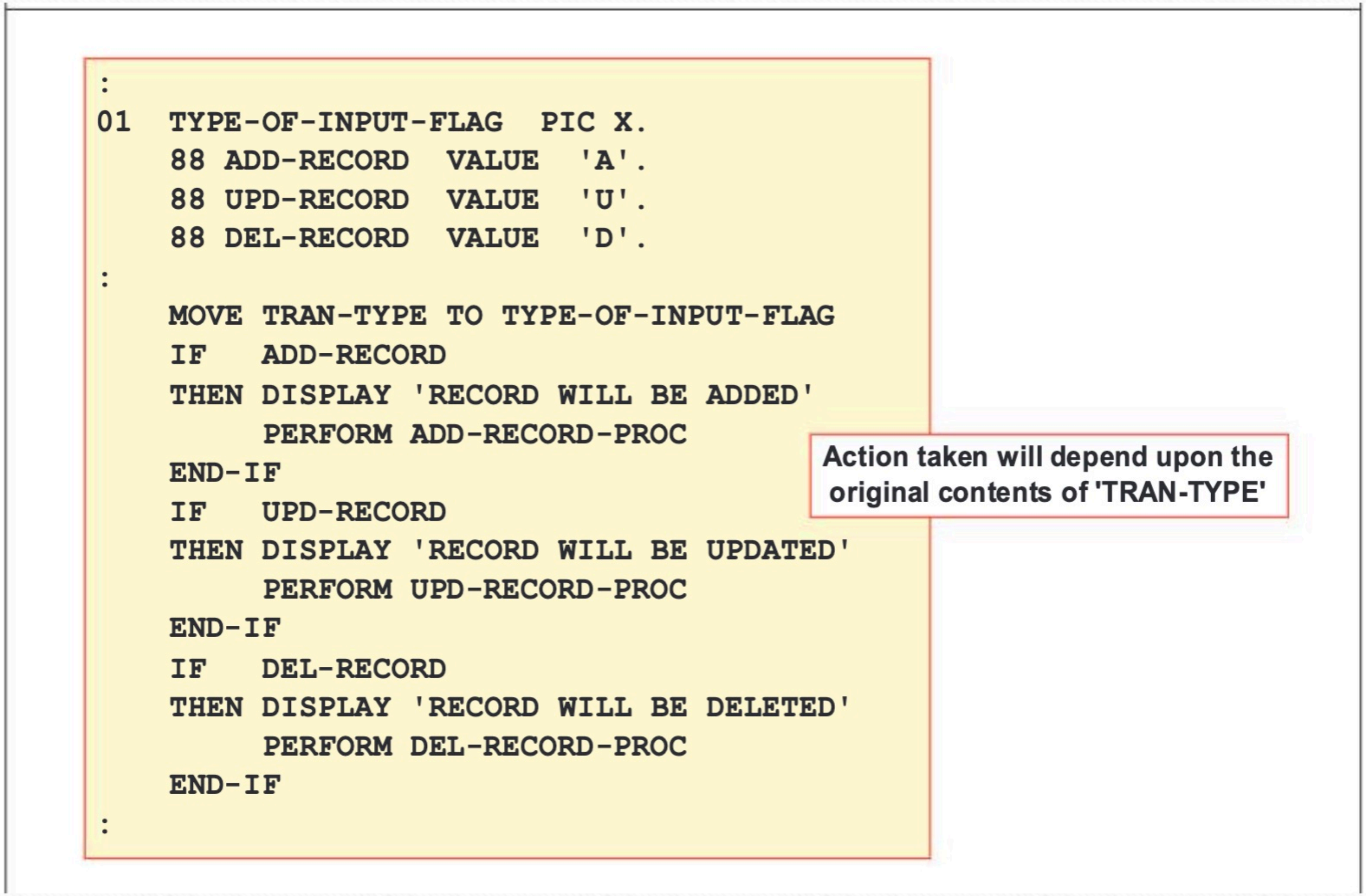
```
:
 01  WS-VAR1 PIC 99     VALUE  8.
 01  WS-VAR2 PIC 99     VALUE 13.
 01  WS-VAR3 PIC 99     VALUE 27.
 01  WS-VAR4 PIC 99     VALUE  2.
:
     IF   WS-VAR1 < WS-VAR2
     THEN IF   WS-VAR3 < WS-VAR4
          THEN DISPLAY 'ALL TRUE'
          ELSE DISPLAY 'FIRST TRUE, SECOND FALSE'
          END-IF
     ELSE DISPLAY 'ALL FALSE'
     END-IF
:
```

It is good practice to indent IF statements for clarity

```
:
000017                    IF   WS-VAR1 < WS-VAR2
000018      1             THEN IF WS-VAR3 < WS-VAR4
000019      2                  THEN DISPLAY 'ALL TRUE'
000020      2                  ELSE DISPLAY 'FIRST TRUE, SECOND FALSE'
000021      1                  END-IF
000022      1             ELSE DISPLAY 'ALL FALSE'
000023                    END-IF
:
```

Compiler output will also show level of nesting

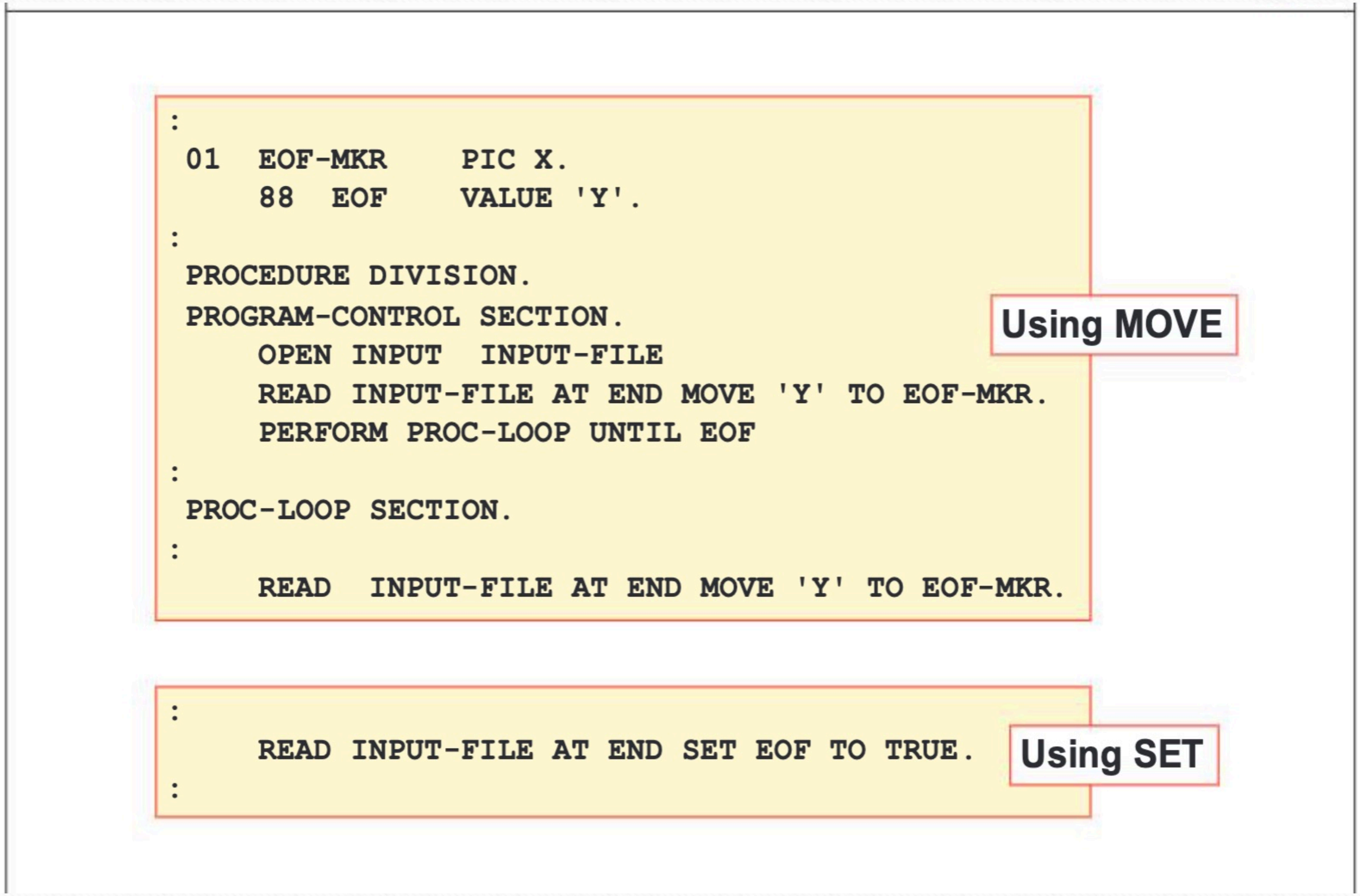
# 88 Level identifiers



```
:
01  TYPE-OF-INPUT-FLAG  PIC X.
    88 ADD-RECORD  VALUE  'A'.
    88 UPD-RECORD  VALUE  'U'.
    88 DEL-RECORD  VALUE  'D'.
:
    MOVE TRAN-TYPE TO TYPE-OF-INPUT-FLAG
    IF   ADD-RECORD
    THEN DISPLAY 'RECORD WILL BE ADDED'
         PERFORM ADD-RECORD-PROC
    END-IF
    IF   UPD-RECORD
    THEN DISPLAY 'RECORD WILL BE UPDATED'
         PERFORM UPD-RECORD-PROC
    END-IF
    IF   DEL-RECORD
    THEN DISPLAY 'RECORD WILL BE DELETED'
         PERFORM DEL-RECORD-PROC
    END-IF
:
```

Action taken will depend upon the original contents of 'TRAN-TYPE'

# Setting 88 levels



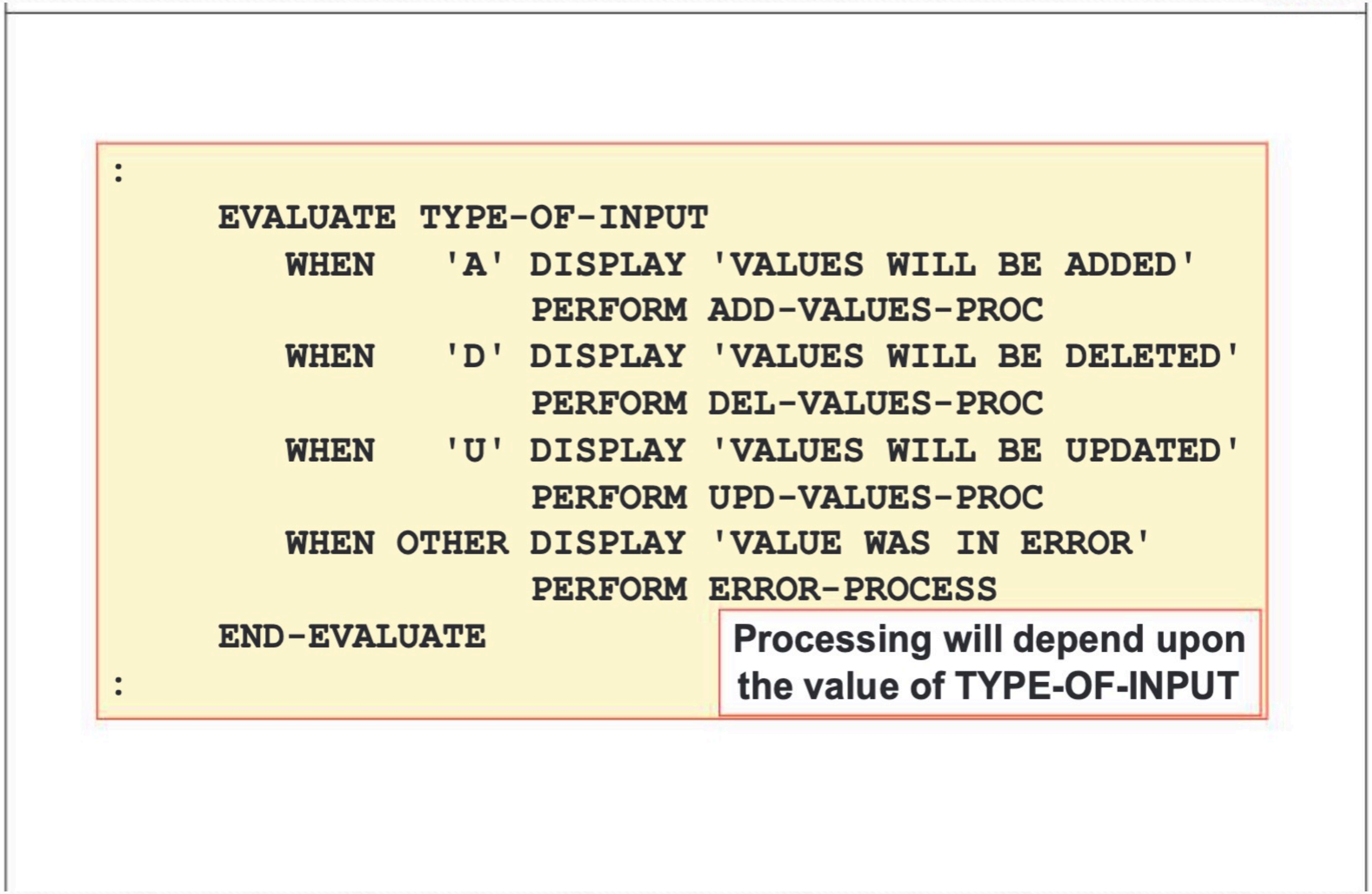
```
:
 01  EOF-MKR    PIC X.
     88  EOF    VALUE 'Y'.
:
 PROCEDURE DIVISION.
 PROGRAM-CONTROL SECTION.
     OPEN INPUT  INPUT-FILE
     READ INPUT-FILE AT END MOVE 'Y' TO EOF-MKR.
     PERFORM PROC-LOOP UNTIL EOF
:
 PROC-LOOP SECTION.
:
     READ  INPUT-FILE AT END MOVE 'Y' TO EOF-MKR.
```

Using MOVE

```
:
     READ INPUT-FILE AT END SET EOF TO TRUE.
:
```

Using SET

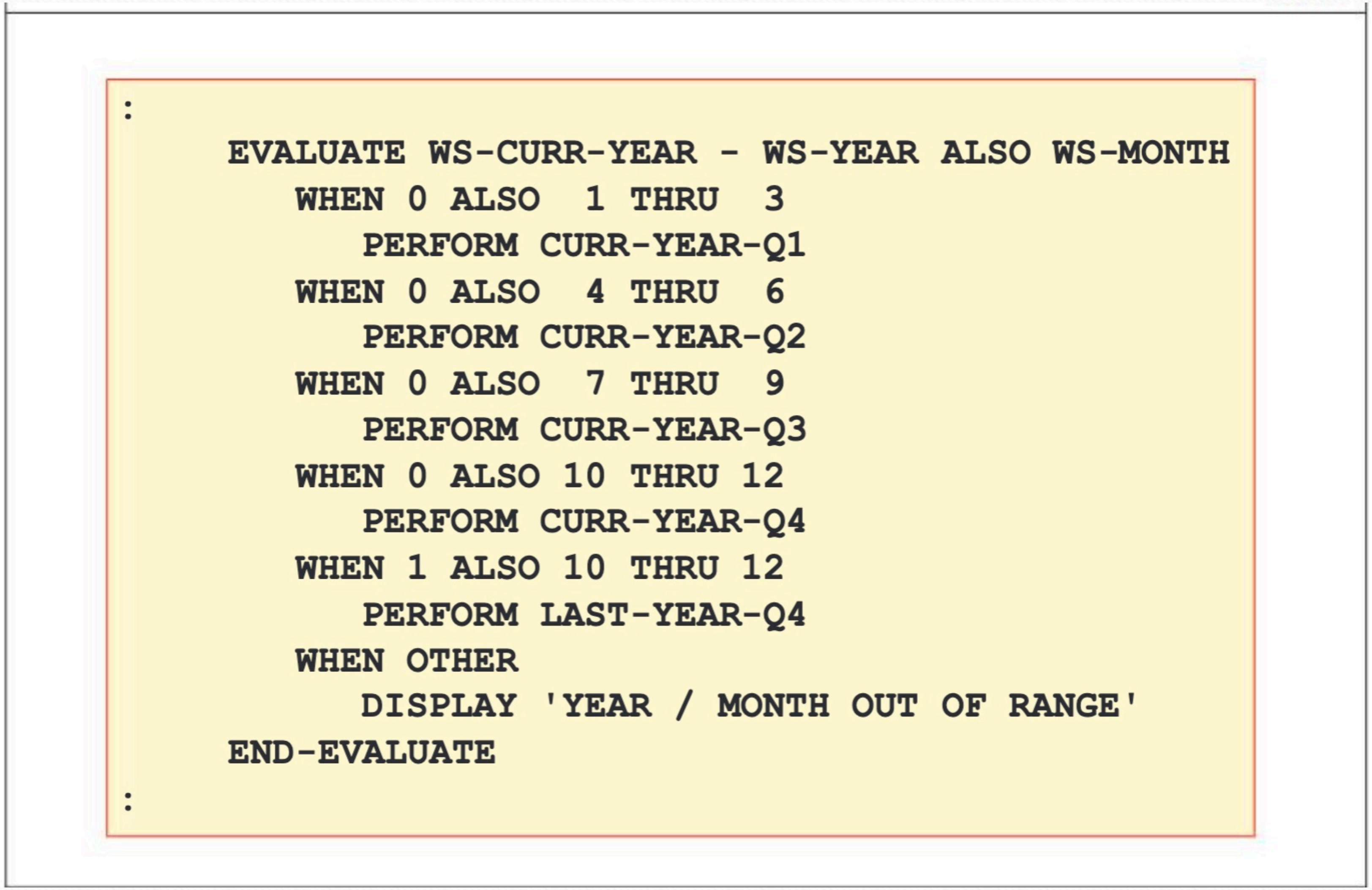
# Evaluate statement



```
:
    EVALUATE TYPE-OF-INPUT
       WHEN   'A' DISPLAY 'VALUES WILL BE ADDED'
                  PERFORM ADD-VALUES-PROC
       WHEN   'D' DISPLAY 'VALUES WILL BE DELETED'
                  PERFORM DEL-VALUES-PROC
       WHEN   'U' DISPLAY 'VALUES WILL BE UPDATED'
                  PERFORM UPD-VALUES-PROC
       WHEN OTHER DISPLAY 'VALUE WAS IN ERROR'
                  PERFORM ERROR-PROCESS
    END-EVALUATE
:
```

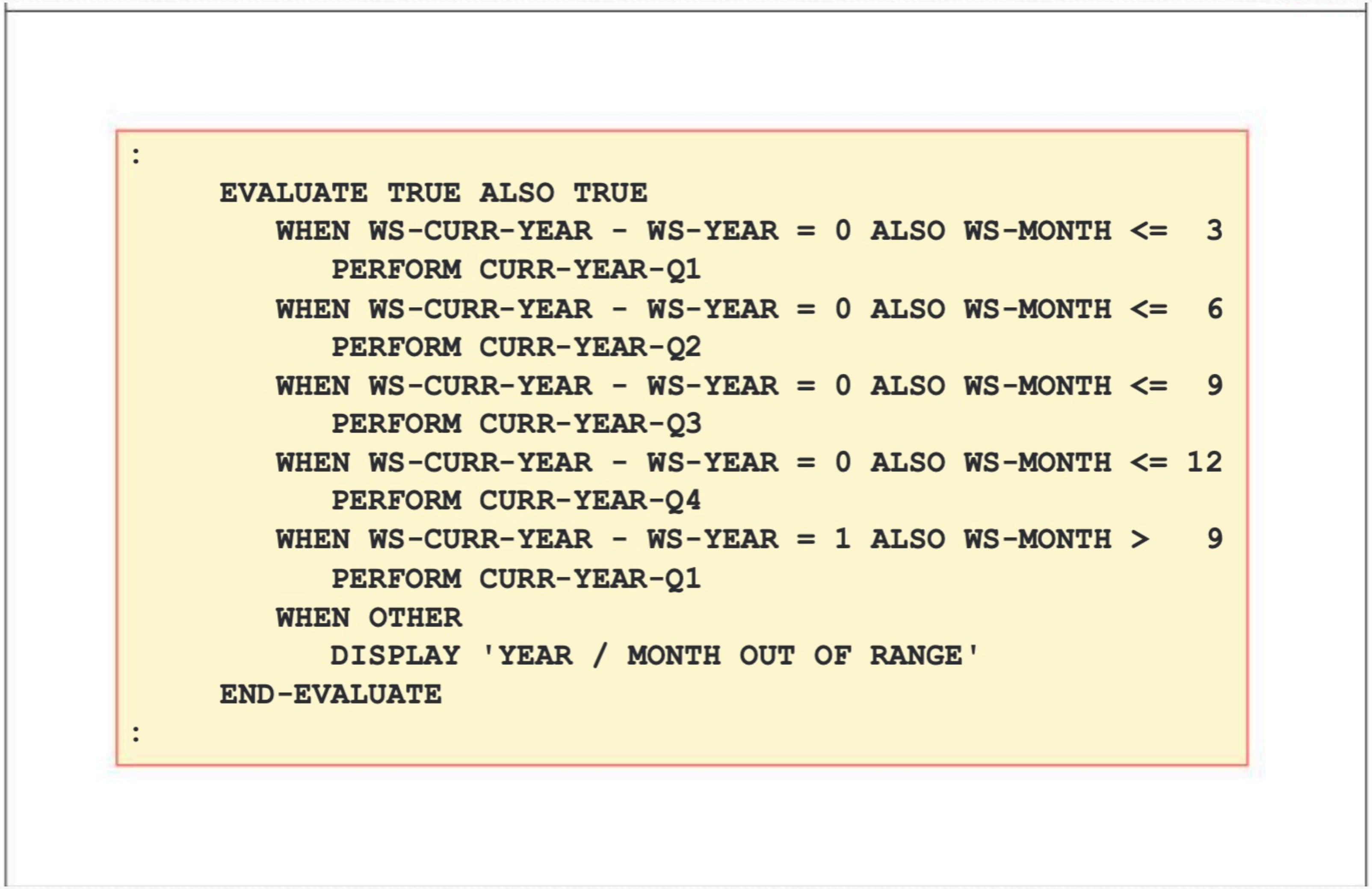
Processing will depend upon the value of TYPE-OF-INPUT

# Evaluate multiple values



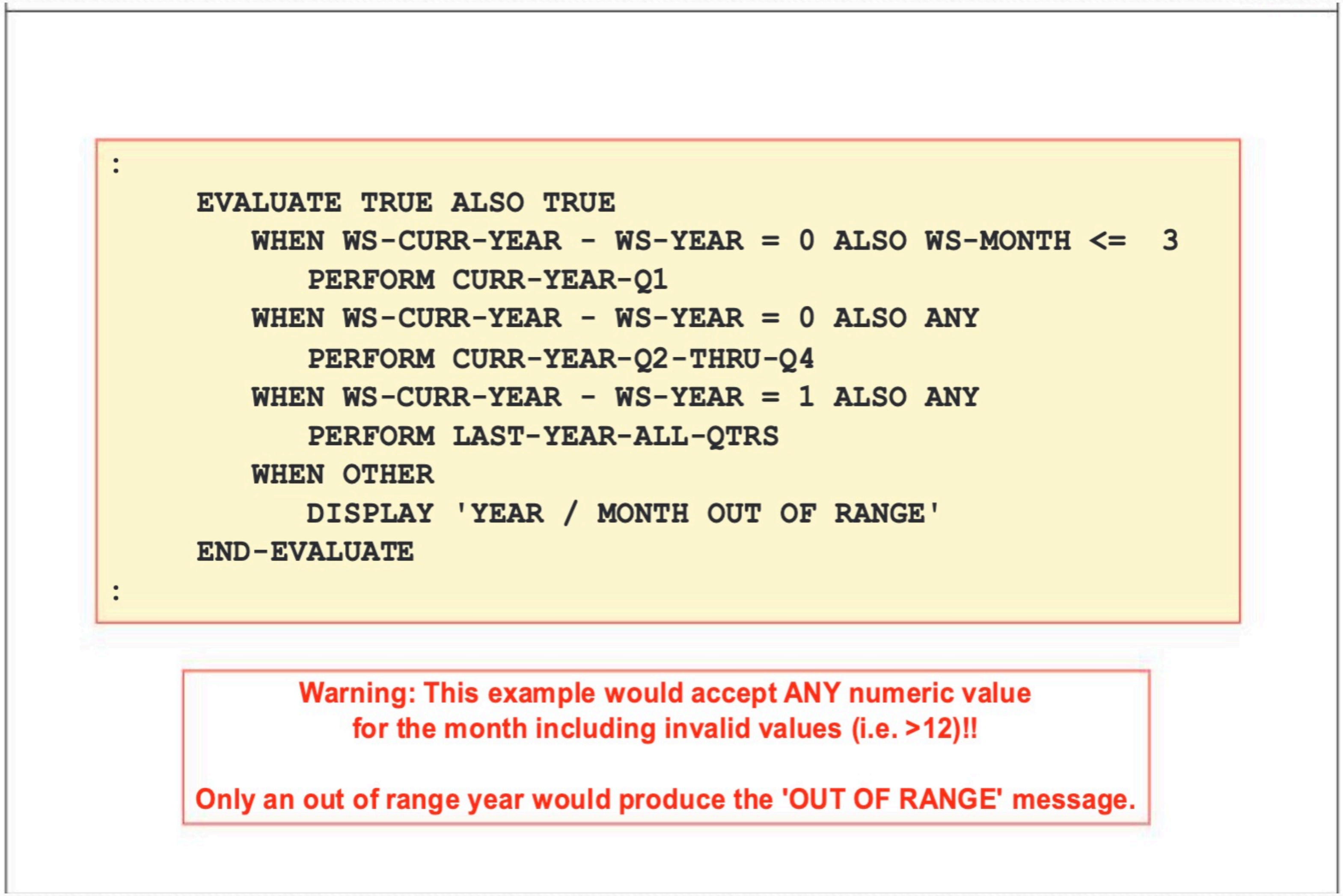
```
:
    EVALUATE WS-CURR-YEAR - WS-YEAR ALSO WS-MONTH
       WHEN 0 ALSO  1 THRU  3
          PERFORM CURR-YEAR-Q1
       WHEN 0 ALSO  4 THRU  6
          PERFORM CURR-YEAR-Q2
       WHEN 0 ALSO  7 THRU  9
          PERFORM CURR-YEAR-Q3
       WHEN 0 ALSO 10 THRU 12
          PERFORM CURR-YEAR-Q4
       WHEN 1 ALSO 10 THRU 12
          PERFORM LAST-YEAR-Q4
       WHEN OTHER
          DISPLAY 'YEAR / MONTH OUT OF RANGE'
    END-EVALUATE
:
```

# Evaluate using TRUE / FALSE



```
:
     EVALUATE TRUE ALSO TRUE
        WHEN WS-CURR-YEAR - WS-YEAR = 0 ALSO WS-MONTH <=  3
           PERFORM CURR-YEAR-Q1
        WHEN WS-CURR-YEAR - WS-YEAR = 0 ALSO WS-MONTH <=  6
           PERFORM CURR-YEAR-Q2
        WHEN WS-CURR-YEAR - WS-YEAR = 0 ALSO WS-MONTH <=  9
           PERFORM CURR-YEAR-Q3
        WHEN WS-CURR-YEAR - WS-YEAR = 0 ALSO WS-MONTH <= 12
           PERFORM CURR-YEAR-Q4
        WHEN WS-CURR-YEAR - WS-YEAR = 1 ALSO WS-MONTH >   9
           PERFORM CURR-YEAR-Q1
        WHEN OTHER
           DISPLAY 'YEAR / MONTH OUT OF RANGE'
     END-EVALUATE
:
```

# Evaluate ANY

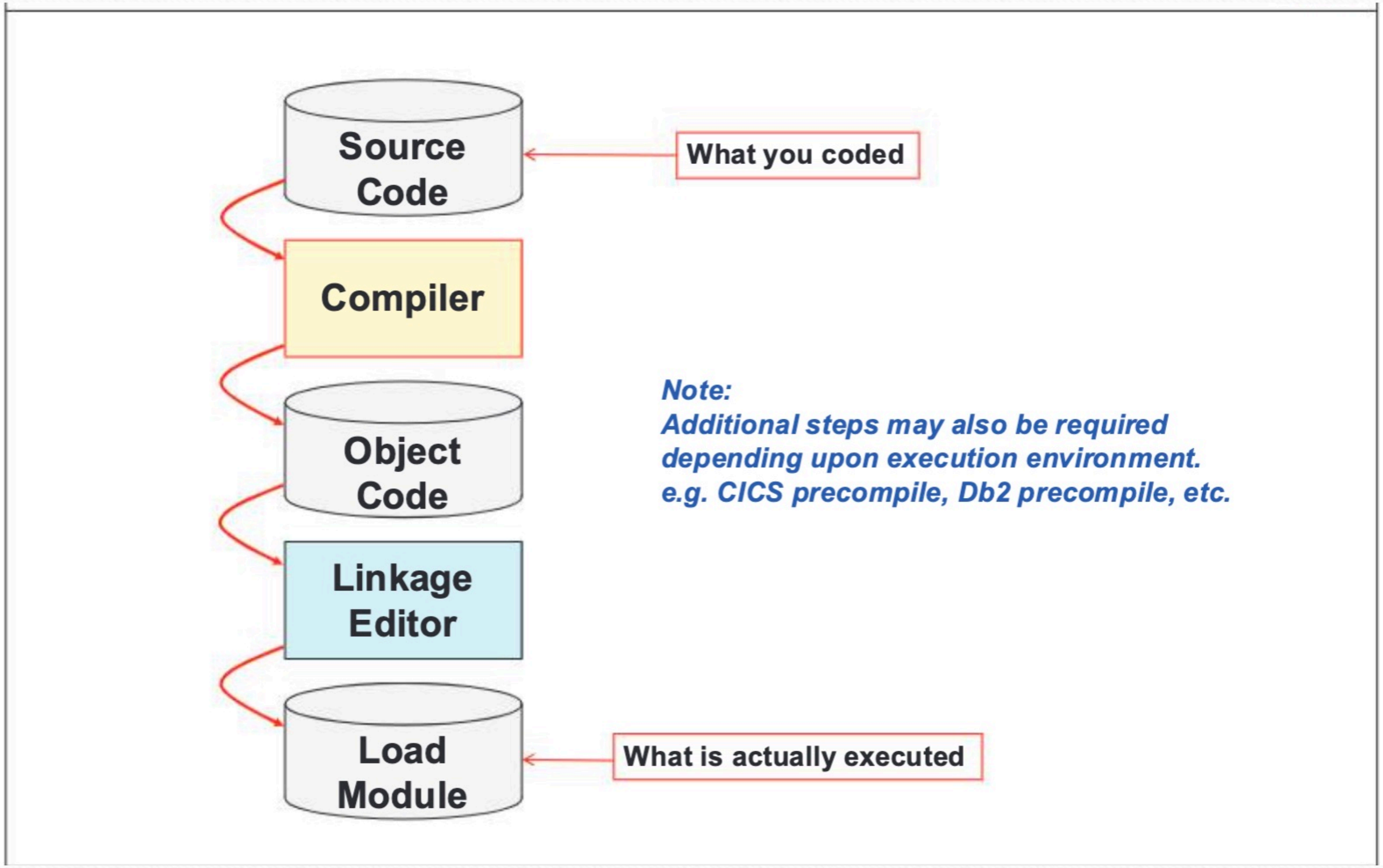


```
:
    EVALUATE TRUE ALSO TRUE
       WHEN WS-CURR-YEAR - WS-YEAR = 0 ALSO WS-MONTH <=  3
          PERFORM CURR-YEAR-Q1
       WHEN WS-CURR-YEAR - WS-YEAR = 0 ALSO ANY
          PERFORM CURR-YEAR-Q2-THRU-Q4
       WHEN WS-CURR-YEAR - WS-YEAR = 1 ALSO ANY
          PERFORM LAST-YEAR-ALL-QTRS
       WHEN OTHER
          DISPLAY 'YEAR / MONTH OUT OF RANGE'
    END-EVALUATE
:
```

Warning: This example would accept ANY numeric value for the month including invalid values (i.e. >12)!!

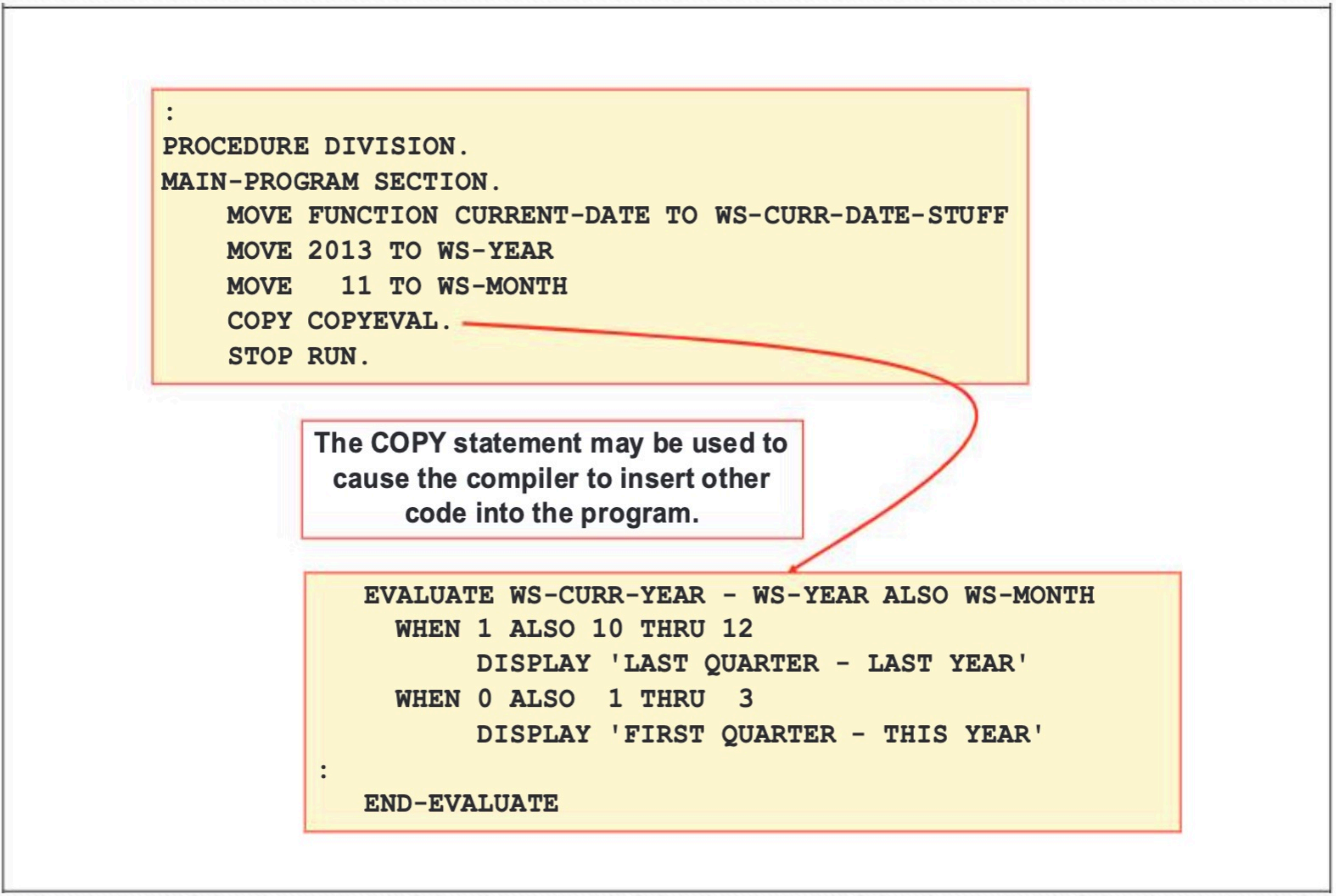
Only an out of range year would produce the 'OUT OF RANGE' message.

# Program running preparations



*Note:*
*Additional steps may also be required depending upon execution environment.*
*e.g. CICS precompile, Db2 precompile, etc.*

# COPY – Compiler directive statements

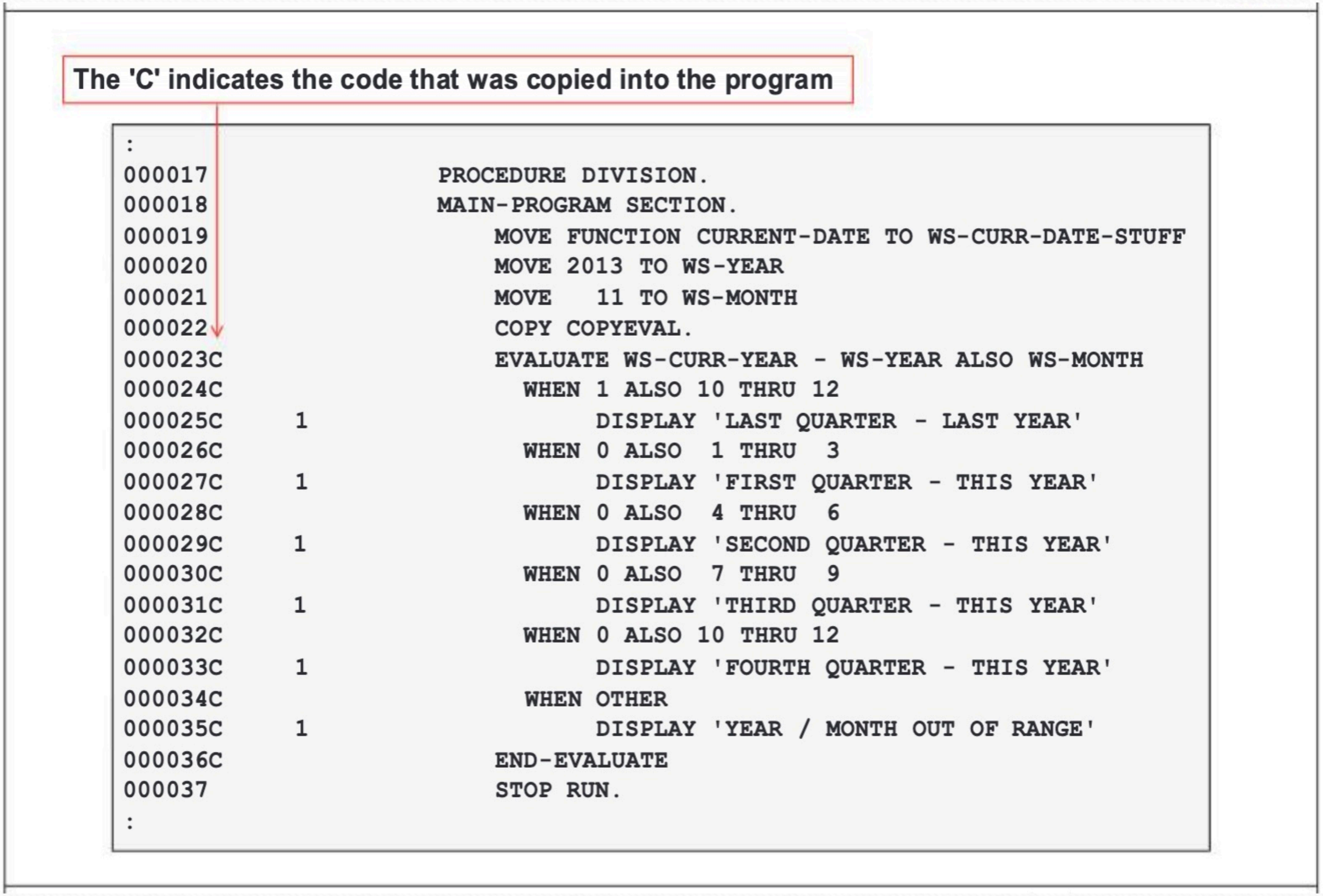


```
:
PROCEDURE DIVISION.
MAIN-PROGRAM SECTION.
    MOVE FUNCTION CURRENT-DATE TO WS-CURR-DATE-STUFF
    MOVE 2013 TO WS-YEAR
    MOVE   11 TO WS-MONTH
    COPY COPYEVAL.
    STOP RUN.
```

The COPY statement may be used to cause the compiler to insert other code into the program.

```
   EVALUATE WS-CURR-YEAR - WS-YEAR ALSO WS-MONTH
     WHEN 1 ALSO 10 THRU 12
          DISPLAY 'LAST QUARTER - LAST YEAR'
     WHEN 0 ALSO  1 THRU  3
          DISPLAY 'FIRST QUARTER - THIS YEAR'
:
   END-EVALUATE
```

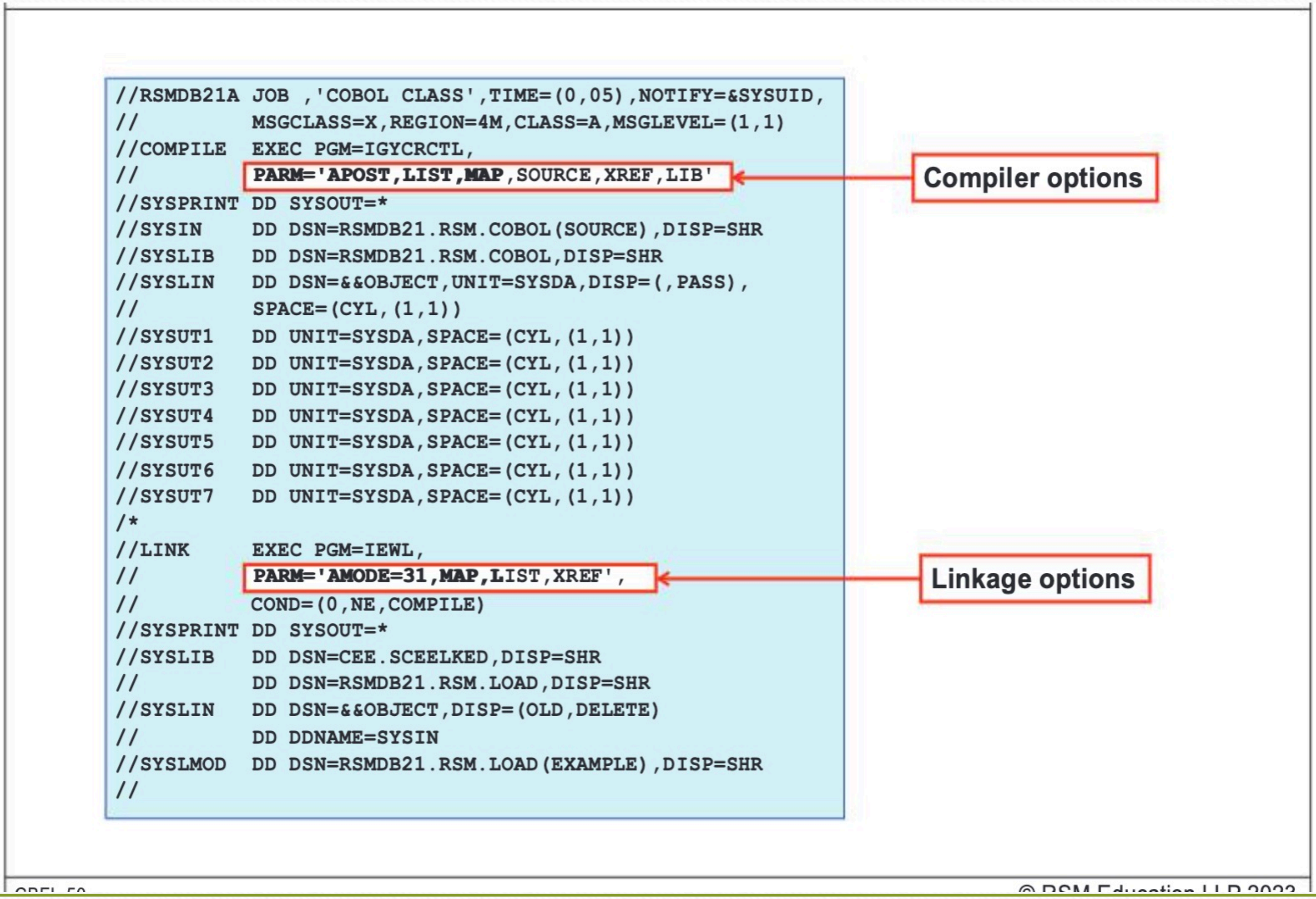
# The ‘c’ lines were copied in



The 'C' indicates the code that was copied into the program

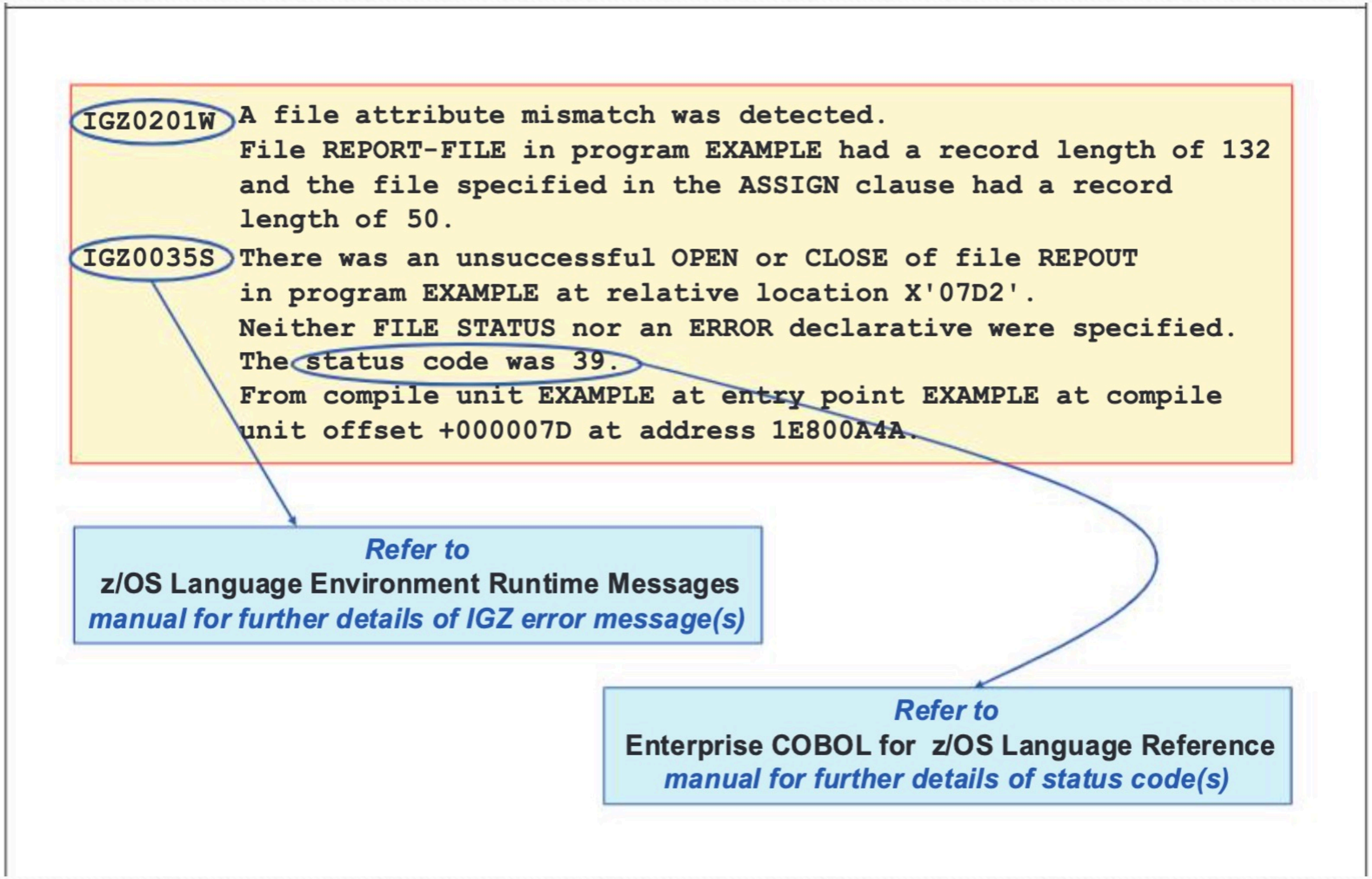
```
:
000017                PROCEDURE DIVISION.
000018                MAIN-PROGRAM SECTION.
000019                    MOVE FUNCTION CURRENT-DATE TO WS-CURR-DATE-STUFF
000020                    MOVE 2013 TO WS-YEAR
000021                    MOVE   11 TO WS-MONTH
000022                    COPY COPYEVAL.
000023C                   EVALUATE WS-CURR-YEAR - WS-YEAR ALSO WS-MONTH
000024C                     WHEN 1 ALSO 10 THRU 12
000025C     1                    DISPLAY 'LAST QUARTER - LAST YEAR'
000026C                     WHEN 0 ALSO  1 THRU  3
000027C     1                    DISPLAY 'FIRST QUARTER - THIS YEAR'
000028C                     WHEN 0 ALSO  4 THRU  6
000029C     1                    DISPLAY 'SECOND QUARTER - THIS YEAR'
000030C                     WHEN 0 ALSO  7 THRU  9
000031C     1                    DISPLAY 'THIRD QUARTER - THIS YEAR'
000032C                     WHEN 0 ALSO 10 THRU 12
000033C     1                    DISPLAY 'FOURTH QUARTER - THIS YEAR'
000034C                     WHEN OTHER
000035C     1                    DISPLAY 'YEAR / MONTH OUT OF RANGE'
000036C                   END-EVALUATE
000037                    STOP RUN.
:
```

# Compile option



```
//RSMDB21A JOB ,'COBOL CLASS',TIME=(0,05),NOTIFY=&SYSUID,
//          MSGCLASS=X,REGION=4M,CLASS=A,MSGLEVEL=(1,1)
//COMPILE  EXEC PGM=IGYCRCTL,
//          PARM='APOST,LIST,MAP,SOURCE,XREF,LIB'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DSN=RSMDB21.RSM.COBOL(SOURCE),DISP=SHR
//SYSLIB   DD DSN=RSMDB21.RSM.COBOL,DISP=SHR
//SYSLIN   DD DSN=&&OBJECT,UNIT=SYSDA,DISP=(,PASS),
//          SPACE=(CYL,(1,1))
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
/*
//LINK     EXEC PGM=IEWL,
//          PARM='AMODE=31,MAP,LIST,XREF',
//          COND=(0,NE,COMPILE)
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSN=CEE.SCEELKED,DISP=SHR
//          DD DSN=RSMDB21.RSM.LOAD,DISP=SHR
//SYSLIN   DD DSN=&&OBJECT,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
//SYSLMOD  DD DSN=RSMDB21.RSM.LOAD(EXAMPLE),DISP=SHR
//
```

# Error message examples

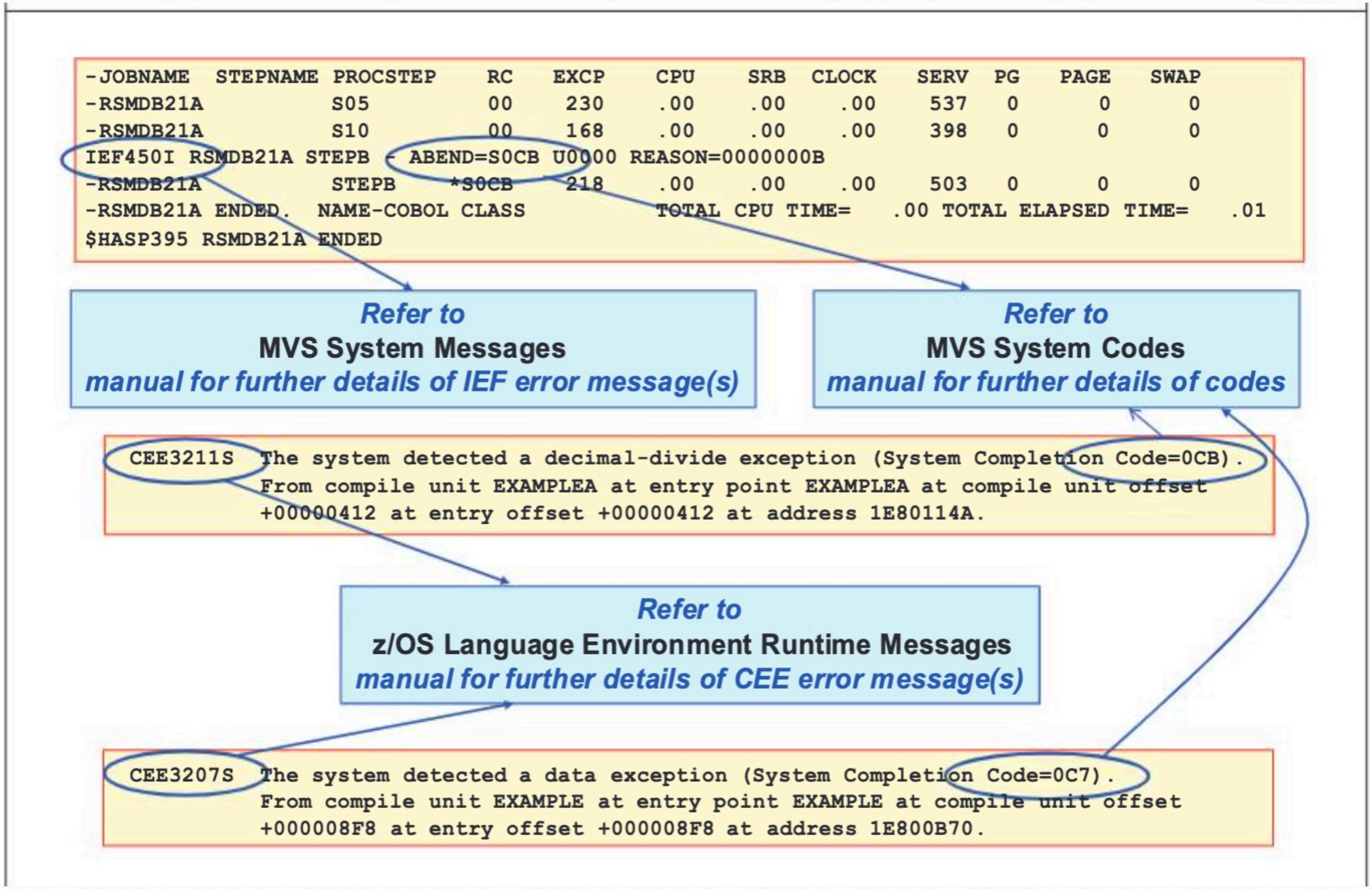


```
IGZ0201W A file attribute mismatch was detected.
         File REPORT-FILE in program EXAMPLE had a record length of 132
         and the file specified in the ASSIGN clause had a record
         length of 50.
IGZ0035S There was an unsuccessful OPEN or CLOSE of file REPOUT
         in program EXAMPLE at relative location X'07D2'.
         Neither FILE STATUS nor an ERROR declarative were specified.
         The status code was 39.
         From compile unit EXAMPLE at entry point EXAMPLE at compile
         unit offset +000007D at address 1E800A4A.
```

*Refer to*
**z/OS Language Environment Runtime Messages**
*manual for further details of IGZ error message(s)*

*Refer to*
**Enterprise COBOL for z/OS Language Reference**
*manual for further details of status code(s)*

# ABEND examples



```
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK    SERV  PG    PAGE    SWAP
-RSMDB21A          S05         00    230    .00    .00    .00     537   0       0       0
-RSMDB21A          S10         00    168    .00    .00    .00     398   0       0       0
IEF450I RSMDB21A STEPB - ABEND=S0CB U0000 REASON=0000000B
-RSMDB21A          STEPB     *S0CB    218    .00    .00    .00     503   0       0       0
-RSMDB21A ENDED.  NAME-COBOL CLASS          TOTAL CPU TIME=    .00 TOTAL ELAPSED TIME=    .01
$HASP395 RSMDB21A ENDED
```

*Refer to*
**MVS System Messages**
*manual for further details of IEF error message(s)*

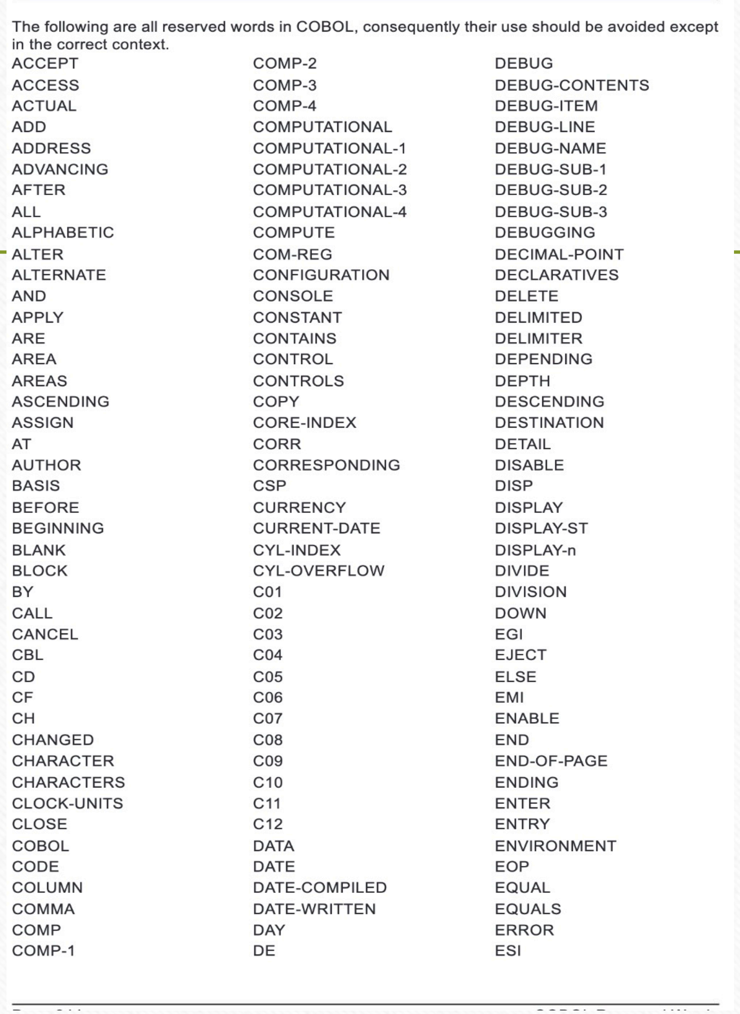
*Refer to*
**MVS System Codes**
*manual for further details of codes*

```
CEE3211S The system detected a decimal-divide exception (System Completion Code=0CB).
         From compile unit EXAMPLEA at entry point EXAMPLEA at compile unit offset
         +00000412 at entry offset +00000412 at address 1E80114A.
```

*Refer to*
**z/OS Language Environment Runtime Messages**
*manual for further details of CEE error message(s)*

```
CEE3207S The system detected a data exception (System Completion Code=0C7).
         From compile unit EXAMPLE at entry point EXAMPLE at compile unit offset
         +000008F8 at entry offset +000008F8 at address 1E800B70.
```

# COBOL reserved words (key words)



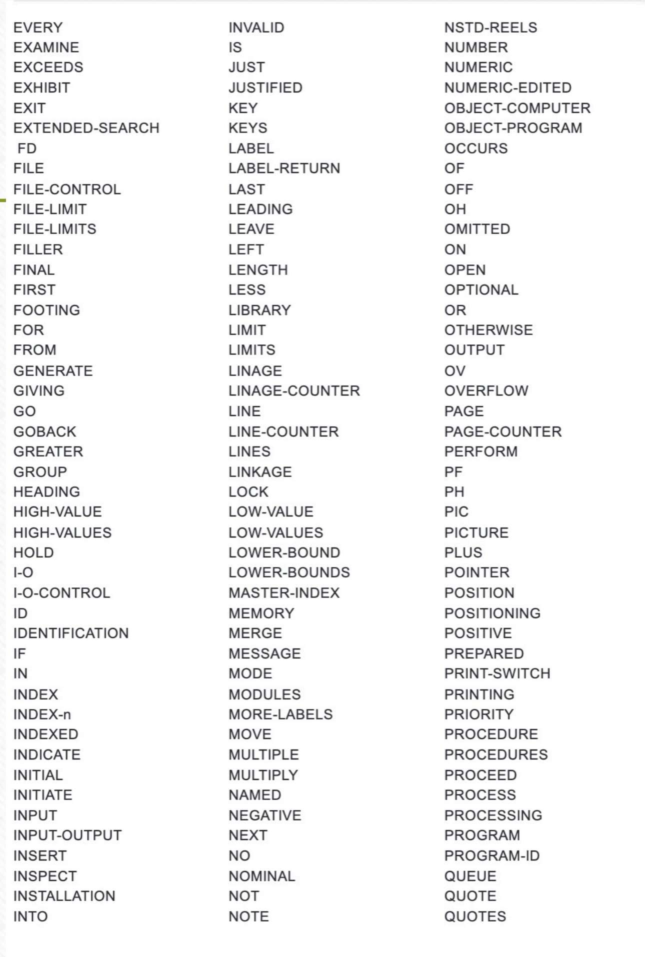
The following are all reserved words in COBOL, consequently their use should be avoided except in the correct context.

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BASIS
BEFORE
BEGINNING
BLANK
BLOCK
BY
CALL
CANCEL
CBL
CD
CF
CH
CHANGED
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMP
COMP-1

COMP-2
COMP-3
COMP-4
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
COM-REG
CONFIGURATION
CONSOLE
CONSTANT
CONTAINS
CONTROL
CONTROLS
COPY
CORE-INDEX
CORR
CORRESPONDING
CSP
CURRENCY
CURRENT-DATE
CYL-INDEX
CYL-OVERFLOW
C01
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE

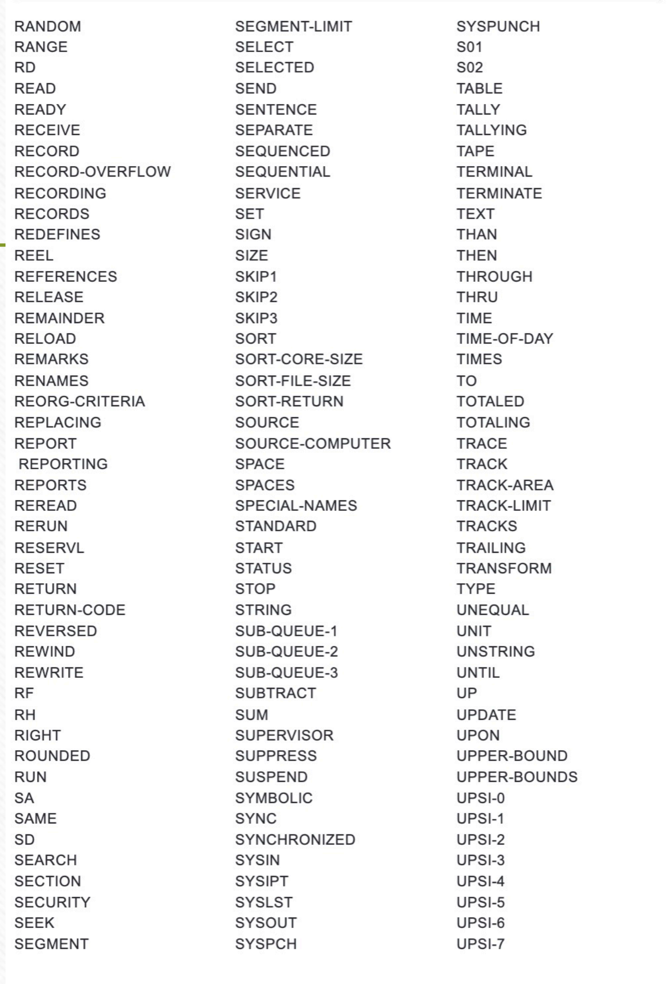
DEBUG
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DEPTH
DESCENDING
DESTINATION
DETAIL
DISABLE
DISP
DISPLAY
DISPLAY-ST
DISPLAY-n
DIVIDE
DIVISION
DOWN
EGI
EJECT
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENDING
ENTER
ENTRY
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
ESI

# Keywords continue



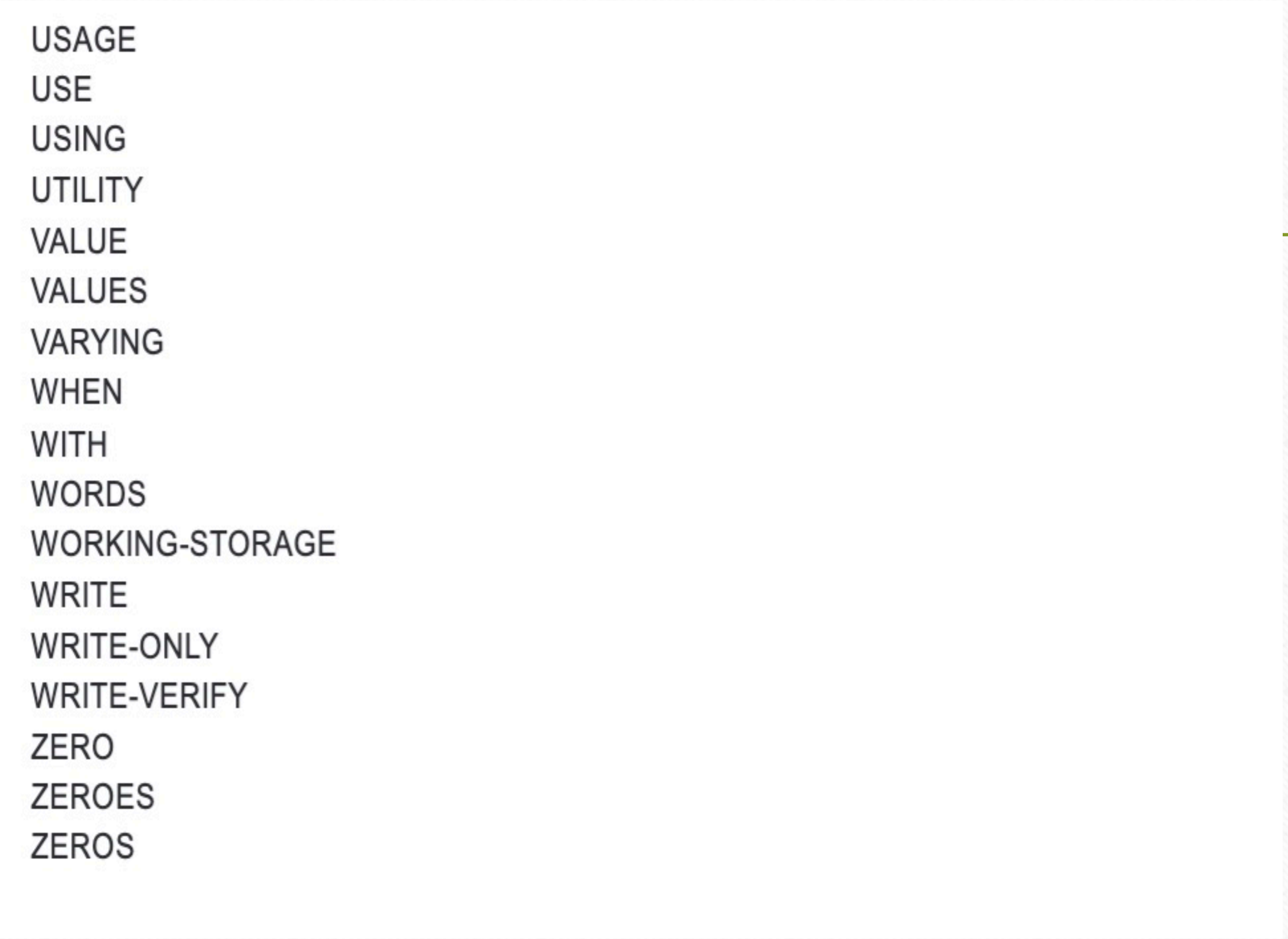
EVERY
EXAMINE
EXCEEDS
EXHIBIT
EXIT
EXTENDED-SEARCH
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GO
GOBACK
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
HOLD
I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INDEX
INDEX-n
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSERT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
KEYS
LABEL
LABEL-RETURN
LAST
LEADING
LEAVE
LEFT
LENGTH
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS
MASTER-INDEX
MEMORY
MERGE
MESSAGE
MODE
MODULES
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY
NAMED
NEGATIVE
NEXT
NO
NOMINAL
NOT
NOTE
NSTD-REELS
NUMBER
NUMERIC
NUMERIC-EDITED
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
OV
OVERFLOW
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIONING
POSITIVE
PREPARED
PRINT-SWITCH
PRINTING
PRIORITY
PROCEDURE
PROCEDURES
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID
QUEUE
QUOTE
QUOTES

# Keywords continue



RANDOM
RANGE
RD
READ
READY
RECEIVE
RECORD
RECORD-OVERFLOW
RECORDING
RECORDS
REDEFINES
REEL
REFERENCES
RELEASE
REMAINDER
RELOAD
REMARKS
RENAMES
REORG-CRITERIA
REPLACING
REPORT
REPORTING
REPORTS
REREAD
RERUN
RESERVL
RESET
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN
SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SELECTED
SEND
SENTENCE
SEPARATE
SEQUENCED
SEQUENTIAL
SERVICE
SET
SIGN
SIZE
SKIP1
SKIP2
SKIP3
SORT
SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SUSPEND
SYMBOLIC
SYNC
SYNCHRONIZED
SYSIN
SYSIPT
SYSLST
SYSOUT
SYSPCH
SYSPUNCH
S01
S02
TABLE
TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIME-OF-DAY
TIMES
TO
TOTALED
TOTALING
TRACE
TRACK
TRACK-AREA
TRACK-LIMIT
TRACKS
TRAILING
TRANSFORM
TYPE
UNEQUAL
UNIT
UNSTRING
UNTIL
UP
UPDATE
UPON
UPPER-BOUND
UPPER-BOUNDS
UPSI-0
UPSI-1
UPSI-2
UPSI-3
UPSI-4
UPSI-5
UPSI-6
UPSI-7

# Keywords continue



USAGE
USE
USING
UTILITY
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
WRITE-ONLY
WRITE-VERIFY
ZERO
ZEROES
ZEROS