CIB 27

ISO/IEC JTC1/SC22/WG4 N 061

Record of Response

Issue 2

ISO/IEC JTC1/SC22/WG4 Defect Reports 056 through 061

Programming Languages -- COBOL

ISO 1989:1985 (ANSI X3.23-1985)
ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989)

May 1994

Introduction

This document contains responses to defects reported concerning ISO 1989:1985 (Reference 1) and ISO 1989:1985/Amd.1:1992 (Reference 2). The responses are interpretations of Programming Language COBOL prepared by Accredited Standards Committee X3J4 and approved by SC22/WG4. These interpretations, reflecting the technical opinion of the experts maintaining Programming Language COBOL, are informative and have no normative effect on Standard COBOL.

Related normative changes, if any, either have been included in ISO/IEC 1989/DAM2, Programming languages - COBOL Amendment 2: Correction and clarification amendment for COBOL, or are planned for a future revision of the standard.

The interpretations in this Record of Response will be published nearly simultaneously by Accredited Standards Committee X3 in the U.S.A. in COBOL Information Bulletin 27 (CIB-27). To maintain correspondence between CIB-27 and this Record of Response, each interpretation carries dual identification in the style "DR-nnn / A-nnn" where

DR-nnn is the WG4 Defect Response identifier, which corresponds to the number of the related Defect Report, and

A-nnn is the X3 identifier.

Terminology and references

Second Standard COBOL ISO 1989:1978, Programming languages — COBOL (ANSI X3.23-1968, Programming Language — COBOL)

Third Standard COBOL ISO 1989:1985, Programming languages — COBOL (ANSI X3.23-1985, Programming Language — COBOL)

Amendment 1 ISO 1989:1985/Amd.1:1992, Programming languages — COBOL, Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language — Intrinsic Function Module for COBOL)

Amendment 2 ISO/IEC 1989/DAM2, Programming languages — COBOL Amendment 2: Correction and clarification amendment for COBOL (ANSI X3.23b-1993, Programming Language - Correction Amendment for COBOL).

Defect Report Summary

Defect Report and Response Number	X3 CIB-27 Answer Number	Subject
DR-056	A-310	Continuation of nonnumeric literals with embedded quotation marks
DR-057	A-313	Intrinsic function "errors" and transfer of control
DR-058	A-338	Zero length groups - class and class tests
DR-059	A-341	Types of problems with intrinsic function arguments
DR-060	A-345	Upper-/lower-case equivalence of D in indicator area
DR-061	A-359	Reference to linage-counter in a program without files

Interpretation DR-056 / A-310

SUBJECT: Continuation of Nonnumeric Literals with Embedded Quotation Marks

REFERENCES: ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL)

1. Page IV-10, Section 4.2.2.2.1.2 Syntax Rule (2)
2. Page IV-42, Section 7.2.2, Continuation of Lines
3. Page IV-9, Section 4.2.2.2.1, Nonnumeric Literals
4. Page IV-5, Section 4.2.1, Separators, Rule (5)

QUESTION:

Reference 1 defines how to embed a quotation mark within a nonnumeric literal by using two contiguous quotation mark characters. Reference 2 explains how to continue a nonnumeric literal. However, there still seems to be some confusion concerning some continued nonnumeric literals with embedded quotation marks.

NOTE: In all the following examples, please assume that the level number is in Area A, that the continuation hyphen is in the indicator area, that the continuation literal begins in Area B, and that the last visible character of the first line is the last character position before the Margin R.

Question 1: Is each of the following data-items defined in a standard conforming way so as to have an initial value of

"AB

If not, are they standard conforming, but with a different value (and if so, what value) or are they not standard conforming (and if so why)?

```
 01  FIELD-1-A PIC X(03)        VALUE          "
-    """AB".
 01  FIELD-1-B PIC X(03)        VALUE         ""
-    ""AB".
 01  FIELD-1-C PIC X(03)        VALUE        """
-    "AB".
 01  FIELD-1-D PIC X(03)        VALUE       """A
-    "B".
 01  FIELD-1-E PIC X(03)        VALUE      """AB
-    "".
 01  FIELD-1-F PIC X(03)        VALUE         ""
-                                             "
-    ""AB".
```

Question 2: Is each of the following data-items defined in a standard conforming way so as to have an initial value of

A"B

If not, are they standard conforming, but with a different value (and if so, what value) or are they not standard conforming (and if so why)?

```
 01  FIELD-2-A PIC X(03)       VALUE          "
-    "A""B".
 01  FIELD-2-B PIC X(03)       VALUE          "A
-    """B".
 01  FIELD-2-C PIC X(03)       VALUE          "A"
-    ""B".
 01  FIELD-2-D PIC X(03)       VALUE          "A""
-    "B".
 01  FIELD-2-E PIC X(03)       VALUE          "A""B
-    "".
 01  FIELD-2-F PIC X(03)       VALUE          "A"
-                                               "
-    ""B".
```

Question 3: Is each of the following data-items defined in a standard conforming way so as to have an initial value of

AB"

If not, are they standard conforming, but with a different value (and if so, what value) or are they not standard conforming (and if so why)?

```
 01  FIELD-3-A PIC X(03)       VALUE          "
-    "AB""".
 01  FIELD-3-B PIC X(03)       VALUE          "A
-    "B""".
 01  FIELD-3-C PIC X(03)       VALUE          "AB
-    """".
 01  FIELD-3-D PIC X(03)       VALUE          "AB"
-    """.
 01  FIELD-3-E PIC X(03)       VALUE          "AB""
-    "".
 01  FIELD-3-F PIC X(03)       VALUE          "AB"
-                                               "
-    """.
```

Question 4: Is each of the following data-items defined in a standard conforming way so as to have an initial value of

' "

If not, are they standard conforming, but with a different value (and if so, what value) or are they not standard conforming (and if so why)?

```
 01  FIELD-4-A PIC X(01)       VALUE          "
-    """".
 01  FIELD-4-B PIC X(01)       VALUE          ""
-    """.
 01  FIELD-4-C PIC X(01)       VALUE          """
-    "".
 01  FIELD-4-D PIC X(01)       VALUE          "
-                                               "
-    """".
 01  FIELD-4-E PIC X(01)       VALUE          ""
-                                               "
-    """.
 01  FIELD-4-F PIC X(01)       VALUE          """
-                                               "
-    "".
```

RESPONSE

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) is that the examples in the questions are standard conforming and have the initial values suggested in the question. This interpretation is based on the following rules:

- the rules for nonnumeric literals (Reference 3) require "separator quotation marks" as delimiters, and
- the rules for separator quotation marks (Reference 4) identify what the successor and predecessor characters must be, and
- the rules for Continuation of Lines (Reference 2) establish the successor/predecessor characters when a literal is being broken across a line, and
- the rule for contiguous quotation marks in literals (Reference 1) addresses quotation marks not accounted for by the above rules.

CHANGES NEEDED IN STANDARD:

None

Interpretation DR-057 / A-313

SUBJECT: Intrinsic Function "errors" and Transfer of Control

REFERENCES:

1. ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL):
   a. Page A-7, 8. INTRINSIC FUNCTION FACILITY
   b. Page A-18, Change to VI-70, Paragraph 6.4.7
   c. Page A-27, Section 1.2.2, Value Returned by a Function
   d. Page A-28, Section 2.2, Arguments, paragraph 3
   e. Page A-29, Section 2.3, Types of Functions
2. ISO 1989:1985/Amd.1:1992, Programming Languages – COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language – Intrinsic Function Module for COBOL):
   a. Page VI-30, GR4b, Content of alphanumeric item
   b. Page IV-25 to IV-26, Explicit and Implicit Transfers of Control
   c. Page VI-70, Section 6.4.7, Incompatible Data
   d. Page VI-29, GR2b, Content of alphabetic item
   e. Page III-1, Section 2, Definitions, Alphabetic Character
   f. Page I-9, Section 1.5, Definition of a Conforming Source Program, last paragraph
   g. Page I-7, Section 1.5.2.1, first sentence

QUESTION:

There are several places within the Intrinsic Function Amendment that clearly state that the returned value is "undefined" when the arguments do not meet the constraints of the function (cf. References 1a, 1c, and 1d). However, each of these references seems worded so that a conforming implementation *must* return some value and that the "transfer of control" is *not* undefined.

The following questions arise:

1. Is it true that a conforming implementation must return some value if the arguments passed to the function do not meet the constraints for that function?
2. Is it true that the "transfer of control" which must follow a reference to a function with invalidly specified arguments is as defined in standard "transfer of control", i.e., in reference 2b?
3. As an alphanumeric item can contain any value in the computer's character set (reference 2a), is there any possible value that can be returned for an "alphanumeric function" which would not be valid for this temporary data item?

4. For Integer and Numeric functions, is it valid for an implementation to simply define that the returned value is *not* a valid numeric (or integer) value or must a conforming implementation define what "undefined" (?) value is returned? Assuming that they need not define what value is returned, does reference 2c (as modified by reference 1b) mean that for these types of functions, transfer of control is not defined. Similarly, if the implementation does define values to be returned that are valid numeric (or integer) values, do the standard rules for transfer of control then apply?
5. Would it be a valid extension for a conforming implementation to provide an alternate "path" for transfer of control when invalid arguments are passed to an alphanumeric function? to numeric and/or integer functions? For example, could a conforming implementation pass control to a "new type" of declarative, when invalid arguments were passed to an alphanumeric function (as an extension of course)?
6. Is conformance to "data type" for arguments of a function considered an "argument constraint" or reference to an item with "incompatible data"? For example, in the LOWER-CASE function, it is clear that passing a 0 length group item would violate a "specified argument constraint"; however, it is unclear whether the rules for "incompatible data" or the rules for "argument constraint" would apply if an alphabetic item with digits in it were used.

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

Question 1:

No, it is not true that a conforming implementation must return a value if the arguments passed to the function do not meet the constraints for that function.

Question 2:

No, transfer of control following a reference to a function with invalidly specified arguments is undefined (Reference 2f).

Question 3:

Since an alphanumeric value can contain any character in the computer's character set (Reference 2a), any possible value returned by an alphanumeric function would be a valid value for the temporary data item.

Question 4:

These questions cannot be answered. The standard does not define what is or is not a valid implementation for circumstances that are undefined in the standard.

Question 5:

The standard does not define valid and invalid extensions, as long as they do "not require the inclusion of substitute or additional language elements in the source program in order to accomplish a function identical to that of a Standard COBOL language element" (Reference 2g). Therefore, a conforming implementation could have an extension to provide an alternate "path" for transfer of control when invalid arguments are passed to a function, regardless of the type of the function. A conforming implementation could pass control to a "new type" of declarative, as an extension.

Question 6:

If there are no specified content constraints, then data not in accordance with its defined class is "incompatible data". For example, the LOWER-CASE function specifies a rule that argument-1 must be class alphabetic or alphanumeric. An argument of class numeric violates this argument constraint; an argument of class alphabetic with numeric content is considered to be incompatible data.

Interpretation DR-058 / A-338

SUBJECT: Zero Length Groups - Class and Class Tests

REFERENCES: ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL)

1. Page IV-15, paragraph 4.3.3, Concept of Classes of Data
2. Page VI-26, paragraph 5.8.3, SR(5), The OCCURS Clause
3. Pages VI-56 to VI-57, paragraph 6.3.1.2, Class Condition

Record of Response Issue 1, Programming Languages– COBOL, ISO/IEC JTC1/SC22/WG4 N054 (X3 CIB-26)

4. Interpretation DR-011 / A-90, Zero-Length Groups

ISO/IEC 1989/DAM2, Programming languages - COBOL Amendment 2: Correction and clarification amendment for COBOL (ANSI X3.23b-1993, Programming Language - Correction Amendment for COBOL)

5. Correction to page VI-57 of ISO 1989:1985 (ANSI X3.23-1985)

QUESTION:

1. The definition of classes and categories on page IV-15 says that the class of all group items is "treated at object time as alphanumeric".

   a. Is it true that the class of a zero-length group item is also "treated at object time as alphanumeric"?

   b. If the class of a zero-length group item is not treated at object time as alphanumeric, what is it treated as and why?

   c. What is the difference between a group item being "class alphanumeric" as opposed to being "treated at object time as alphanumeric"?

2. The definition of classes and categories states that the "alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric (without editing)." and the table shows this to be true of group items as well as elementary items.

   a. What is the category of a zero-length group item (or what is it treated as at object time if that is different)?

   b. If different zero-length groups may be of different categories, how does one determine to which category a specific zero-length group belongs?

c. If all zero-length groups must fail both a NUMERIC and an ALPHABETIC class test, does this mean that all zero-length group items must be of one of the categories, NUMERIC EDITED, ALPHANUMERIC EDITED, or ALPHANUMERIC (by process of elimination)?

3. Given the rules for class tests, isn't it true that for any non-zero-length item (group or elementary) which will pass a class test, that any subitem (subset of the item) will pass the same class test? For example, if the following code displays ALPHABETIC for the first display, it will display true for the remainder while if it displays NUMERIC as its first display, it will still display true for the remainder of the displays?

```
01  Group-Item.
    05  ODO-Obj              Pic 9(02)   Value 99.
    05  Tabl.
        10  Tabl-Elem   Occurs 1 to 99 times
                        Depending on ODO-Obj
                        Indexed by Ind
                            Pic X(01).

...
If Tabl Alphabetic
    Display "Alphabetic"
    Perform
        With Test After
        Varying ODO-OBJ
            From 99 by -1
            Until ODO-OBJ = 1
                If Tabl Alphabetic
                    Display "True"
                Else
                    Display "False"
                End-IF
    End-Perform
Else If Tabl Numeric
    Display "Numeric"
    Perform
        With Test After
        Varying ODO-OBJ
            From 99 by -1
            Until ODO-OBJ = 1
                If Tabl Numeric
                    Display "True"
                Else
                    Display "False"
                End-IF
    End-Perform
End-If
End-If
```

4. Given the rules for class tests, isn't it true that for any variable length group item that can contain a zero-length group item that will pass a class test, that any subitem (subset of the item) will pass the same class test except for the zero-length group item which will fail? For example, if the following code displays ALPHABETIC for the first display, it will display true for the remainder except for the last display which will display FALSE while if it displays NUMERIC as its first display, it will still display true for the remainder of the displays except for the last which will display FALSE?

```
01  Group-Item.
    05  ODO-Obj              Pic 9(02)   Value 99.
    05  Tabl.
        10  Tabl-Elem   Occurs 0 to 99 times
                        Depending on ODO-Obj
                        Indexed by Ind
                            Pic X(01).
...                     .
If Tabl Alphabetic
    Display "Alphabetic"
    Perform
        With Test After
        Varying ODO-OBJ
            From 99 by -1
            Until ODO-OBJ = 0
                If Tabl Alphabetic
                    Display "True"
                Else
                    Display "False"
                End-IF
    End-Perform
Else If Tabl Numeric
    Display "Numeric"
    Perform
        With Test After
        Varying ODO-OBJ
            From 99 by -1
            Until ODO-OBJ = 0
                If Tabl Numeric
                    Display "True"
                Else
                    Display "False"
                End-IF
    End-Perform
End-If
End-If
```

5. If the answer to the preceding question is "yes", is it true that contrary to standard rules of logic, within COBOL the "null set" is not considered a subset of all sets - at least as far as class tests go?

Response:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) is the following:

1. a. Yes.

   c. There is no difference between a group item being "class alphanumeric" as opposed to being "treated at object time as alphanumeric."

2. Group items belong to no category, therefore zero-length group items belong to no category.

3. Yes, a class test in COBOL is a positive test, that is "if it consists entirely of the characters" (Reference 3).

4. Yes.

5. A zero-length group item is not considered a null set in COBOL, therefore we cannot answer your question as asked.

Changes Needed in Standard:

The table on Page IV-15 is incorrect. Changes will be addressed in a future standard.

Interpretation DR-059 / A-341

SUBJECT: Types of Problems with Intrinsic Function Arguments

REFERENCES:

1. ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL):
   a. Page I-9, 1.6, Definition of a Conforming Source Program
   b. Page VI-70, 6.4.7, Incompatible Data
2. ISO 1989:1985/Amd.1:1992, Programming Languages – COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language – Intrinsic Function Module for COBOL):
   a. Page A-28, 2.2, Arguments
3. ISO/IEC JTC1/SC22/WG4 N053, Interpretation DR-059 / A-313, Intrinsic Function "errors" and Transfer of Control

QUESTION:

Introduction:

Interpretation DR-059 / A-313 (which had not been completed at the time of writing this question) partially addresses the classification of "errors" in arguments used in function-identifiers. However, it seems possible that it may not resolve some issues. Therefore, this interpretation request is aimed at clarifying how a standard conforming implementation needs to classify "problems" with intrinsic function arguments.

Question 1

Are the following statements all true:

1. If the specification of one or more arguments within a function-identifier causes the program to contain "nonconforming source code" a conforming implementation need not even compile the source. If it does compile the source then the run-time results of executing that program are not defined in amended third Standard COBOL.
2. If the specification of a function-identifier is fully conforming source code, however, at run-time one or more arguments specified within the function-identifier (at the time the statement referencing the function-identifier is executed) contains a value that is not compatible with the class specified for that argument, this constitutes "incompatible data" and an ANSI conforming implementation need not continue execution beyond that statement. If execution does continue beyond that statement, then the values contained in all identifiers within the program and all transfers of control after that statement are not defined in amended third Standard COBOL.

3. If the specification of a function-identifier is fully conforming source code and all arguments used in that function-identifier contain "compatible data" at run-time, however, the value of one or more of the arguments violates a documented "argument constraint" then the resultant identifiers and the transfer of control are whatever the processing of X3J4 document A-313 determines is meant by the phrase "the returned value for the function is undefined".

Question 2

Assuming that the three categories of "problems" listed above are correct and that a conforming implementation needs to "evaluate" situations (implicitly or explicitly) in the order listed, would you please explain which of the three categories listed above, each of the following portions of sample code are. (If any of them do not fall into one of these categories, please explain into what category it does fall or what amended third Standard COBOL requires a conforming implementation to do with it.)

> **Note:** Assume that these samples are included within an (otherwise) conforming source program and that no other source statements "impact" these extracts.

1.

```
Compute Num-Field = Function SQRT (-1)
```

2.

```
Move Function Lower-Case (+ 123.45) to AlphaNum
```

3.

```
Compute Num-Field = Function LOG10 (Zero)
```

4.

```
Compute Num-Field = Function NumVal (+ 123,456,789,012,345,678,9)
```

5.

```
Compute Num-Field = Function Ord ("AB")
```

6.

```
Compute Num-Field = Function Integer-of-Day (1991366)
```

7.

```
Compute Num-Field = Function Integer-of-Date (19911)
```

8.

```
05  Num-Field          Pic S9(09).
05  AlphaNum           Pic  X(10).
    ...

Move Function Max (Num-Field AlphaNum) to Other-Field
```

9.

```
05  AlphaNum           Pic  X(02).
    ...

Compute Num-Field = Function Ord (AlphaNum)
```

10.

```
05 Int-Field          Pic 9(07).
   ...

Compute Num-Field = Function Integer-of-Date (Int-Field)
```

11.

```
Compute Num-Field = Function SQRT (Function Random - 1)
```

12.

```
05 Int-Field          Pic 9(09).
   ...

Move Function Upper-Case (Int-Field) to Other-Field
```

13.

```
Move Function Current-Date (Any-Field) to Other-Field
```

14.

```
05 Num-Field          Pic S9(09).
   ...

Move Spaces to Num-Field (1:)
Compute Num-Field = Function Integer (Num-Field)
```

15.

```
05  Alpha-Field      Pic  A(1).
05  AlphaNum         Redefines Alpha-Field
                     Pic  X(1).
    ...

Move "9" to AlphaNum
Move Function Min (Alpha-Field) to Other-Field
```

16.

```
05  Num-Field        Pic 9(05).
05  Alpha-Field      Redefines Num-Field
                     Pic A(05).
    ...

If Function Min (Num-Field) = Function Max (Alpha-Field)
```

17.

```
05  Group-Item.
    10  ODO-Obj      Pic 9(01).
    10  Tabl.
        15  Tabl-Elem   Occurs 0 to 9 times
                        Depending on ODO-Obj
                        Indexed by Ind
                        Pic X(01).
    ...

Move Zero to ODO-Obj
Compute Num-Field = Function Sum (Tabl-Elem (all))
```

18.

```
05  Group-Item.
    10  ODO-Obj        Pic 9(01).
    10  Tabl.
        15  Tabl-Elem  Occurs 0 to 9 times
                       Depending on ODO-Obj
                       Indexed by Ind
                       Pic X(01).
    ...

Move Zero to ODO-Obj
Compute Num-Field = Function Ord-Min (Tabl-Elem (all))
```

19.

```
05  Num-Field          Pic S9(09).
05  Int-Field          Pic  9(09).
    ...

Move -1 to Num-Field
Compute Other-Field = Function Annuity (Int-Field Num-Field)
```

20.

```
05  Num-Field          Pic S9(09).
05  Int-Field          Pic  9(09).
    ...

Compute Num-Field = Function Annuity (Num-Field (- Int-Field))
```

21.

```
05  Int-Field          Pic  9(09).
    ...

Compute Num-Field = Function Mod (Int-Field)
```

22.

```
05  Group-Item.
    10  ODO-Obj        Pic 9(01).
    10  Tabl.
        15  Tabl-Elem  Occurs 0 to 9 times
                       Depending on ODO-Obj
                       Indexed by Ind
                       Pic 9(01).
    ...

Move 1 to ODO-Obj
          Tabl-Elem (1)
Compute Num-Field = Function Present-Value (Tabl-Elem (all))
```

23.

```
05  Group-Item.
    10  ODO-Obj        Pic 9(01).
    10  Tabl.
        15  Tabl-Elem  Occurs 0 to 9 times
                       Depending on ODO-Obj
                       Indexed by Ind
                       Pic 9(01).
    ...

Move 2 to ODO-Obj
          Tabl-Elem (1)
          Tabl-Elem (2)
Compute Num-Field = Function Present-Value (Tabl-Elem (all))
```

24.

```
05  Num-Field          Pic S9(09).
05  Int-Field          Pic  9(09).
    ...

Compute Other-Field = Function Rem (Num-Field (Int-Field - Int-Field))
```

25.

```
05  Group-Item.
    10  ODO-Obj        Pic 9(01).
    10  Tabl.
        15  Tabl-Elem  Occurs 0 to 9 times
                       Depending on ODO-Obj
                       Indexed by Ind
                       Pic X(01).
    ...

Move Zero to ODO-Obj
If Function Reverse (Tabl) Alphabetic
     Display "False"
```

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

Question 1: All 3 statements are true.

Question 2: The samples posed in Question 2 are classified below into the categories outlined in Question 1, that is:

1. Nonconforming source code (Reference 1a),
2. Incompatible data (Reference 1b),
3. Violation of a function argument constraint (References 2 and 3).

In this response, violation of a COBOL rule is classified as nonconforming source when that violation is syntactically distinguishable by examination of the single statement containing the function reference. Any implementation that can detect category 2 and category 3 above at compile time may treat the source code as non-conforming. Where an arithmetic expression appears as a function argument, it is

assumed that the expression is evaluated in an intuitive manner, although the standard makes no such requirement.

Text in brackets identifies the rule violated or otherwise explains the reason for the classification.

Classification of Code Samples

1. Nonconforming source code
   [Reference 2, 2.40.3, SQRT, Argument Rule 2, the value of argument-1 must be zero or positive.]

```
Compute Num-Field = Function SQRT (-1)
```

2. Nonconforming source code
   [Reference 2, 2.22.3, LOWER-CASE, Argument Rule 1, argument-1 must be class alphabetic or alphanumeric ...]

```
Move Function Lower-Case ( + 123.45) to AlphaNum
```

3. Nonconforming source code
[Reference 2, 2.21.2, LOG10, Argument Rule 2, the value of argument-1 must be greater than zero.]

```
Compute Num-Field = Function LOG10 (Zero)
```

4. Nonconforming source code; either:
[Reference 2, 2.29.3, NUMVAL, Argument Rule 1, Argument-1 must be a nonnumeric literal or alphanumeric data item ...]

or
[Reference 2, 2.29.3, NUMVAL, Argument Rule 1, Argument-1 must be ... whose content has one of the following two formats ...]
or
[Reference 2, 2.29.3, NUMVAL, Argument Rule 2, The total number of digits in argument-1 must not exceed 18.]

The order of determination is undefined.

```
Compute Num-Field = Function NumVal (+123,456,789,012,345,678,9)
```

5. Nonconforming source code
[Reference 2, 2.31.3, ORD, Argument Rule 1, argument-1 must be one character in length ...]

```
Compute Num-Field = Function Ord ("AB")
```

6. Nonconforming source code
[Reference 2, 2.17.3, INTEGER-OF-DAY, Argument Rule 2b, DDD ... must be a positive integer less than 367 provided that it is valid for the year specified.]

```
Compute Num-Field = Function Integer-of-Day (1991366)
```

7. Nonconforming source code
[Reference 2, 2.16.3, INTEGER-OF-DATE, Argument Rule 1, argument-1 must be an integer of the form YYYYMMDD ...]

```
Compute Num-Field = Function Integer-of-Date (19911)
```

8. Nonconforming source code
[Reference 2, 2.23.2, MAX, Argument Rule 1, if more than one argument is specified, all arguments must be of the same class.]

```
05 Num-Field        Pic S9(09).
05 AlphaNum         Pic X(10).
   ...

Move Function Max (Num-Field AlphaNum) to Other-Field
```

9. Nonconforming source code
[Reference 2, 2.31.3, ORD, Argument Rule 1, argument-1 must be one character in length ...]

```
05 AlphaNum         Pic X(02).
   ...

Compute Num-Field = Function Ord (AlphaNum)
```

10. Nonconforming source code
[Reference 2, 2.16.3, INTEGER-OF-DATE, Argument Rule 1, Argument-1 must be an integer of the form YYYYMMDD ...]

```
05 Int-Field        Pic 9(07).
   ...

Compute Num-Field = Function Integer-of-Date (Int-Field)
```

11. Violation of argument constraint
[Reference 2, 2.40.3, SQRT, Argument Rule 2, the value of argument-1 must be zero or positive.]

```
Compute Num-Field = Function SQRT (Function Random - 1)
```

12. Nonconforming source code
[Reference 2, 2.44.3, UPPER-CASE, Argument Rule 1, argument-1 must be class alphabetic or alphanumeric ...]

```
05 Int-Field          Pic 9(09).
   ...

Move Function Upper-Case (Int-Field) to Other-Field
```

13. Nonconforming source code
[Reference 2, 2.11.2, CURRENT-DATE, general format. CURRENT-DATE has no arguments.]

```
Move Function Current-Date (Any-Field) to Other-Field
```

14. Incompatible data
[Reference 1b.]

```
05 Num-Field          Pic S9(09).
   ...

Move Spaces to Num-Field (1:)
Compute Num-Field = Function Integer (Num-Field)
```

15. Incompatible data
[Reference 1b.]

```
05  Alpha-Field      Pic  A(1).
05  AlphaNum         Redefines Alpha-Field
                     Pic  X(1).
  ...
Move "9" to AlphaNum
Move Function Min (Alpha-Field) to Other-Field
```

16. Incompatible data
[Because of the manner in which REDEFINES is used in this example, one or the other of the function-identifiers will reference incompatible data.]

```
05  Num-Field        Pic 9(05).
05  Alpha-Field      Redefines Num-Field
                     Pic A(05).
  ...

If Function Min (Num-Field) = Function Max (Alpha-Field)
```

17. Nonconforming source code
[Reference 2, 2.42.3, SUM, Argument Rule 1, Argument-1 must be class numeric.]

```
05  Group-Item.
  10  ODO-Obj        Pic 9(01).
  10  Tabl.
    15  Tabl-Elem    Occurs 0 to 9 times
                     Depending on ODO-Obj
                     Indexed by Ind
                     Pic X(01).
  ...
Move Zero to ODO-Obj
Compute Num-Field = Function Sum (Tabl-Elem (all))
```

18. Violation of argument constraint
[Reference 2, 2.2, "The evaluation of an ALL subscript must result in at least one argument, otherwise the returned value is undefined."]

```
05 Group-Item.
   10 ODO-Obj        Pic 9(01).
   10 Tabl.
      15 Tabl-Elem   Occurs 0 to 9 times
                     Depending on ODO-Obj
                     Indexed by Ind
                     Pic X(01).
   ...

Move Zero to ODO-Obj
Compute Num-Field = Function Ord-Min (Tabl-Elem (all))
```

19. Violation of argument constraint
[Reference 2, 2.6.3, ANNUITY, Argument rule 3, Argument-2 must be a positive integer.]

```
05 Num-Field        Pic S9(09).
05 Int-Field        Pic  9(09).
   ...

Move -1 to Num-Field
Compute Other-Field = Function Annuity (Int-Field Num-Field)
```

20. Nonconforming source code
[Reference 2, 2.6.3, ANNUITY, Argument rule 3, Argument-2 must be a positive integer.]

```
05 Num-Field        Pic S9(09).
05 Int-Field        Pic  9(09).
   ...
Compute Num-Field = Function Annuity (Num-Field (- Int-Field))
```

21. Nonconforming source code
[Reference 2, 2.28.2, MOD, General Format. Two arguments are required.]

```
05 Int-Field          Pic 9(09).
  ...
Compute Num-Field = Function Mod (Int-Field)
```

22. Nonconforming source code
[Reference 2, 2.34.2, PRESENT-VALUE, General Format. Two arguments are required. Subscript ALL is allowed "when the definition of a function permits an argument to be repeated a variable number of times". The PRESENT-VALUE format does not permit argument-1 to be repeated a variable number of times, and argument-2 is required.]

```
05 Group-Item.
   10 ODO-Obj        Pic 9(01).
   10 Tabl.
      15 Tabl-Elem   Occurs 0 to 9 times
                    Depending on ODO-Obj
                    Indexed by Ind
                    Pic 9(01).
  ...
Move 1 to ODO-Obj
          Tabl-Elem (1)
Compute Num-Field = Function Present-Value (Tabl-Elem (all))
```

23. Nonconforming source code
[Reference 2, 2.34.2, PRESENT-VALUE, General Format. Two arguments are required. Subscript ALL is allowed "when the definition of a function permits an argument to be repeated a variable number of times". The PRESENT-VALUE format does not permit argument-1 to be repeated a variable number of times, and argument-2 is required.]

```
05 Group-Item.
   10 ODO-Obj        Pic 9(01).
   10 Tabl.
      15 Tabl-Elem   Occurs 0 to 9 times
                    Depending on ODO-Obj
                    Indexed by Ind
                    Pic 9(01).
   ...

Move 2 to ODO-Obj
          Tabl-Elem (1)
          Tabl-Elem (2)
Compute Num-Field = Function Present-Value (Tabl-Elem (all))
```

24. Violation of Argument Constraint
[Reference 2, 2.37.3, REM, Argument Rule 2, The value of argument-2 must not be zero.]

```
05 Num-Field        Pic S9(09).
05 Int-Field        Pic  9(09).
   ...

Compute Other-Field = Function Rem (Num-Field (Int-Field - Int-Field))
```

25. Violation of argument constraint
[Reference 2, 2.28.3, REVERSE, Argument Rule 1, Argument-1 must be ... at least one character in length.]

```
05 Group-Item.
   10 ODO-Obj        Pic 9(01).
   10 Tabl.
      15 Tabl-Elem   Occurs 0 to 9 times
                     Depending on ODO-Obj
                     Indexed by Ind
                     Pic X(01).
   ...

Move Zero to ODO-Obj
If Function Reverse (Tabl) Alphabetic
      Display "False"
```

Interpretation DR-060 / A-345

SUBJECT: Upper-/Lower-Case Equivalence of D in Indicator Area

REFERENCES: ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL)

1. Page I-5, Required Modules
2. Page I-8, paragraph 1.5.2.5.2, Extension Language Elements
3. Page I-9, paragraph 1.6, Definition of a Conforming Source Program
4. Page I-43, Summary of Elements in Language Concepts
5. Page I-45, Summary of Elements in Environment Division
6. Page III-7, Definition of a Debugging Line
7. Page III-25, Definition of a Text Word
8. Page IV-4, paragraph 4.2, Language Structure
9. Page IV-5, paragraph 4.2.2, Character-Strings
10. Page IV-41, section 7, Reference Format
11. Page VI-10, paragraph 4.3, The Source-Computer Paragraph
12. Page VI-29, paragraph 5.9.3, Syntax Rule (3).
13. Page VI-141, paragraph 7.3, Debugging Lines
14. Page XII-4, paragraph 2.4, General Rule 7, the COPY Statement
15. Page XII-7, paragraph 3.4, General Rule 8, The REPLACE Statement

QUESTION:

1. Given the information in References (8.) and (9.), it would appear that the entire "text of a source program" is composed of separators, COBOL words, literals, PICTURE character-strings, and comment-entries. Into what classification do "elements" that are valid in the indicator area belong?

2. Is there either an option or a requirement that an implementation also treat the characters "D" and "d" as equivalent when they appear in the indicator area? If so why?

Response:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is that:

1. The standard does not specify the classification of the elements that are valid in the indicator area.

2. There is an unintentional omission in the standard. Third Standard COBOL requires that "D" and "d" be treated as equivalent in the indicator area.

Changes Needed in Standard:

Correct References 6, 13, 14 and 15 to allow "D" or "d" in the indicator area.

Interpretation DR-062 / A-359

SUBJECT: Reference to Linage-Counter in a Program without Files

REFERENCES: ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL)

1. Page IV-9, Special Registers
2. Page IV-20, Qualification, Rule 8, Linage-Counter Qualification
3. Page VII-5, Special Register LINAGE-COUNTER
4. Page VII-28, The LINAGE Clause, GR 9, The LINAGE-COUNTER
5. Page X-18, File Description Entry, GR 1, External and Linage-Counter

QUESTION SUMMARY:

Amended third Standard COBOL seems clear about how one can reference Linage-Counter special registers in programs with one or more files with a LINAGE clause. However, it is unclear whether the following statement on page IV-9 means that a single Linage-Counter special register is available to a program without any files with a LINAGE clause,

> "Unless specified otherwise in these specifications, one special register of each type is allocated for each program."

QUESTION:

1. Does amended third Standard COBOL require that a conforming COBOL implementation support references to (but not modifications of) a Linage-Counter special register in a program which has no files defined at all?

2. Does amended third Standard COBOL require that a conforming COBOL implementation support references to (but not modifications of) a Linage-Counter special register in a program which has files defined, but none with a LINAGE clause?

3. If an application has defined an External file with a Linage clause, does general rule (1) on page X-18 mean that any separately compiled program in the run-unit can reference the Linage-Counter special register or does it mean that only those separately compiled programs which actually have the external FD in them can reference the Linage-Counter special register? If

it means the former and there are two external files in the run-unit with the LINAGE clause, can conforming source code still use the file-name to qualify each Linage-Counter special register, even in those programs which do not contain the specific file's FD?

4. If the answers to the above questions mean that for a run-unit which has at least one External file with a LINAGE Clause, that every program (even those without any files defined) must be able to reference the Linage-Counter special register, but that for run-units without any files defined with the LINAGE clause, there is no requirement to be able to reference a Linage-Counter special register, then what is the run-time results of a reference to a Linage-Counter special register in a run-unit without any files or without any files defined with a LINAGE clause?

Response:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) is the following:

1. No.

2. No.

3. If an application has defined an External file with a LINAGE clause, Reference 5 means that only those separately compiled programs which have the external FD in them can reference the Linage-Counter special register.

4. Source code that has no file definitions and that references a Linage-Counter special register is non-conforming.

Changes needed in Standard:

Correction of the Standard is under investigation. A restriction should be added to Linage-Counter: each register needs to be checked to see if it needs rules to specify otherwise.