EBook Gratis

APRENDIZAJE cobol

Free unaffiliated eBook created from
Stack Overflow contributors.

#cobol

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cobol

It is an unofficial and free cobol ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cobol.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Cobol

Observaciones

COBOL es la usiness **CO** mmon **B O** riented programación **L** anguage.

A pesar de que se ha convertido en un nombre pronunciado, COBOL aún es tratado como acrónimo por el comité de estándares, y COBOL es la ortografía preferida por los organismos de estándares ISO e INCITS.

Especificacion estandar

La especificación actual es

ISO / IEC 1989: 2014 Tecnología de la información - Lenguajes de programación, sus entornos e interfaces de software del sistema - Lenguaje de programación COBOL

Ese documento se publicó en mayo de 2014 y se puede comprar en varias sucursales de organismos estándar, oficialmente alojados en

http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51416

Principal campo de uso

Orientado a los negocios. Eso generalmente significa procesamiento de transacciones. La banca, las agencias gubernamentales y la industria de seguros son áreas importantes de los despliegues de aplicaciones COBOL. Los sistemas mainframe de IBM generalmente tienen un compilador COBOL instalado. Hay más de 300 dialectos COBOL en existencia, con aproximadamente 10 versiones que toman la mayor parte de las implementaciones. La mayoría de estos compiladores son sistemas propietarios, pero también está disponible el software libre COBOL.

Categoría

COBOL es un lenguaje de programación procesal, imperativo, compilado. A partir de la especificación COBOL 2002, las características Orientadas a Objetos se agregaron al estándar.

Por intención de diseño, COBOL es un lenguaje de programación muy detallado. Aunque la forma algebraica está permitida:

```
COMPUTE I = R * B
```

la intención inicial era usar palabras completas para las descripciones computacionales y la manipulación de datos:

```
MULTIPLY INTEREST-RATE BY BALANCE GIVING CURRENT-INTEREST ROUNDED MODE IS NEAREST-EVEN
```

Esta decisión de diseño tiene tanto campeones como detractores. Algunos piensan que es demasiado detallado, mientras que otros argumentan que la sintaxis permite una mayor legibilidad en un entorno empresarial.

Matemáticas decimales

COBOL está diseñado alrededor de la aritmética decimal, a diferencia de la mayoría de los lenguajes que usan una representación interna binaria. La especificación COBOL exige cálculos decimales de punto fijo muy precisos, un aspecto del lenguaje que ha sido bien considerado en los sectores financieros. *COBOL también permite el USO BINARIO, pero se inclina hacia representaciones decimales (base-10).*

Historia

COBOL se remonta a finales de la década de 1950, con implementaciones iniciales publicadas en 1960.

La contraalmirante de la Armada de los EE. UU., Grace Hopper, a menudo se asocia con COBOL y se defiende en nombre del idioma durante las primeras etapas de desarrollo. No fue la única persona involucrada en el diseño y desarrollo de COBOL, de ninguna manera, pero a menudo se la conoce como la Madre de COBOL.

Debido al temprano respaldo de los gobiernos y las grandes corporaciones, COBOL se ha utilizado ampliamente durante muchas décadas. Sigue siendo un punto de orgullo para algunos y una espina para otros, que lo consideran obsoleto. La verdad probablemente se encuentra en algún lugar entre estas opiniones extremas. Cuando se aplica al procesamiento de transacciones, COBOL está en casa. Cuando se aplica a las pantallas web modernas y las aplicaciones de red, puede que no se sienta tan cómodo.

Estructura

Los programas COBOL están escritos en cuatro divisiones separadas.

- DIVISIÓN DE IDENTIFICACIÓN
- DIVISIÓN DE MEDIO AMBIENTE
- DIVISION DE DATOS
- DIVISION DE PROCEDIMIENTO

Descripciones de datos

Al estar diseñado para manejar datos decimales, COBOL permite descripciones de datos basadas en IMAGEN, en jerarquías agrupadas.

```
01 record-group.
   05 balance        pic s9(8)v99.
   05 rate           pic 999v999.
   05 show-balance   pic $Z(7)9.99.
```

Eso define el `balance` como un valor de ocho dígitos firmado con dos dígitos asumidos después del punto decimal. `rate` es de tres dígitos antes y tres dígitos después de un punto decimal supuesto. `show-balance` es un campo de edición numérica que tendrá un signo de dólar inicial, siete dígitos (cero suprimido) con al menos un dígito mostrado antes de dos dígitos después de un punto decimal.

`balance` se puede utilizar en los cálculos, `show-balance` es solo para fines de visualización y no se puede utilizar en instrucciones computacionales.

Declaraciones de procedimiento

COBOL es una palabra reservada de lenguaje pesado. El estilo MOVE, COMPUTE, MULTIPLY, PERFORM de forma larga conforman la mayoría de las especificaciones estándar. Más de 300 palabras clave y 47 declaraciones operativas en la especificación COBOL 2014. Muchas implementaciones de compiladores agregan aún más a la lista de palabras reservadas.

Examples

Hola Mundo

```
HELLO * HISTORIC EXAMPLE OF HELLO WORLD IN COBOL
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HELLO.
       PROCEDURE DIVISION.
           DISPLAY "HELLO, WORLD".
           STOP RUN.
```

Los días de diseño de las tarjetas perforadas y las entradas solo en mayúsculas están muy por detrás. Sin embargo, la mayoría de las implementaciones de COBOL todavía manejan el mismo diseño de código. Incluso las implementaciones actuales siguen lo mismo (a menudo incluso en mayúsculas,) compiladas y en producción.

Una implementación moderna bien formateada podría verse como:

```
*> Hello, world
identification division.
program-id. hello.

procedure division.
display "Hello, world"
goback.
end program hello.
```

Con algunas implementaciones de COBOL, esto se puede reducir a:

```
display "Hello, world".
```

Este formato generalmente requiere compiladores de tiempo para poner un compilador COBOL en un modo de sintaxis relajada, ya que faltan algunas de las declaraciones `DIVISION` normalmente

obligatorias.

COBOL asume las fuentes de formato FIJO de forma predeterminada, incluso en la especificación actual.

Pre-2002 COBOL

Columna	Zona
1-6	Área del número de secuencia
7	Área del indicador
8-12	Area a
12-72	Area b
73-80	Área de Nombre del Programa

Los editores de texto de mainframe de IBM todavía están configurados para este formulario en algunos casos.

Después de 2002 y en COBOL 2014, el Área A y B se fusionaron y se extendieron a la columna 255, y el Área de nombre del programa se eliminó.

Columna	Zona
1-6	Área del número de secuencia
7	Área del indicador
8-	Area de texto del programa

La columna 8 a través de una implementación definida en la columna *Margen R* , por lo general todavía está limitada a la columna 72, pero la especificación puede ejecutarla hasta la columna 255.

COBOL 2002 introdujo el texto fuente de `FORMAT FREE` . No hay *Área de Número de Secuencia* , *Área de Indicador* , y las líneas de origen pueden tener cualquier longitud (hasta un límite de *Margen R* definido por la implementación, por lo general menos de 2048 caracteres por línea, generalmente 255).

Pero el compilador comienza en el modo FORMATO FIJO por defecto. Por lo general, existe un *modificador de compilación* o una declaración de la *instalación de directiva del compilador* antes de que se reconozca la fuente de formato libre.

```
bbbbbb >>SOURCE FORMAT IS FREE
```

Donde `bbbbbb` representa 6 espacios en blanco, o cualquier otro carácter. (Estos se ignoran como parte del área de número de secuencia del modo de formato fijo predeterminado inicial).

Instalar gnu-cobol en Mac OS X

gnu-cobol está disponible a través del sistema homebrew.

Abra una ventana de terminal desde `/Applications/Utilities/Terminal` o use la tecla de `Command+Space` y escriba `"Terminal"` .

Si no tiene instalado el sistema homebrew, agréguelo escribiendo o copiando y pegando en su terminal:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Una vez que el comando ha terminado, escriba:

```
brew install gnu-cobol
```

Eso es todo, ahora puede compilar programas Cobol en su Mac.

Lea Empezando con Cobol en línea: https://riptutorial.com/es/cobol/topic/4728/empezando-con-cobol

Capítulo 2: ¿Cómo funciona el computacional en cobol?

Introducción

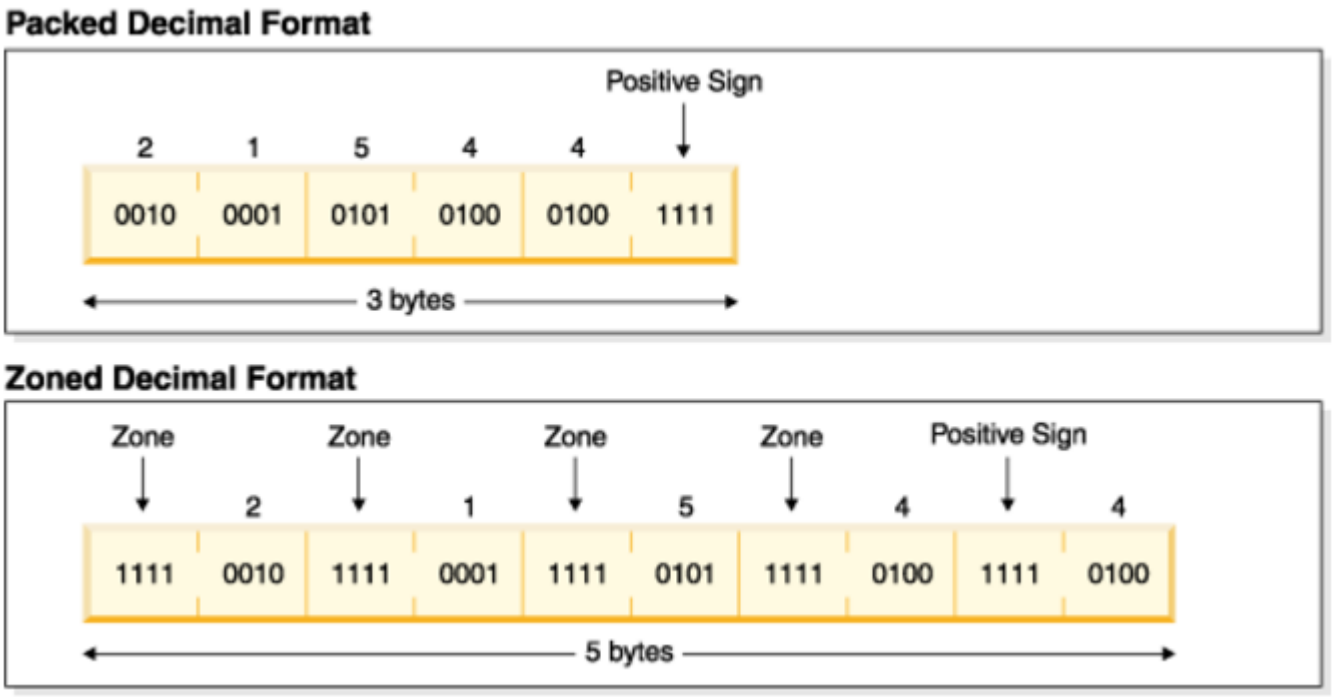
La cláusula computacional se usa para describir el tipo de almacenamiento utilizado en COBOL. Se utiliza para 3 formas: COMP-1, COMP-2 y COMP-3. La forma más común de computación es COMP-3. Con frecuencia es simplemente llamado "COMP" por los programadores.

Examples

COMP-3

El elemento de datos se almacena en formato decimal empaquetado en COMP-3. El formato decimal empaquetado significa que cada byte de almacenamiento (excepto el byte de orden bajo) puede contener dos números decimales. El byte de orden inferior contiene un dígito en la parte izquierda y el signo (positivo o negativo) en la parte derecha.

"Formato decimal zonificado" en la imagen de abajo es el almacenamiento predeterminado para un número en COBOL.




```
01 WS-NUM PIC 9(5) USAGE IS COMP-3 VALUE 21544.
```

El almacenamiento computacional se usa frecuentemente para reducir el tamaño de un archivo.

Implementaciones comunes

La implementación de comp, comp-1 ... comp-5 depende de la implementación.

```
Format     Normal Implementation

Comp       Big endian binary integer
Comp-1     4 byte floating point
Comp-2     8 byte floating point

Comp-3     Packed decimal 123 is stored as x'123c'

Comp-5     Binary Integer optermised for performance.
           Big Endian on the Mainframe, Little Endian on Intel Hardware
```

Los compiladores de Ibm normalmente admiten Comp, Comp-4, Comp-5 en tamaños de 2,4,8 bytes. Soporte GNU Cobolo con tamaños de 1,2,4,8.

Comp-1, los campos de Comp-2 se definen sin una cláusula de imagen:

```
03 Floating-Field      Comp-1.
03 Double-Field        Comp-2
```

Para otros Comp's se ingresa la imagen:

```
03 Big-Endian           Pic S9(4) Comp.
03 Packed-Decimal       Pic S9(5) Comp.
```

Lea ¿Cómo funciona el computacional en cobol? en línea:
https://riptutorial.com/es/cobol/topic/10873/-como-funciona-el-computacional-en-cobol-

Capítulo 3: Cuerda

Examples

STRINGVAL ... Mover -versus- STRING

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  STRINGVAL.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  WORK-AREAS.
    05 I-STRING        PIC X(08) VALUE   'STRNGVAL'.

    05 O-STRING        PIC XBXBXBXBXBXBXBX.
       88  O-STRING-IS-EMPTY     VALUE   SPACES.

PROCEDURE DIVISION.
GENESIS.

    PERFORM MAINLINE

    PERFORM FINALIZATION

    GOBACK

     .


MAINLINE.

    DISPLAY 'STRINGVAL EXAMPLE IS STARTING !!!!!!!!!!!!!!'

    DISPLAY '=== USING MOVE STATEMENT ==='
    MOVE I-STRING TO O-STRING
    DISPLAY 'O STRING= ' O-STRING

    DISPLAY '=== USING STRING STATEMENT ==='
    SET O-STRING-IS-EMPTY    TO  TRUE
    STRING I-STRING ( 1 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 2 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 3 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 4 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 5 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 6 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 7 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        I-STRING ( 8 : 1 ) DELIMITED BY SIZE
        ' ' DELIMITED BY SIZE
        INTO O-STRING
```

```
    DISPLAY 'O STRING= ' O-STRING

     .


FINALIZATION.

    DISPLAY 'STRINGVAL EXAMPLE IS COMPLETE !!!!!!!!!!!!!!'

     .

END PROGRAM STRINGVAL.
```

No es un ejemplo, pero ...

Parecía la única forma de añadir un comentario. Una cosa que es fácil de olvidar es que si encadena algunas variables como el ejemplo anterior, y la longitud resultante es MÁS PÚBLICA de lo que originalmente estaba en la variable receptora (cadena arriba), los caracteres "finales" se dejan en su lugar.

Por ejemplo, si la cadena o contenía "la cadena contiene estos datos" y usted "juntó" fred & Bert ", entonces la cadena o contendría" fred & Bert contiene estos datos "(si conté correctamente).

En resumen, adquiera el hábito de SIEMPRE mover espacios a su variable receptora antes de comenzar a encadenar.

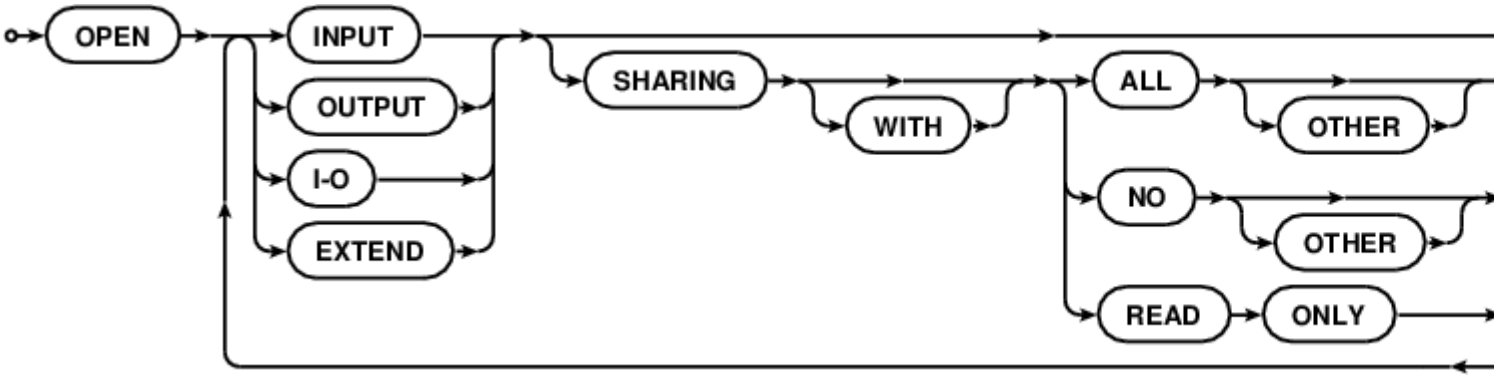
Lea Cuerda en línea: https://riptutorial.com/es/cobol/topic/7039/cuerda

Capítulo 4: Declaración ABIERTA

Observaciones

La sentencia COBOL `OPEN` inicia el procesamiento del archivo. Los recursos de archivos en COBOL se definen en la `ENVIRONMENT DIVISION` , nombrados en los párrafos `FD` (Descriptor de archivos). Estos nombres fd se utilizan para acceder a los archivos del disco físico y se especifican varias opciones en las cláusulas `SELECT` en el párrafo `FILE-CONTROL` de la `INPUT-OUTPUT SECTION` . Se espera que un programador pruebe un identificador de `FILE STATUS` para los códigos de estado y error.

Los modos incluyen `INPUT` , `OUTPUT` , `IO` y `EXTEND` .



Examples

Muestra ABIERTA, con mini informe LINAGE

```
COBOL ******************************************************************
      * Example of LINAGE File Descriptor
      * Tectonics: $ cocb -x linage.cob
      *            $ ./linage <filename ["linage.cob"]>
      *            $ cat -n mini-report
      ******************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. linage-demo.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           select optional data-file assign to file-name
               organization is line sequential
               file status is data-file-status.
           select mini-report assign to "mini-report".

       DATA DIVISION.
       FILE SECTION.
       FD  data-file.
       01  data-record.
           88 endofdata        value high-values.
           02 data-line         pic x(80).
```

```
FD  mini-report
    linage is 16 lines
        with footing at 15
        lines at top 2
        lines at bottom 2.
01  report-line         pic x(80).

WORKING-STORAGE SECTION.
01  command-arguments   pic x(1024).
01  file-name           pic x(160).
01  data-file-status    pic 99.
01  lc                  pic 99.
01  report-line-blank.
    02 filler           pic x(18) value all "*".
    02 filler           pic x(05) value spaces.
    02 filler           pic x(34)
        VALUE "THIS PAGE INTENTIONALLY LEFT BLANK".
    02 filler           pic x(05) value spaces.
    02 filler           pic x(18) value all "*".
01  report-line-data.
    02 body-tag         pic 9(6).
    02 line-3           pic x(74).
01  report-line-header.
    02 filler           pic x(6) VALUE "PAGE: ".
    02 page-no          pic 9999.
    02 filler           pic x(24).
    02 filler           pic x(5) VALUE " LC: ".
    02 header-tag       pic 9(6).
    02 filler           pic x(23).
    02 filler           pic x(6) VALUE "DATE: ".
    02 page-date        pic x(6).

01  page-count          pic 9999.

PROCEDURE DIVISION.

accept command-arguments from command-line end-accept.
string
    command-arguments delimited by space
    into file-name
end-string.
if file-name equal spaces
    move "linage.cob" to file-name
end-if.

open input data-file.
read data-file
    at end
        display "File: " function trim(file-name) " open error"
        go to early-exit
end-read.

open output mini-report.

write report-line
    from report-line-blank
end-write.

move 1 to page-count.
accept page-date from date end-accept.
move page-count to page-no.
```

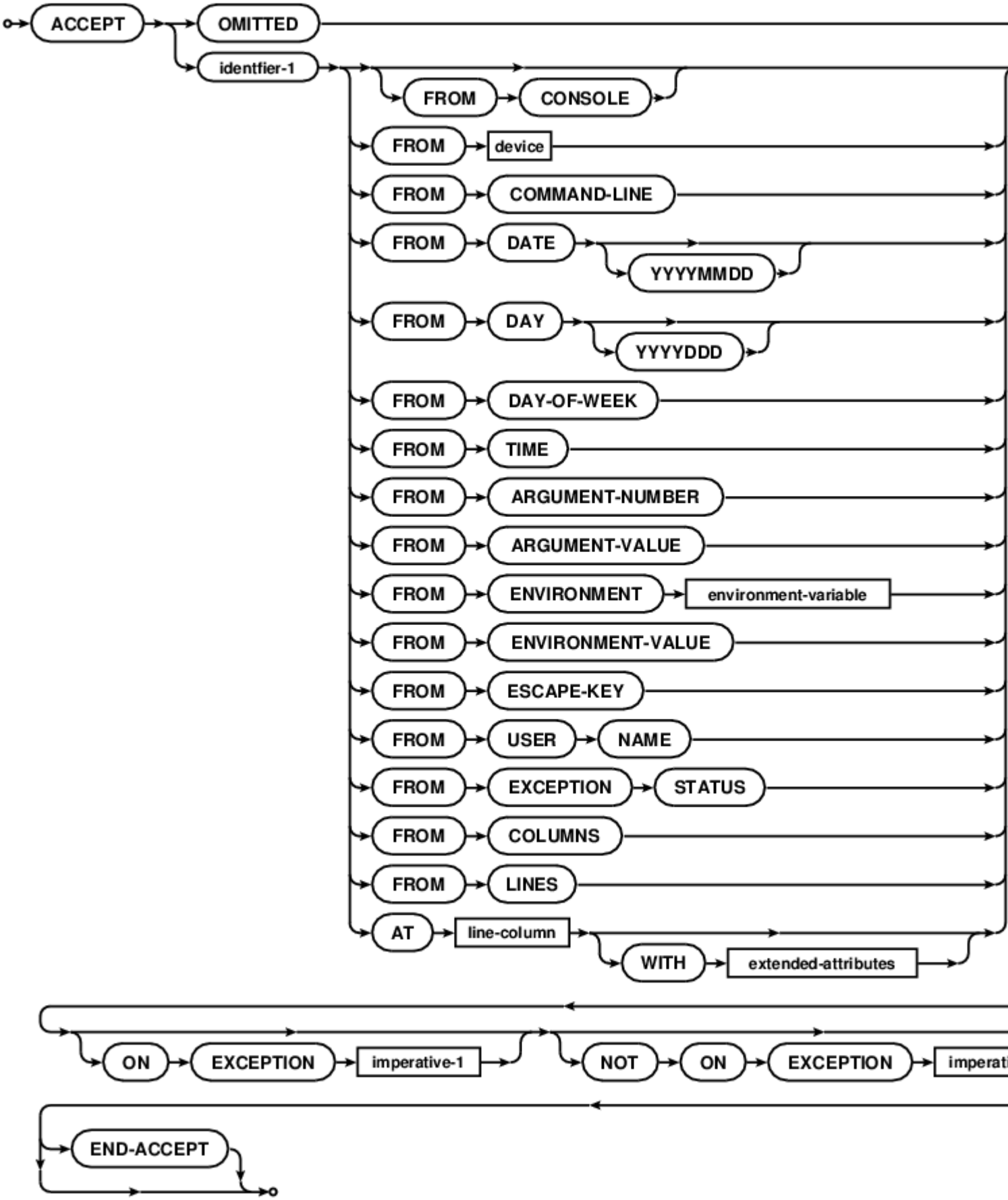
```
       write report-line
           from report-line-header
           after advancing page
       end-write.

       perform readwrite-loop until endofdata.

       display
           "Normal termination, file name: "
           function trim(file-name)
           " ending status: "
           data-file-status
       close mini-report.

      * Goto considered harmful?  Bah!  :)
       early-exit.
       close data-file.
       exit program.
       stop run.

      ****************************************************************
       readwrite-loop.
       move data-record to report-line-data
       move linage-counter to body-tag
       write report-line from report-line-data
           end-of-page
               add 1 to page-count end-add
               move page-count to page-no
               move linage-counter to header-tag
               write report-line from report-line-header
                   after advancing page
               end-write
       end-write
       read data-file
           at end set endofdata to true
       end-read
       .

      ****************************************************************
      * Commentary
      * LINAGE is set at a 20 line logical page
      *  16 body lines
      *   2 top lines
      *   A footer line at 15 (inside the body count)
      *   2 bottom lines
      * Build with:
      * $ cobc -x -Wall -Wtruncate linage.cob
      * Evaluate with:
      * $ ./linage
      * This will read in linage.cob and produce a useless mini-report
      * $ cat -n mini-report
      ****************************************************************
       END PROGRAM linage-demo.
```

Lea Declaración ABIERTA en línea: https://riptutorial.com/es/cobol/topic/7288/declaracion-abierta

Capítulo 5: Declaración ACCEPT

Observaciones

La sentencia COBOL ACCEPT se utiliza para recuperar datos del sistema.



Examples

Declaración ACCEPT

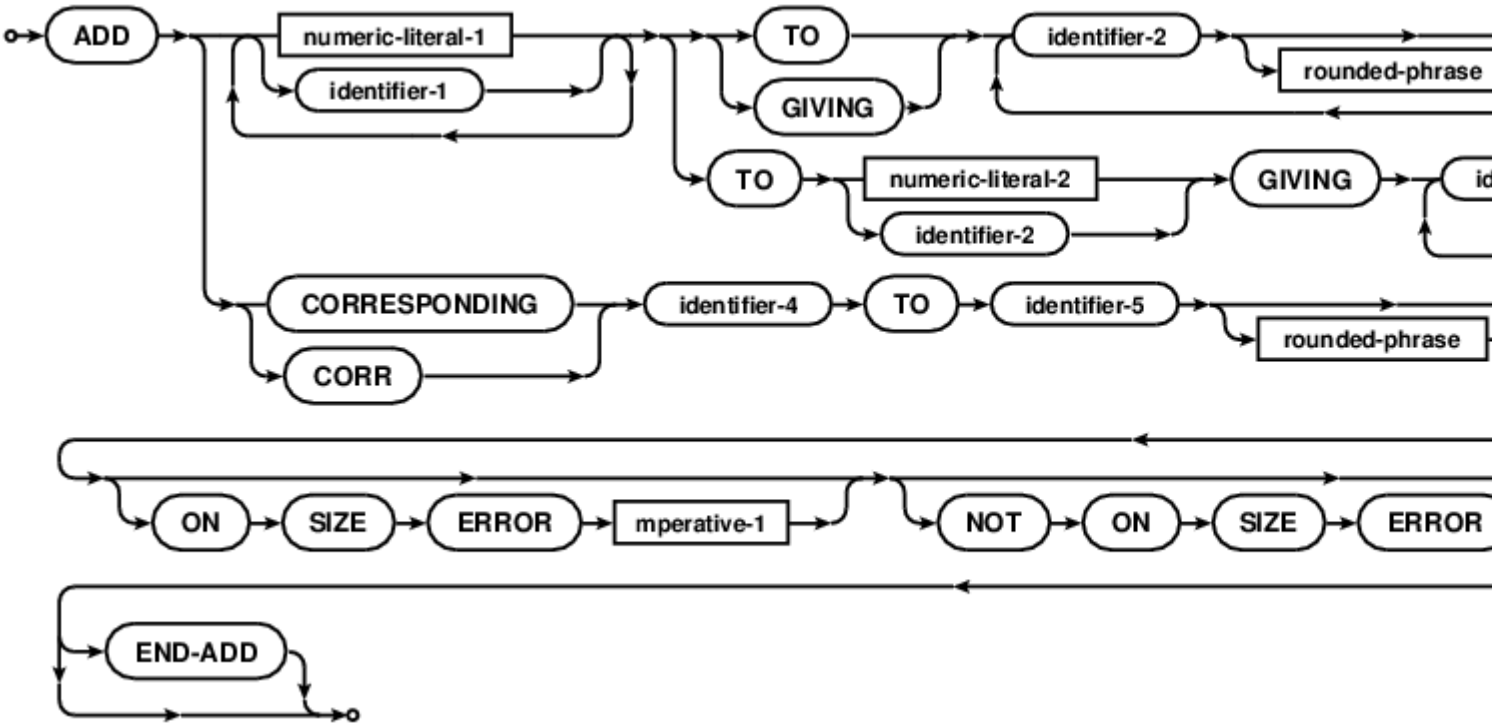
```
ACCEPT variable.
ACCEPT variable FROM CONSOLE.

ACCEPT variable FROM ENVIRONMENT "path".
ACCEPT variable FROM COMMAND-LINE.

ACCEPT variable FROM ARGUMENT-NUMBER
ACCEPT variable FROM ARGUMENT-VALUE

ACCEPT variable AT 0101.
ACCEPT screen-variable.

ACCEPT today FROM DATE.
ACCEPT today FROM DATE YYYYMMDD.
ACCEPT thetime FROM TIME.

ACCEPT theday FROM DAY.
ACCEPT theday FROM DAY YYYYDDD.

ACCEPT weekday FROM DAY-OF-WEEK.

ACCEPT thekey FROM ESCAPE KEY.

ACCEPT username FROM USER NAME.

ACCEPT exception-stat FROM EXCEPTION STATUS.

ACCEPT some-data FROM device-name.
```

Consulte http://open-cobol.sourceforge.net/faq/index.html#accept para obtener más detalles.

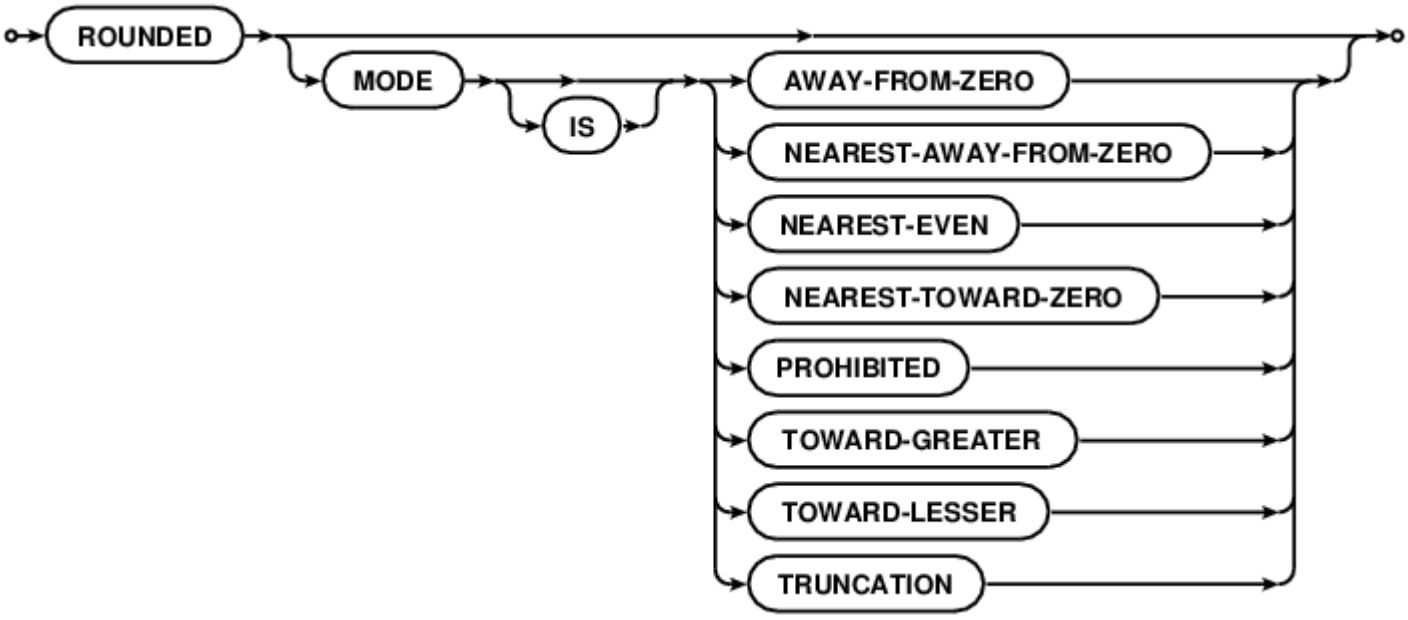
Lea Declaración ACCEPT en línea: https://riptutorial.com/es/cobol/topic/5512/declaracion-accept

Capítulo 6: Declaración ADD

Observaciones



Donde la fase redondeada es



Examples

Declaración ADD

```
ADD 1 TO cobol
```

Esto modifica la variable `cobol` . Desbordamiento silenciosamente ignorado.

```
ADD 1 TO cobol GIVING GnuCOBOL
```

Esto no modifica `cobol` , el resultado del ADD se almacena en `GnuCOBOL` . Nuevamente, el desbordamiento de la asignación de almacenamiento se ignora silenciosamente (el campo permanecerá en su valor anterior en los errores de tamaño y no se generará ninguna excepción).

```
ADD
    a b c d f g h i j k l m n o p q r s t u v w x y z
    GIVING total-of
    ON SIZE ERROR
        PERFORM log-problem
    NOT ON SIZE ERROR
        PERFORM graph-result
END-ADD
```

Se permiten múltiples entradas, con pruebas de tamaño de almacenamiento explícitas. COBOL tiene una `FUNCTION E` intrínseca, por lo que no es una opción inteligente para un identificador de una sola letra.

`SIZE ERROR` en COBOL depende del tipo y / o la `PICTURE` . Un campo `PIC 9` solo almacenará de forma segura los valores de 0 a 9, un resultado intermedio de 10 activará la frase `ON SIZE ERROR` en ese caso.

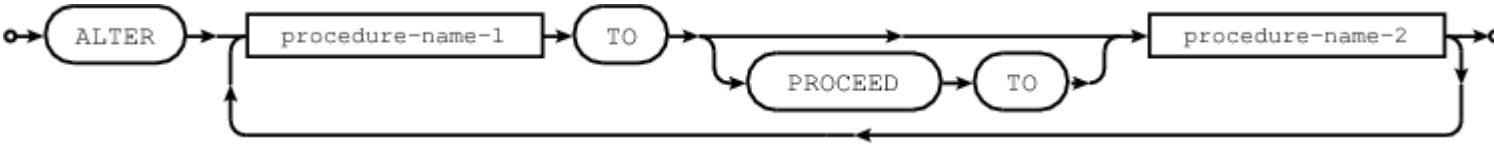
Lea Declaración ADD en línea: https://riptutorial.com/es/cobol/topic/5533/declaracion-add

Capítulo 7: Declaración ALTER

Observaciones

La muy amada declaración de ALTER. Cambia el objetivo de un párrafo IR A.

Ya no forma parte del estándar COBOL, todavía es compatible con muchos compiladores por razones de compatibilidad con versiones anteriores. *(El diagrama de sintaxis está atenuado para mostrar que esto ya no es COBOL estándar).*



Examples

Un ejemplo artificial usando ALTER.

```
 identification division.
 program-id. altering.
 date-written. 2015-10-28/06:36-0400.
 remarks. Demonstrate ALTER.

 procedure division.
 main section.

*> And now for some altering.
 contrived.
 ALTER story TO PROCEED TO beginning
 GO TO story
 .

*> Jump to a part of the story
 story.
 GO.
 .

*> the first part
 beginning.
 ALTER story TO PROCEED to middle
 DISPLAY "This is the start of a changing story"
 GO TO story
 .

*> the middle bit
 middle.
 ALTER story TO PROCEED to ending
 DISPLAY "The story progresses"
 GO TO story
 .

*> the climatic finish
```

```
 ending.
 DISPLAY "The story ends, happily ever after"
 .

*> fall through to the exit
 exit program.
```

Con una muestra de ejecución de

```
prompt$ cobc -xj -debug altering.cob
This is the start of a changing story
The story progresses
The story ends, happily ever after

prompt$ COB_SET_TRACE=Y ./altering
Source:     'altering.cob'
Program-Id: altering         Entry:     altering              Line: 8
Program-Id: altering         Section:   main                  Line: 8
Program-Id: altering         Paragraph: contrived             Line: 11
Program-Id: altering         Statement: ALTER                 Line: 12
Program-Id: altering         Statement: GO TO                 Line: 13
Program-Id: altering         Paragraph: story                 Line: 17
Program-Id: altering         Paragraph: beginning             Line: 22
Program-Id: altering         Statement: ALTER                 Line: 23
Program-Id: altering         Statement: DISPLAY               Line: 24
This is the start of a changing story
Program-Id: altering         Statement: GO TO                 Line: 25
Program-Id: altering         Paragraph: story                 Line: 17
Program-Id: altering         Paragraph: middle                Line: 29
Program-Id: altering         Statement: ALTER                 Line: 30
Program-Id: altering         Statement: DISPLAY               Line: 31
The story progresses
Program-Id: altering         Statement: GO TO                 Line: 32
Program-Id: altering         Paragraph: story                 Line: 17
Program-Id: altering         Paragraph: ending                Line: 36
Program-Id: altering         Statement: DISPLAY               Line: 37
The story ends, happily ever after
Program-Id: altering         Statement: EXIT PROGRAM          Line: 41
Program-Id: altering         Exit:      altering
prompt$
```

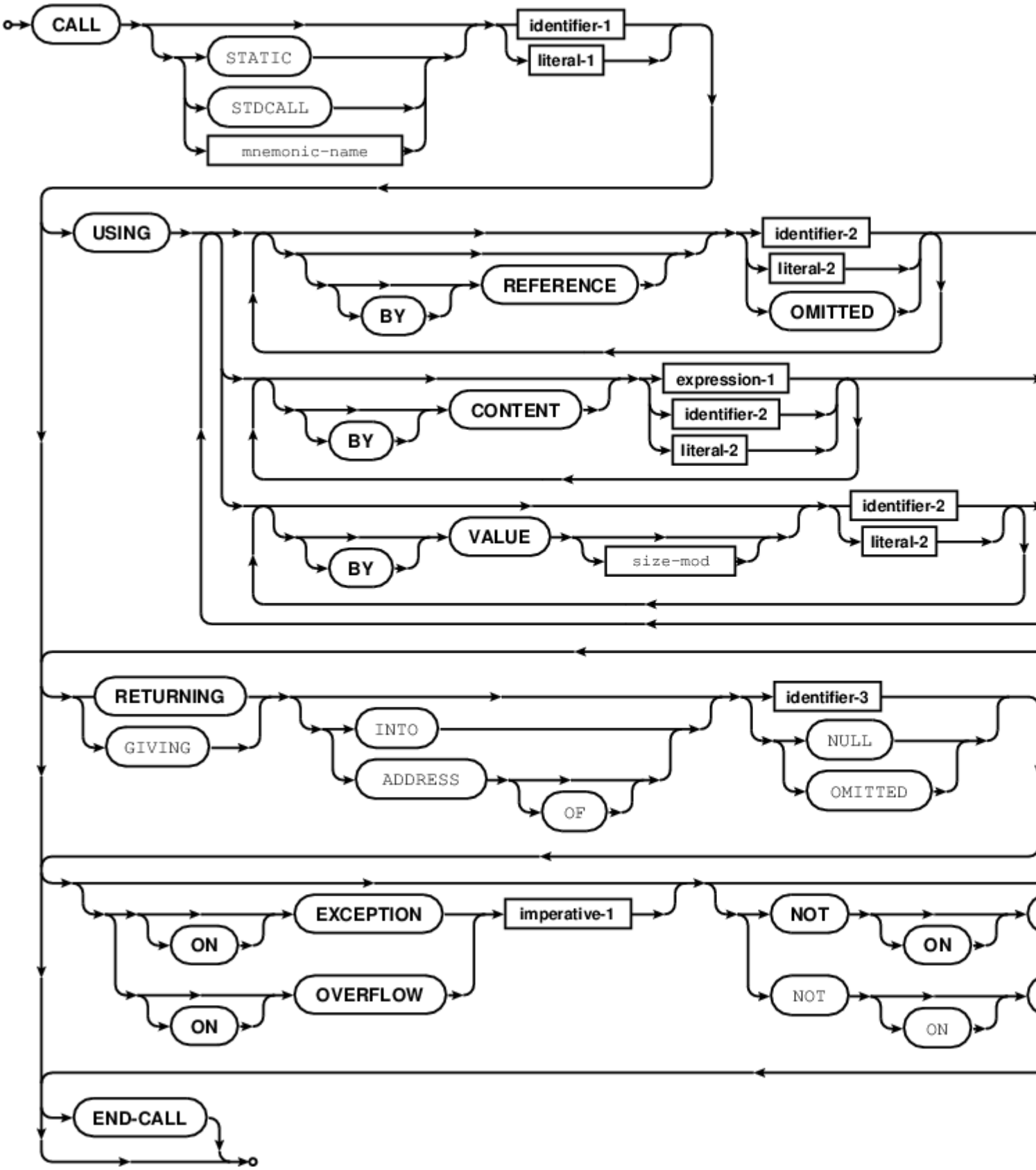
Consulte http://open-cobol.sourceforge.net/faq/index.html#alter para obtener más detalles.

Lea Declaración ALTER en línea: https://riptutorial.com/es/cobol/topic/5584/declaracion-alter

Capítulo 8: Declaración CALL

Observaciones

La instrucción COBOL CALL proporciona acceso a las rutinas de la biblioteca compilada.



Examples

Declaración CALL

COBOL puede usar enlaces estáticos para la siguiente declaración. GnuCOBOL utiliza el enlace dinámico de forma predeterminada para todos los símbolos externos conocidos en el momento de

la compilación, incluso cuando el símbolo es un literal:

```
CALL "subprogram" USING a b c *> run a (possibly static linked) sub program
                              *> passing three fields

CALL some-prog USING a b c    *> some-prog is a PIC X item and can be changed
                              *> at run-time to do a dynamic lookup
```

Esta declaración obliga a compilar la resolución de edición del enlace de tiempo. *(No estándar, extensión de sintaxis)* :

```
CALL STATIC "subprogram" USING a b c
```

Los campos en COBOL se pueden pasar `BY REFERENCE` (el valor predeterminado, hasta que se invalida - las anulaciones se `sticky` en un orden de izquierda a derecha), `BY CONTENT` (se pasa una copia POR REFERENCIA), o en algunos casos directamente `BY VALUE` :

```
CALL "calculation" USING BY REFERENCE a BY VALUE b BY CONTENT c RETURNING d
    ON EXCEPTION DISPLAY 'No linkage to "calculation"' UPON SYSERR
END-CALL
```

COBOL está diseñado para ser un lenguaje `BY REFERENCE` , por lo que el uso de `BY VALUE` puede presentar problemas. Por ejemplo, los números literales no tienen un tipo explícito y la especificación COBOL no tiene reglas de promoción de tipo explícitas. Por lo tanto, los desarrolladores tienen que preocuparse por la configuración del marco de llamada con `BY VALUE` de los literales.

Consulte http://open-cobol.sourceforge.net/faq/index.html#call para obtener más detalles.

TIEMPO DE DORMIR

CALL también es una forma de ampliar la funcionalidad COBOL, y también para permitir la reutilización del código. También puede dar acceso a la funcionalidad del "sistema".

Este ejemplo ilustra las formas de proporcionar la funcionalidad de "suspensión" a los COBOL de mainframe de IBM. Tenga en cuenta que el requisito de hacerlo es raro en la medida en que, por lo general, cuando alguien piensa que necesita "dormir" por alguna razón, es algo incorrecto.

ILBOWAT0 es de la antigua era del tiempo de ejecución específica de COBOL en Mainframes. BXP1SLP y BXP4SLP son rutinas de Servicios del Sistema Unix (USS) que pueden ser utilizadas por cualquier idioma. En efecto, son solicitudes de "dormir" de Unix.

El actual IBM Mainframe Runtime (Language Environment (LE)) proporciona comunicación entre idiomas, y los servicios de CEE3DLY LE se muestran en otro ejemplo, Uso del servicio de retardo de subprocesos de z / OS Language Environment .

ILBOWAT0 ha existido por mucho tiempo (quizás más de 40 años), y todavía puedes encontrarlo. Su uso debe ser reemplazado por CEE3DLY o BXP1SLP, lo que sea más apropiado para el requisito particular.

A veces, necesita hacer que un programa se duerma, o hacer que un Trabajo se duerma por un tiempo (después de un paso de FTP o NDM), que generalmente se ejecutan como trabajos separados, y tendría que estar en modo de suspensión / bucle buscando los conjuntos de datos resultantes.

Aquí hay un pequeño y lindo programa COBOL para realizar dicha tarea, llamando a los programas de suspensión COBOL disponibles en OS / VS y quizás en otros entornos operativos heredados y de mainframe actuales.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SLEEPYTM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  WAIT-PARM.
           05  WAIT-TIME            PIC S9(8) COMP VALUE 90.
           05  WAIT-RESPONSE        PIC S9(8) COMP VALUE 0.
           05  WAIT-PROGRAM-24BIT   PIC  X(8)      VALUE 'ILBOWAT0'.
           05  WAIT-PROGRAM-31BIT   PIC  X(8)      VALUE 'BPX1SLP '.
           05  WAIT-PROGRAM-64BIT   PIC  X(8)      VALUE 'BPX4SLP '.

       PROCEDURE DIVISION.
       GENESIS.
           DISPLAY 'START CALLING WAIT PROGRAM'
           CALL WAIT-PROGRAM-24BIT USING WAIT-TIME WAIT-RESPONSE
           DISPLAY 'END   CALLING WAIT PROGRAM'
           GOBACK
PERIOD     .
```

forma de microfoco

Para Microfocus, utiliza la API "SleepEx". Como ejemplo;

```
environment division.
special-names.
    call-convention 74 is winAPI.
         :
         :
01  wSleep-time              pic 9(8) comp-5.
01  wSleep-ok                pic 9(8) comp-5.
         :
         :
move 10000 to wSleep-time  *>10seconds
call winAPI "SleepEx" using by value wSleep-time
                        by value 0 size 4
             returning wSleep-ok
end-call.
```

Uso del servicio de retardo de subprocesos del entorno de idioma de z / OS

Puede llamar al servicio CEE3DLY en modo de 24-31 o 64 bits para retrasar una tarea al segundo más cercano. Es guardar CICS y solo retrasará el hilo.

Un ejemplo:

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID.  SLEEPYTM.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  WAIT-PARM.
      05  WAIT-SECS            PIC S9(8) COMP VALUE 90.
      05  WAIT-FC              PIC X(12).

    PROCEDURE DIVISION.

      CALL CEE3DLY USING WAIT-SECS WAIT-FC

      GOBACK.
```

Puedes ver más detalles aquí:

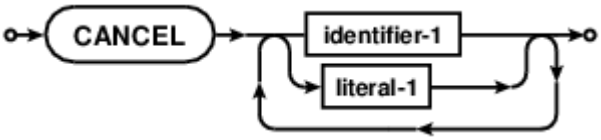
Servicios invocables de IBM Language Environment - Suspender

Lea Declaración CALL en línea: https://riptutorial.com/es/cobol/topic/5601/declaracion-call

Capítulo 9: Declaración CANCEL

Observaciones

La instrucción CANCEL asegura que un programa referenciado estará en un estado inicial la próxima vez que se llame, y que descargue cualquier recurso para el módulo.



Examples

Declaración CANCEL

```
CALL "submodule"
CALL "submodule"

CANCEL "submodule"
CALL "submodule"
```

Cualquier dato estático en el conjunto de trabajo del `submodule` estará en un estado inicial en la última instrucción `CALL` anterior. La segunda `CALL` tendrá cualquier valor inicial establecido como sobras de la primera `CALL` .

Los compiladores COBOL pueden admitir la cancelación física (objeto descargado de la memoria) y / o la cancelación virtual (garantizar un estado inicial, pero dejar el objeto disponible para el entorno operativo del host). Este es un detalle de implementación.

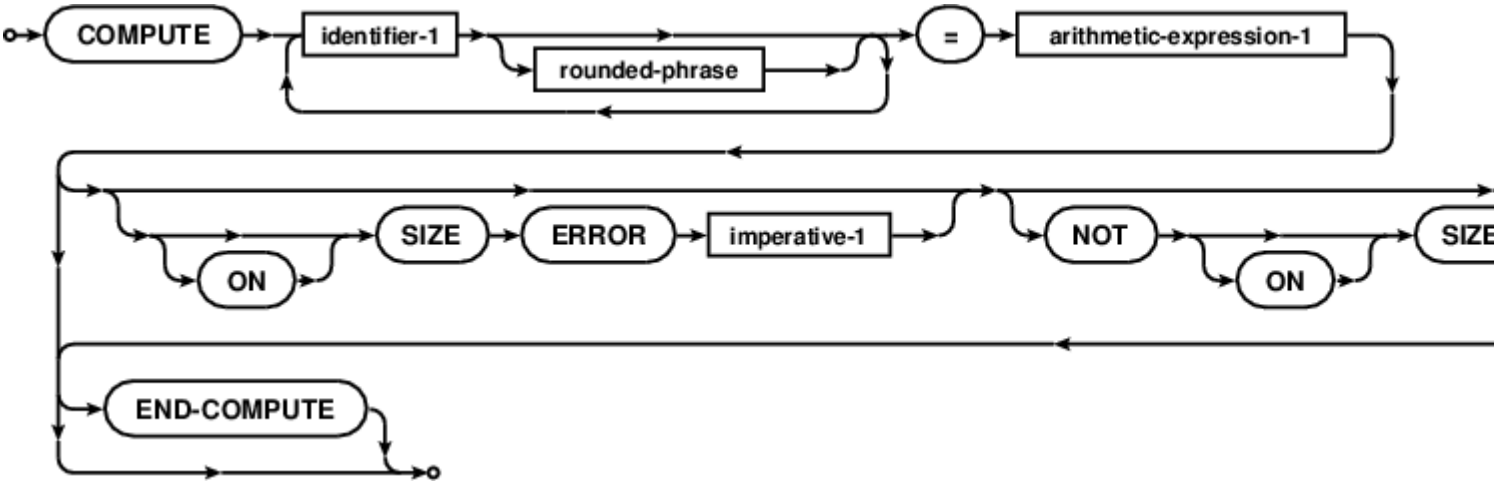
Consulte http://open-cobol.sourceforge.net/faq/index.html#cancel para obtener más detalles.

Lea Declaración CANCEL en línea: https://riptutorial.com/es/cobol/topic/5600/declaracion-cancel

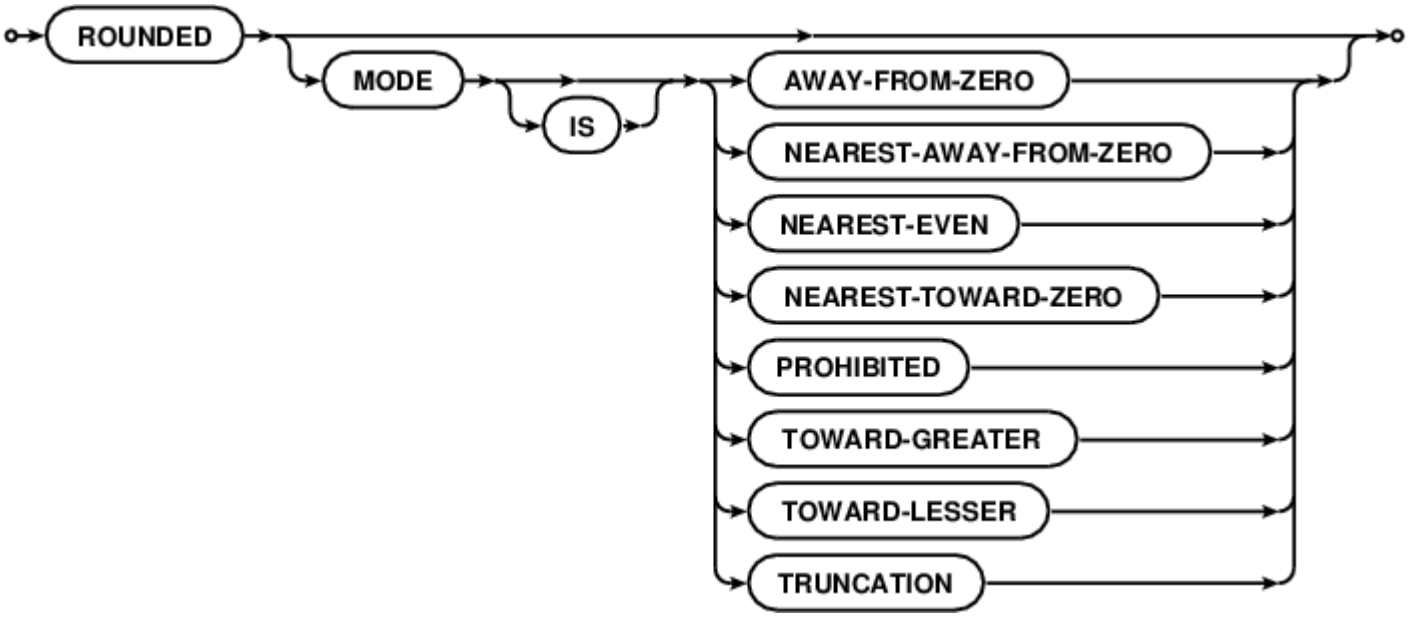
Capítulo 10: Declaración COMPUTE

Observaciones

La instrucción COMPUTE permite expresiones de cálculo algebraico.



Frase redondeada es



Examples

Consejo: Usa espacios alrededor de todos los componentes.

```
COMPUTE answer = 3*var-1
```

Esa es una referencia a la variable `var-1` , y no `var - 1` .

```
COMPUTE answer = 3 * var - 1
```

Recomendado, *opinión* .

Lea Declaración COMPUTE en línea: https://riptutorial.com/es/cobol/topic/6726/declaracion-compute

Capítulo 11: Declaración CONTINUAR

Observaciones

La instrucción CONTINUE hace que el flujo de control continúe en la siguiente instrucción. No es un no-op, ya que puede influir en el flujo de control cuando está dentro de secuencias de sentencias compuestas, en particular IF / THEN / ELSE.

Un práctico? el ejemplo es durante el desarrollo temprano y la construcción con y sin ayudas de depuración.

```
CALL "CBL_OC_DUMP" USING structure ON EXCEPTION CONTINUE END-CALL
```

Ese código, aunque costoso, permitirá realizar volcados de memoria con formato cuando el módulo `CBL_OC_DUMP` esté vinculado al ejecutable, pero fallará de manera inofensiva cuando no lo esté. * Ese truco solo es aplicable durante las primeras etapas de desarrollo. El costo de una falla en la búsqueda dinámica no es algo que deba dejarse en el código activo, y esas líneas deben eliminarse de la fuente tan pronto como se satisfagan las preocupaciones iniciales en las pruebas alfa. En el primer día de codificación, puede ser una ayuda útil. Para el segundo día, la codificación EN EXCEPCIÓN CONTINUAR las incidencias debe limpiarse.

Examples

Marcador de posición

Esto es ideado; pero algunos programadores COBOL pueden preferir la claridad positiva, en lugar de usar `NOT` en expresiones condicionales (especialmente con la lógica propensa al error `var NOT = value OR other-value`).

```
if action-flag = "C" or "R" or "U" or "D"
    continue
else
    display "invalid action-code" upon syserr
    perform report-exception
    exit section
end-if
```

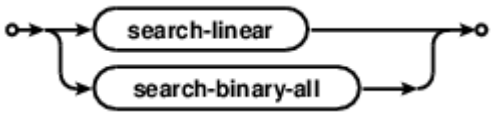
Lea Declaración CONTINUAR en línea: https://riptutorial.com/es/cobol/topic/6981/declaracion-continuar

Capítulo 12: Declaración de búsqueda

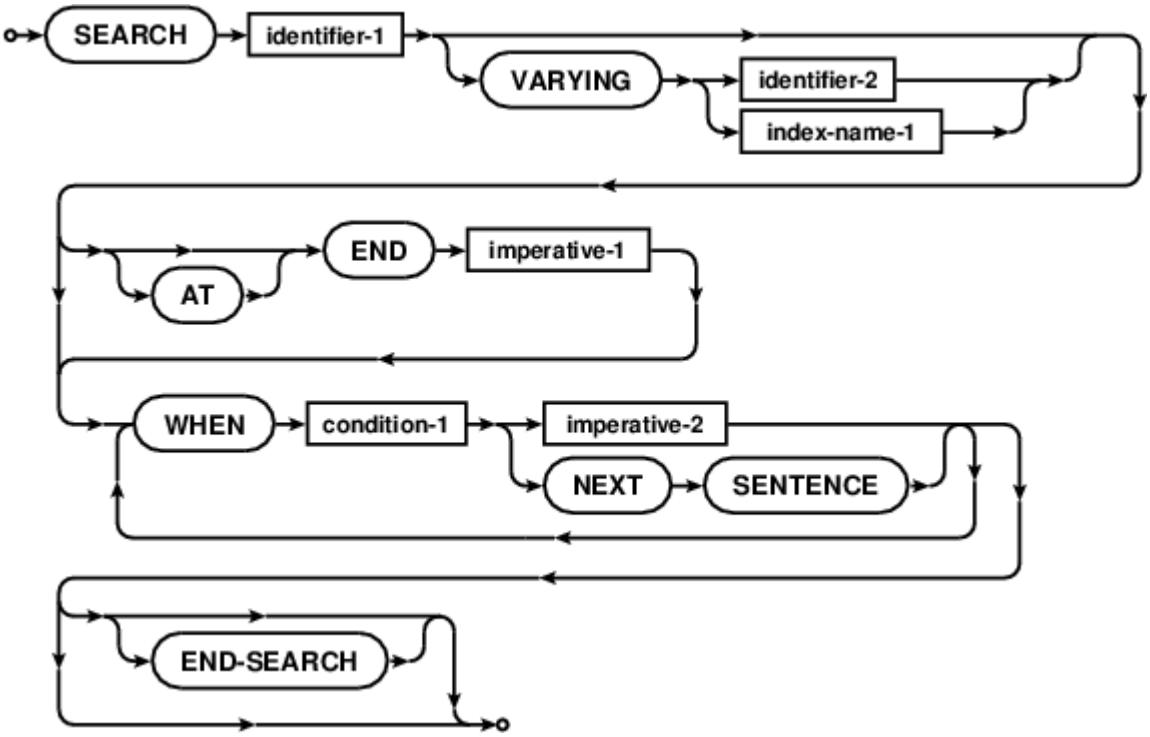
Observaciones

La instrucción COBOL `SEARCH` viene en dos formas. `SEARCH` lineal de arriba a abajo y un algoritmo binario de `SEARCH ALL` . SEARCH ALL binario asume una tabla ordenada adecuada para una búsqueda binaria sin elementos fuera de orden.

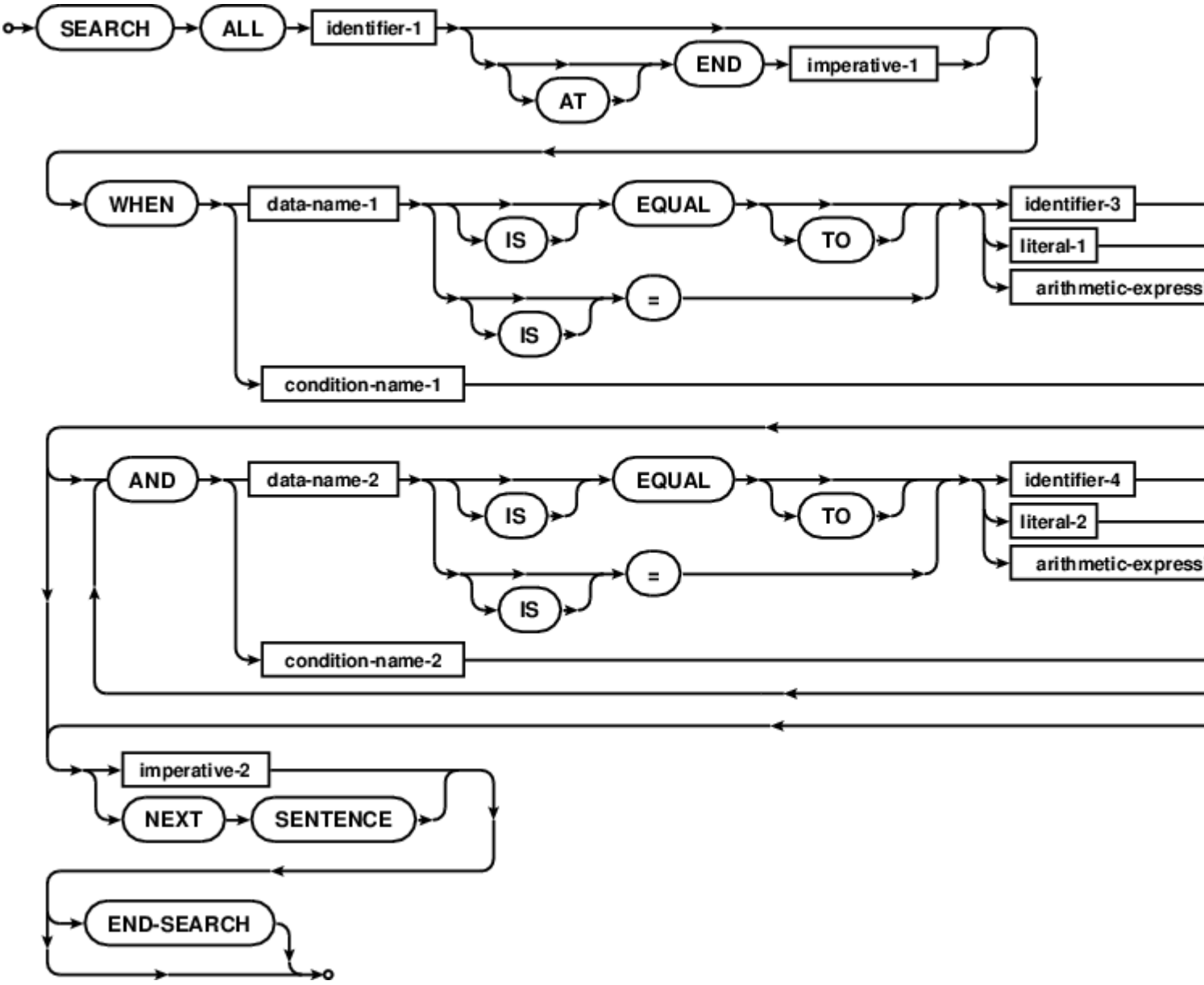
Declaración de búsqueda



Busqueda lineal



Binario BÚSQUEDA TODO



Examples

Busqueda lineal

```
GCobol >>SOURCE FORMAT IS FIXED
      *> ***************************************************************
      *> Purpose:   Demonstration of the SEARCH verb
      *> Tectonics: cobc -x searchlinear.cob
      *> ***************************************************************
       identification division.
       program-id. searchlinear.

       data division.

       working-storage section.
       01 taxinfo.
          05 tax-table occurs 4 times indexed by tt-index.
             10 province       pic x(2).
             10 taxrate        pic 999v9999.
             10 federal        pic 999v9999.
```

```
       01 prov                 pic x(2).
       01 percent              pic 999v9999.
       01 percentage           pic zz9.99.

      *> ****************************************************************
       procedure division.
       begin.

      *> ****************************************************************
      *> Sample for linear SEARCH, requires INDEXED BY table
      *> populate the provincial tax table;
      *>  *** (not really, only a couple of sample provinces) ***
      *> populate Ontario and PEI using different field loaders
       move 'AB' to province(1)
       move 'ON' to province(2)
       move 0.08 to taxrate(2)
       move 0.05 to federal(2)
       move 'PE00014000000000' to tax-table(3)
       move 'YT' to province(4)

      *> Find Ontario tax rate
       move "ON" to prov
       perform search-for-taxrate

      *> Setup for Prince Edward Island
       move 'PE' to prov
       perform search-for-taxrate

      *> Setup for failure
       move 'ZZ' to prov
       perform search-for-taxrate

       goback.
      *> ****************************************************************

       search-for-taxrate.
           set tt-index to 1
           search tax-table
               at end display "no province: " prov end-display
               when province(tt-index) = prov
                   perform display-taxrate
           end-search
       .

       display-taxrate.
           compute percent = taxrate(tt-index) * 100
           move percent to percentage
           display
               "found: " prov " at " taxrate(tt-index)
               "," percentage "%, federal rate of " federal(tt-index)
           end-display
       .

       end program searchlinear.
```

Binario BÚSQUEDA TODO

```
GCobol >>SOURCE FORMAT IS FIXED
      *> ****************************************************************
```

```
*> Purpose:   Demonstration of the SEARCH ALL verb and table SORT
*> Tectonics: cobc -x -fdebugging-line searchbinary.cob
*> ************************************************************
 identification division.
 program-id. searchbinary.

 environment division.
 input-output section.
 file-control.
     select optional wordfile
     assign to infile
     organization is line sequential.

 data division.
 file section.
 fd wordfile.
     01 wordrec          pic x(20).

 working-storage section.
 01 infile               pic x(256) value spaces.
    88 defaultfile       value '/usr/share/dict/words'.
 01 arguments            pic x(256).

*> Note the based clause, this memory is initially unallocated
 78 maxwords            value 500000.
 01 wordlist             based.
    05 word-table occurs maxwords times
        depending on wordcount
        descending key is wordstr
        indexed by wl-index.
       10 wordstr        pic x(20).
       10 wordline       usage binary-long.
 01 wordcount            usage binary-long.

 01 file-eof             pic 9 value low-value.
    88 at-eof            value high-values.

 01 word                 pic x(20).

*> ************************************************************
 procedure division.
 begin.

*> Get the word file filename
 accept arguments from command-line end-accept
 if arguments not equal spaces
     move arguments to infile
 else
     set defaultfile to true
 end-if

*> ************************************************************
*> Try playing with the words file and binary SEARCH ALL
*>   requires KEY IS and INDEXED BY table description

*> Point wordlist to valid memory
 allocate wordlist initialized

 open input wordfile

 move low-value to file-eof
```

```
       read wordfile
           at end set at-eof to true
       end-read

       perform
           with test before
           until at-eof or (wordcount >= maxwords)
               add 1 to wordcount
               move wordrec to wordstr(wordcount)
               move wordcount to wordline(wordcount)
               read wordfile
                   at end set at-eof to true
               end-read
       end-perform

       close wordfile

      *> ensure a non-zero length table when allowing optional file
       evaluate true                   also file-eof
           when wordcount = 0          also any
               move 1 to wordcount
               display "No words loaded" end-display
           when wordcount >= maxwords also low-value
               display "Word list truncated to " maxwords end-display
       end-evaluate

      >>D display "Count: " wordcount ": " wordstr(wordcount) end-display

      *> Sort the words from z to a
       sort word-table on descending key wordstr

      *> fetch a word to search for
       display "word to find: " with no advancing end-display
       accept word end-accept

      *> binary search the words for word typed in and display
      *> the original line number if/when a match is found
       set wl-index to 1
       search all word-table
           at end
               display
                   word " not a word of " function trim(infile)
               end-display
           when wordstr(wl-index) = word
               display
                   word " sorted to " wl-index ", originally "
                   wordline(wl-index) " of " function trim(infile)
               end-display
       end-search

      *> Release memory ownership
       free address of wordlist

       goback.
       end program searchbinary.
```

Lea Declaración de búsqueda en línea: https://riptutorial.com/es/cobol/topic/7462/declaracion-de-busqueda

Capítulo 13: Declaración de compromiso

Observaciones

Vacía TODOS los bloqueos actuales, sincronizando buffers de E / S de archivos.

Esta es una extensión no estándar, disponible con algunas implementaciones COBOL que admiten funciones `ROLLBACK` .

Examples

Declaración de compromiso

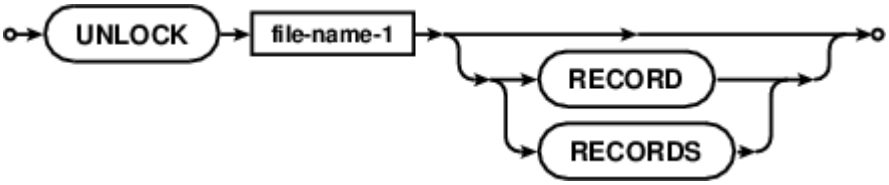
```
WRITE record
COMMIT
```

Lea Declaración de compromiso en línea: https://riptutorial.com/es/cobol/topic/6357/declaracion-de-compromiso

Capítulo 14: Declaración de desbloqueo

Observaciones

La declaración `UNLOCK` libera explícitamente cualquier bloqueo de registro asociado con un conector de archivo.



Examples

Desbloquear registro desde un conector de archivo

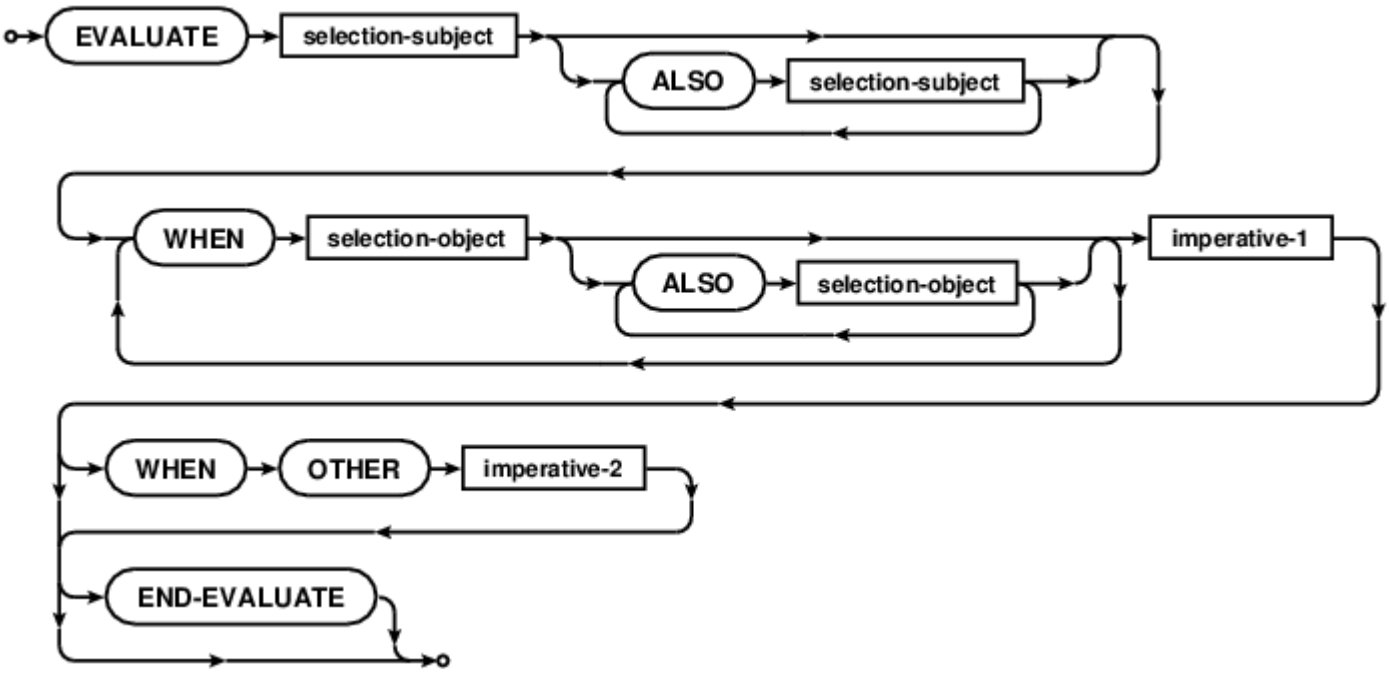
```
UNLOCK filename-1 RECORDS
```

Lea Declaración de desbloqueo en línea: https://riptutorial.com/es/cobol/topic/7471/declaracion-de-desbloqueo

Capítulo 15: Declaración de evaluación

Observaciones

La instrucción `EVALUATE` es una estructura de prueba y selección de rama múltiple, unión múltiple, condicional.



Examples

Una condicion de tres condiciones

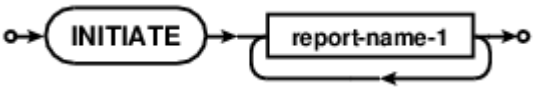
```
EVALUATE a ALSO b ALSO TRUE
    WHEN 1 ALSO 1 THRU 9 ALSO c EQUAL 1 PERFORM all-life
    WHEN 2 ALSO 1 THRU 9 ALSO c EQUAL 2 PERFORM life
    WHEN 3 THRU 9 ALSO 1 ALSO c EQUAL 9 PERFORM disability
    WHEN OTHER PERFORM invalid
END-EVALUATE
```

Lea Declaración de evaluación en línea: https://riptutorial.com/es/cobol/topic/7083/declaracion-de-evaluacion

Capítulo 16: Declaración de INICIACIÓN

Observaciones

La instrucción `INITIATE` inicializa los campos de control internos del `Report Writer` . La mayor parte de la configuración de un escritor de informes se produce en la `DATA DIVISION` con declaraciones muy breves de `PROCEDURE DIVISION` . Una vez inicializado, `GENERATE` realiza todo el trabajo duro de control de ruptura y paginación de informes.



Examples

INICIAR reportando variables de control

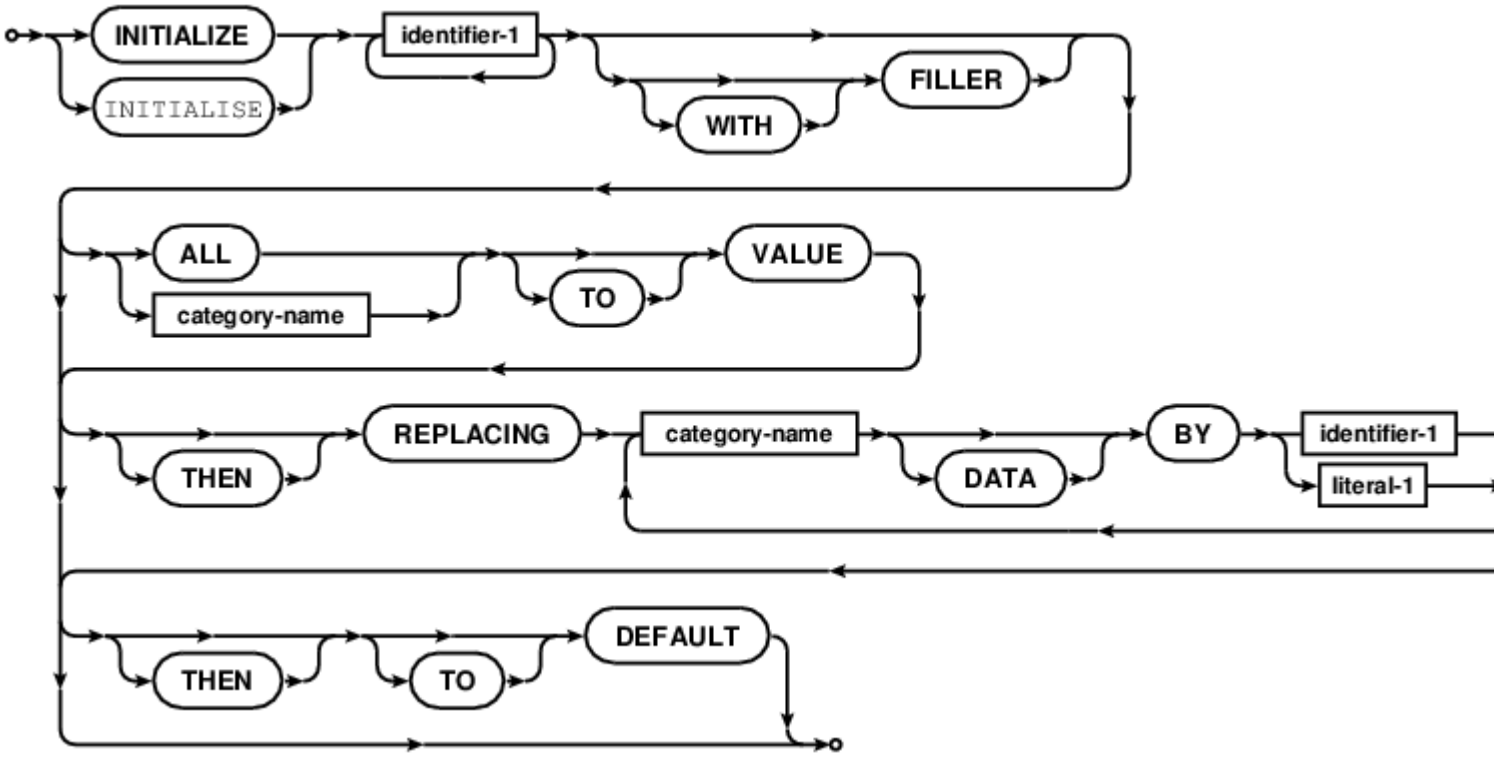
```
INITIATE report-1 report-2
```

Lea Declaración de INICIACIÓN en línea: https://riptutorial.com/es/cobol/topic/7180/declaracion-de-iniciacion

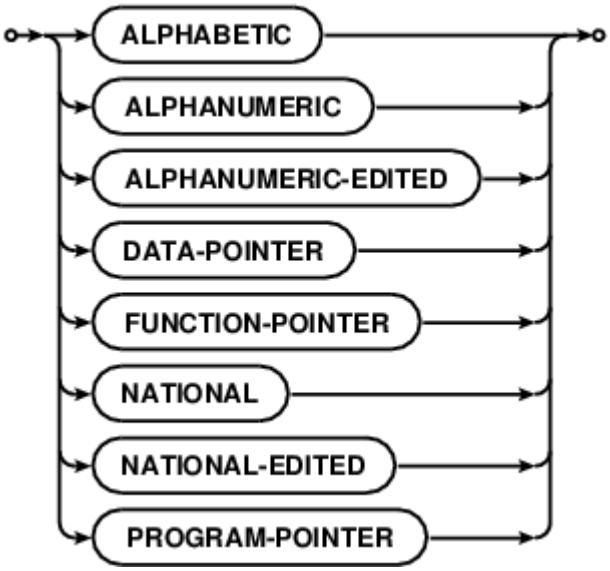
Capítulo 17: Declaración de inicialización

Observaciones

La instrucción `INITIALIZE` establece los datos seleccionados en valores específicos.



Donde `category-name` es:



Examples

Varias cláusulas de INICIALIZACIÓN

```
01  fillertest.
    03 fillertest-1 PIC 9(10) value 2222222222.
    03 filler       PIC X     value '|'.
    03 fillertest-2 PIC X(10) value all 'A'.
    03 filler       PIC 9(03) value 111.
    03 filler       PIC X     value '.'.

INITIALIZE fillertest

INITIALIZE fillertest REPLACING NUMERIC BY 9

INITIALIZE fillertest REPLACING ALPHANUMERIC BY 'X'

INITIALIZE fillertest REPLACING ALPHANUMERIC BY ALL 'X'

INITIALIZE fillertest WITH FILLER

INITIALIZE fillertext ALL TO VALUE
```

Dando:

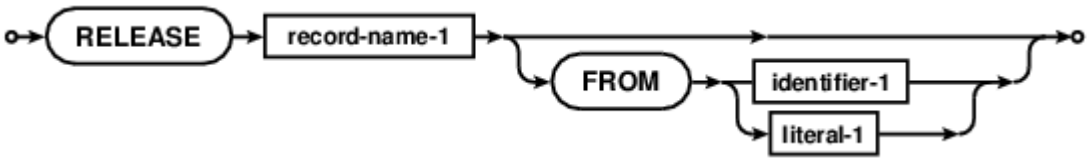
```
fillertest on start:
2222222222|AAAAAAAAAA111.
fillertest after initialize:
0000000000|          111.
fillertest after initialize replacing numeric by 9:
0000000009|          111.
fillertest after initialize replacing alphanumeric by "X":
0000000009|X         111.
fillertest after initialize replacing alphanumeric by all "X":
0000000009|XXXXXXXXXX111.
fillertest after initialize with filler:
0000000000           000
fillertest after initialize all to value:
2222222222|AAAAAAAAAA111.
```

Lea Declaración de inicialización en línea: https://riptutorial.com/es/cobol/topic/7179/declaracion-de-inicializacion

Capítulo 18: Declaración de LIBERACIÓN

Observaciones

La instrucción `RELEASE` se utiliza para dar registros al algoritmo COBOL `SORT` en condiciones controladas por el programador.



Examples

LIBERAR un registro a un PROCEDIMIENTO DE ENTRADA DE ORDENACIÓN

Esta es una muestra artificial. Ordena los registros en función de un `ALPHABET` que tiene mayúsculas y minúsculas juntas, con `A` y `a` intercambio en comparación con las otras letras. Esto se hizo a propósito para demostrar las posibilidades. El lector de algoritmos SORT recupera registros utilizando `RELEASE` en el `INPUT PROCEDURE RELEASE INPUT PROCEDURE` . El `OUTPUT PROCEDURE` utiliza `RETURN` para el escritor del algoritmo `SORT` .

```
GCobol >>SOURCE FORMAT IS FIXED
      *****************************************************************
      * Purpose:   A GnuCOBOL SORT verb example
      * Tectonics: cobc -x sorting.cob
      *     ./sorting <input >output
      *   or simply
      *     ./sorting
      *   for keyboard and screen demos
      *****************************************************************
       identification division.
       program-id. sorting.

       environment division.
       configuration section.
      * This sets up a sort order lower/upper except for "A" and "a"
       special-names.
           alphabet mixed is " AabBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTu
      -"UvVwWxXyYzZ0123456789".

       input-output section.
       file-control.
           select sort-in
               assign keyboard
               organization is line sequential.
           select sort-out
               assign display
               organization is line sequential.
           select sort-work
               assign "sortwork".
```

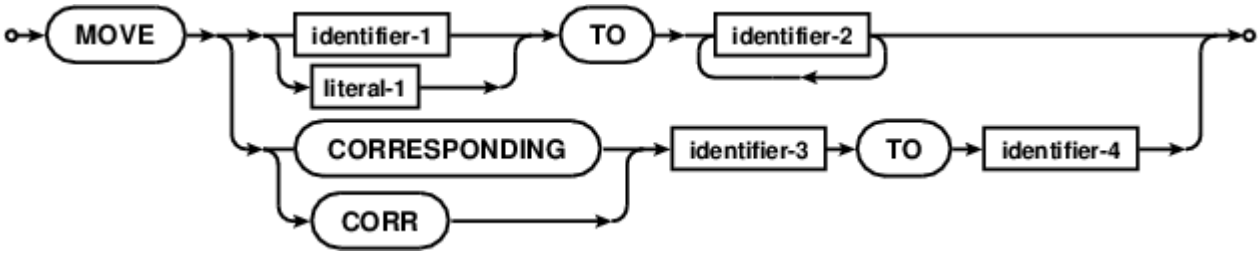
```
       data division.
       file section.
       fd sort-in.
          01 in-rec        pic x(255).
       fd sort-out.
          01 out-rec       pic x(255).
       sd sort-work.
          01 work-rec      pic x(255).

       working-storage section.
       01 loop-flag        pic x value low-value.

       procedure division.
       sort sort-work
           on descending key work-rec
           collating sequence is mixed
           input procedure is sort-transform
           output procedure is output-uppercase.

       display sort-return.
       goback.

      ******************************************************************
       sort-transform.
       move low-value to loop-flag
       open input sort-in
       read sort-in
           at end move high-value to loop-flag
       end-read
       perform
           until loop-flag = high-value
               move in-rec to work-rec
               RELEASE work-rec
               read sort-in
                   at end move high-value to loop-flag
               end-read
       end-perform
       close sort-in
       .

      ******************************************************************
       output-uppercase.
       move low-value to loop-flag
       open output sort-out
       return sort-work
           at end move high-value to loop-flag
       end-return
       perform
           until loop-flag = high-value
               move work-rec to out-rec
               write out-rec end-write
               return sort-work
                   at end move high-value to loop-flag
               end-return
       end-perform
       close sort-out
       .

       exit program.
       end program sorting.
```

Lea Declaración de LIBERACIÓN en línea: https://riptutorial.com/es/cobol/topic/7337/declaracion-de-liberacion

Capítulo 19: Declaración de MOVE

Observaciones

`MOVE` es el caballo de batalla de COBOL. Los datos se mueven de un literal o identificador a uno o más identificadores. COBOL tiene una distinción entre MOVE *elemental* y *grupal* . Los datos elementales son tipos convertidos de origen a destino. Los datos de grupo se mueven como una matriz de bytes, sin tener en cuenta los tipos de campo con una estructura. Los campos numéricos se mueven de derecha a izquierda, truncamiento de dígitos de orden superior con relleno cero (normalmente). Los datos de caracteres alfanuméricos se mueven de izquierda a derecha. El truncamiento de los caracteres del extremo derecho con el relleno de espacio. Existen bastantes reglas sobre cómo `MOVE` se `MOVE` de su negocio, con los formularios de datos BINARY y PICTURE DISPLAY, y se tienen en cuenta las jerarquías de grupo.



Examples

Algunos detalles de MOVE, hay muchos

```
01 a PIC 9.
01 b PIC 99.
01 c PIC 999.

01 s PIC X(4).

01 record-group.
   05 field-a PIC 9.
   05 field-b PIC 99.
   05 field-c PIC 999.
01 display-record.
   05 field-a PIC Z.
   05 field-b PIC ZZ.
   05 field-c PIC $Z9.

*> numeric fields are moved left to right
*> a set to 3, b set to 23, c set to 123
MOVE 123 TO a b c

*> moves can also be by matching names within groups
MOVE a TO field-a OF record-group
MOVE b TO field-b OF record-group
MOVE c TO field-c OF record-group
MOVE CORRESPONDING record-group TO display-record
```

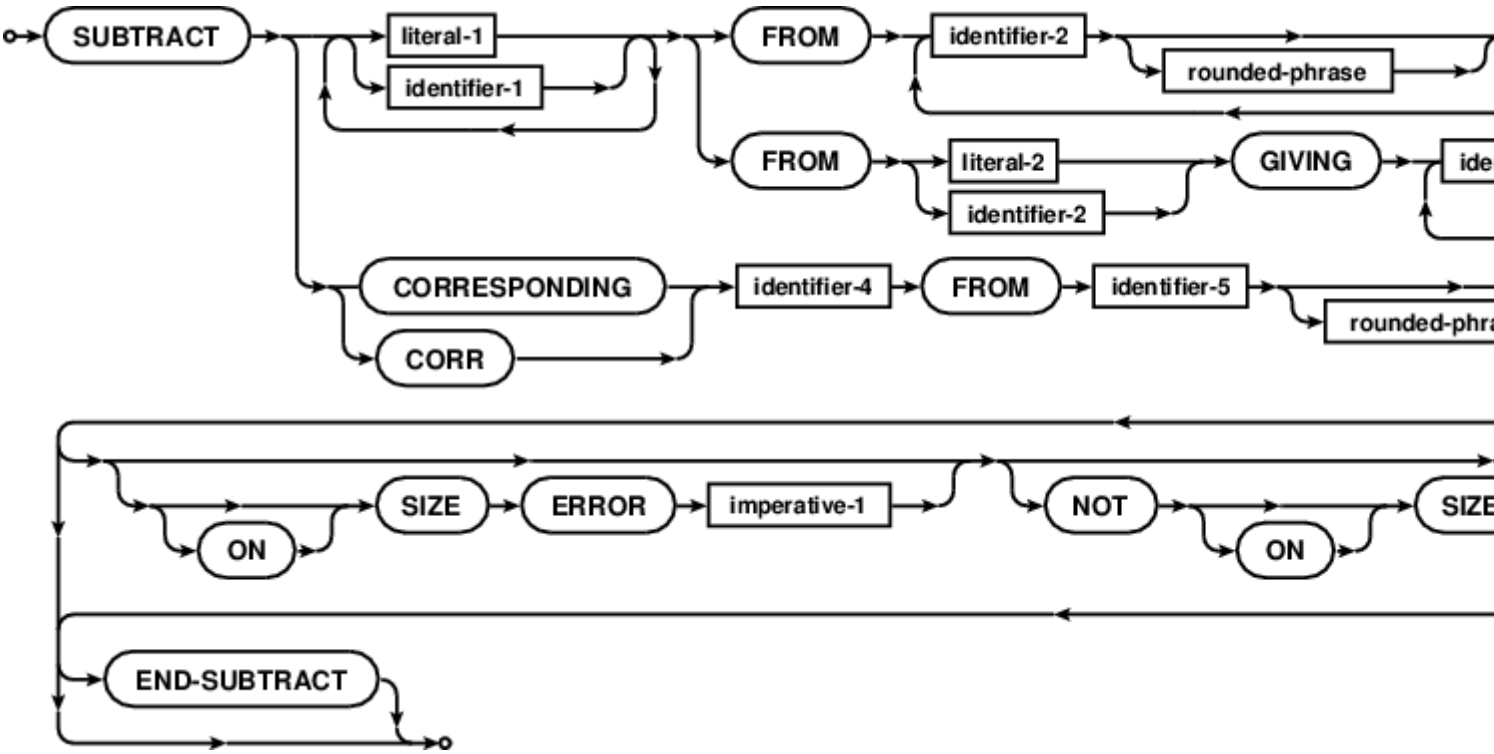
```
*> character data is moved right to left
*> s will be set to xyzz
MOVE "xyzzy" TO s
```

Lea Declaración de MOVE en línea: https://riptutorial.com/es/cobol/topic/7263/declaracion-de-move

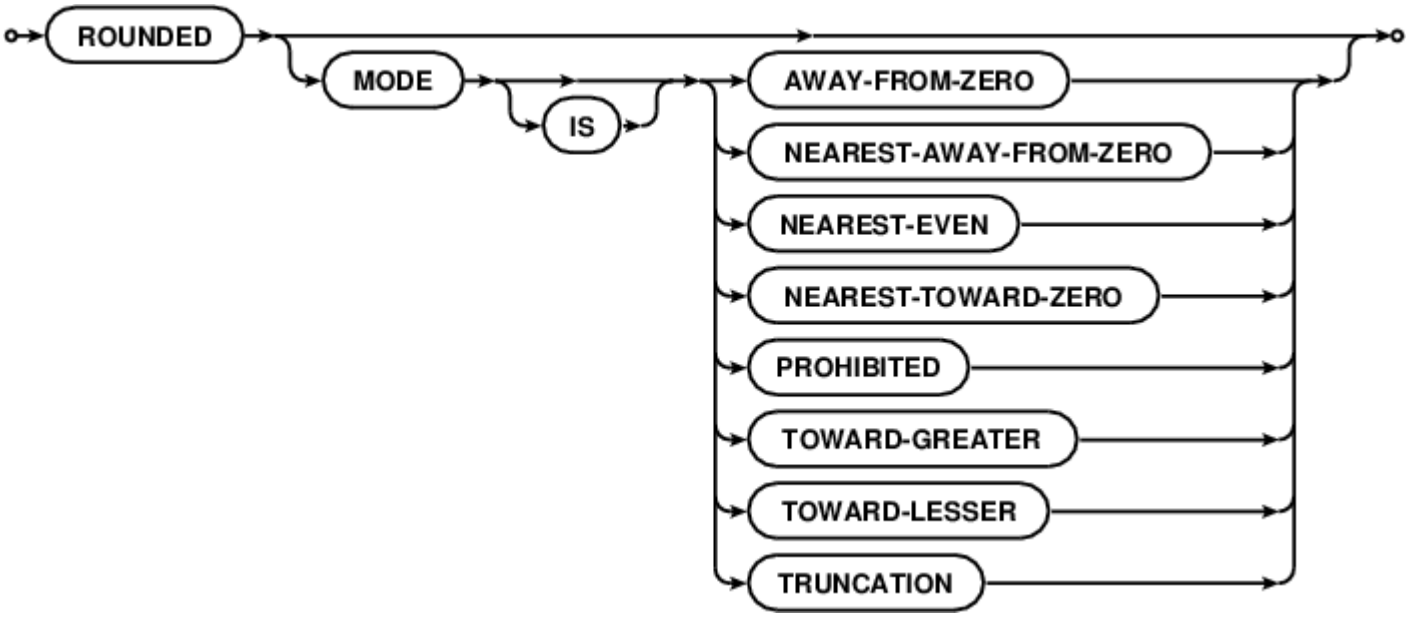
Capítulo 20: Declaración de réplica

Observaciones

La instrucción `SUBTRACT` se usa para restar uno o la suma de dos o más elementos de datos numéricos de uno o más elementos, y establecer los valores de uno o más identificadores al resultado.



frase redondeada



Examples

Ejemplo de restar

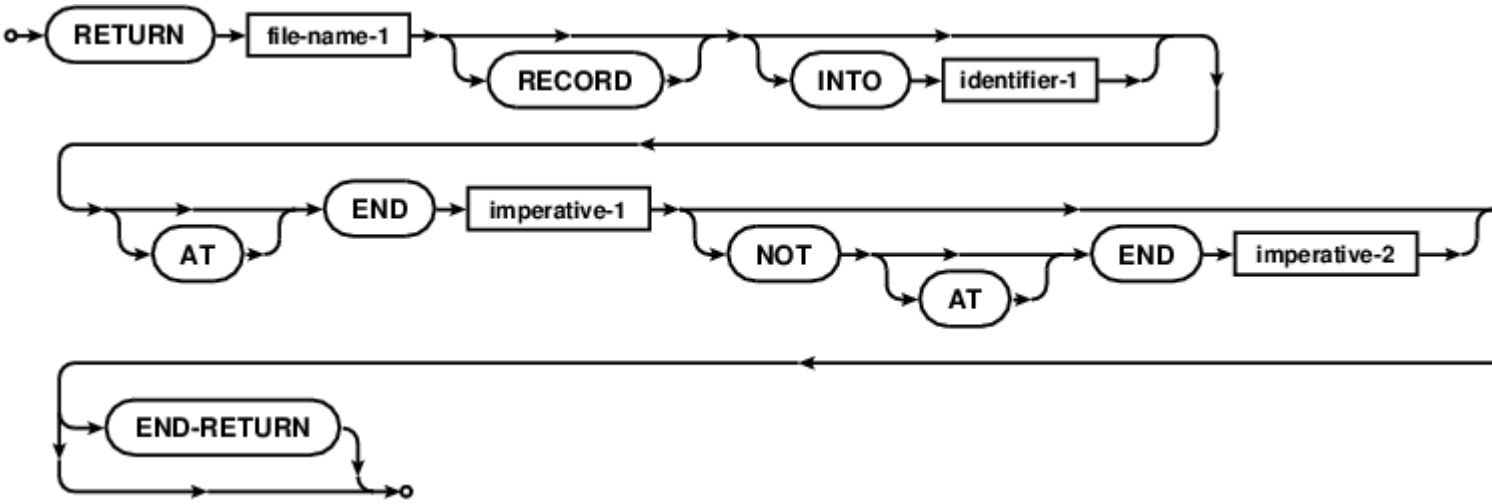
```
SUBTRACT item-a item-b item-c FROM account-z ROUNDED MODE IS NEAREST-EVEN
    ON SIZE ERROR
        DISPLAY "CALL THE BOSS, Account `Z` is OUT OF MONEY" END-DISPLAY
        PERFORM promisary-processing
    NOT ON SIZE ERROR
        PERFORM normal-processing
END-SUBTRACT
```

Lea Declaración de réplica en línea: https://riptutorial.com/es/cobol/topic/7465/declaracion-de-replica

Capítulo 21: Declaración de retorno

Observaciones

La instrucción `RETURN` controla cuándo se envían los datos al escritor interno del algoritmo de clasificación COBOL, como parte de un `OUTPUT PROCEDURE` . Los datos posteriores a la clasificación se pueden transformar bajo el control del programador antes de ser devueltos y escritos por el algoritmo de clasificación en el archivo de salida.



Examples

REGRESAR un registro para ordenar el procedimiento de salida

Esta es una muestra de semillas. El `SORT OUTPUT PROCEDURE` podía manipular los registros ordenados antes de que se devuelven a la parte de escritura del algoritmo de clasificación interna COBOL. En este caso, no se realiza ninguna transformación, `work-rec` se mueve directamente a `out-rec` .

```
GCobol >>SOURCE FORMAT IS FIXED
      ************************************************************
      * Purpose:   A GnuCOBOL SORT verb example
      * Tectonics: cobc -x sorting.cob
      *     ./sorting <input >output
      *   or simply
      *     ./sorting
      *   for keyboard and screen demos
      ************************************************************
       identification division.
       program-id. sorting.

       environment division.
       configuration section.
      * Set up a sort order where lower and upper case stay together
       special-names.
           alphabet mixed is " aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTu
      -"UvVwWxXyYzZ0123456789".
```

```
       input-output section.
       file-control.
           select sort-in
               assign keyboard
               organization is line sequential.
           select sort-out
               assign display
               organization is line sequential.
           select sort-work
               assign "sortwork".

       data division.
       file section.
       fd sort-in.
          01 in-rec        pic x(255).
       fd sort-out.
          01 out-rec       pic x(255).
       sd sort-work.
          01 work-rec      pic x(255).

       working-storage section.
       01 loop-flag        pic x value low-value.

       procedure division.
       sort sort-work
           on descending key work-rec
           collating sequence is mixed
           input procedure is sort-reader
           output procedure is sort-writer.

       display sort-return.
       goback.

      ******************************************************************
       sort-reader.
       move low-value to loop-flag
       open input sort-in
       read sort-in
           at end move high-value to loop-flag
       end-read
       perform
           until loop-flag = high-value
               move in-rec to work-rec
               release work-rec
               read sort-in
                   at end move high-value to loop-flag
               end-read
       end-perform
       close sort-in
       .

      ******************************************************************
       sort-writer.
       move low-value to loop-flag
       open output sort-out
       return sort-work
           at end move high-value to loop-flag
       end-return
       perform
           until loop-flag = high-value
               move work-rec to out-rec
```

```
            write out-rec end-write
            RETURN sort-work
                at end move high-value to loop-flag
            end-return
    end-perform
    close sort-out
    .

    exit program.
    end program sorting.
```

Lea Declaración de retorno en línea: https://riptutorial.com/es/cobol/topic/7338/declaracion-de-retorno

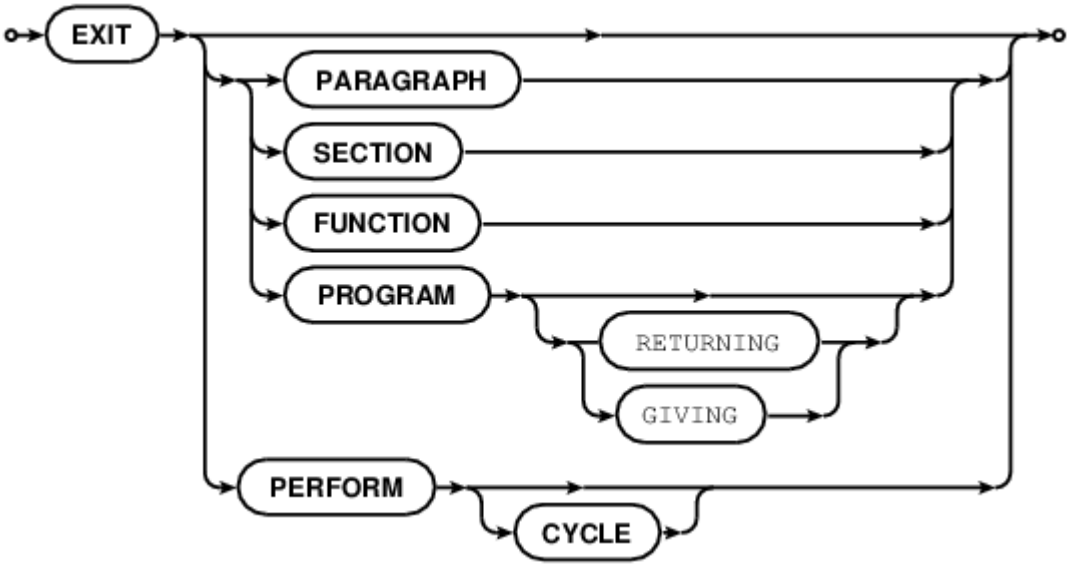
Capítulo 22: Declaración de salida

Observaciones

La instrucción COBOL `EXIT` es un verbo de control de flujo de terminación.

`EXIT` viene es algunos sabores:

- bare `EXIT` es un punto final común para una serie de procedimientos.
- `EXIT PARAGRAPH` `EXIT SECTION` , `EXIT SECTION` proporciona un medio para salir de un procedimiento estructurado sin ejecutar ninguna de las declaraciones posteriores.
- `EXIT FUNCTION` `EXIT METHOD` , `EXIT METHOD` `EXIT PROGRAM` , `EXIT PROGRAM` marca el final lógico de un módulo de código.
- `EXIT PERFORM` sale de un bucle de ejecución en línea.
- `EXIT PERFORM CYCLE` hace que un bucle de ejecución en línea comience la siguiente iteración.



Examples

Declaración de salida

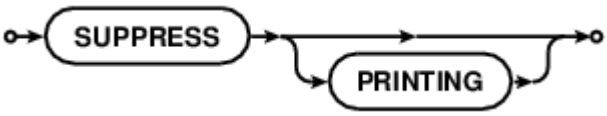
```
PERFORM VARYING counter FROM 1 BY 1 UNTIL counter > 10
    IF debug-override THEN EXIT PERFORM
    IF counter = 5 THEN EXIT PERFORM CYCLE
    PERFORM some-miracle
END-PERFORM
```

Lea Declaración de salida en línea: https://riptutorial.com/es/cobol/topic/7084/declaracion-de-salida

Capítulo 23: Declaración de supresión

Observaciones

La instrucción `SUPPRESS` inhibe la impresión de un grupo de informes. Función COBOL Report Writer.



Examples

Ejemplo de supresión

```
SUPPRESS PRINTING
```

Lea Declaración de supresión en línea: https://riptutorial.com/es/cobol/topic/7470/declaracion-de-supresion

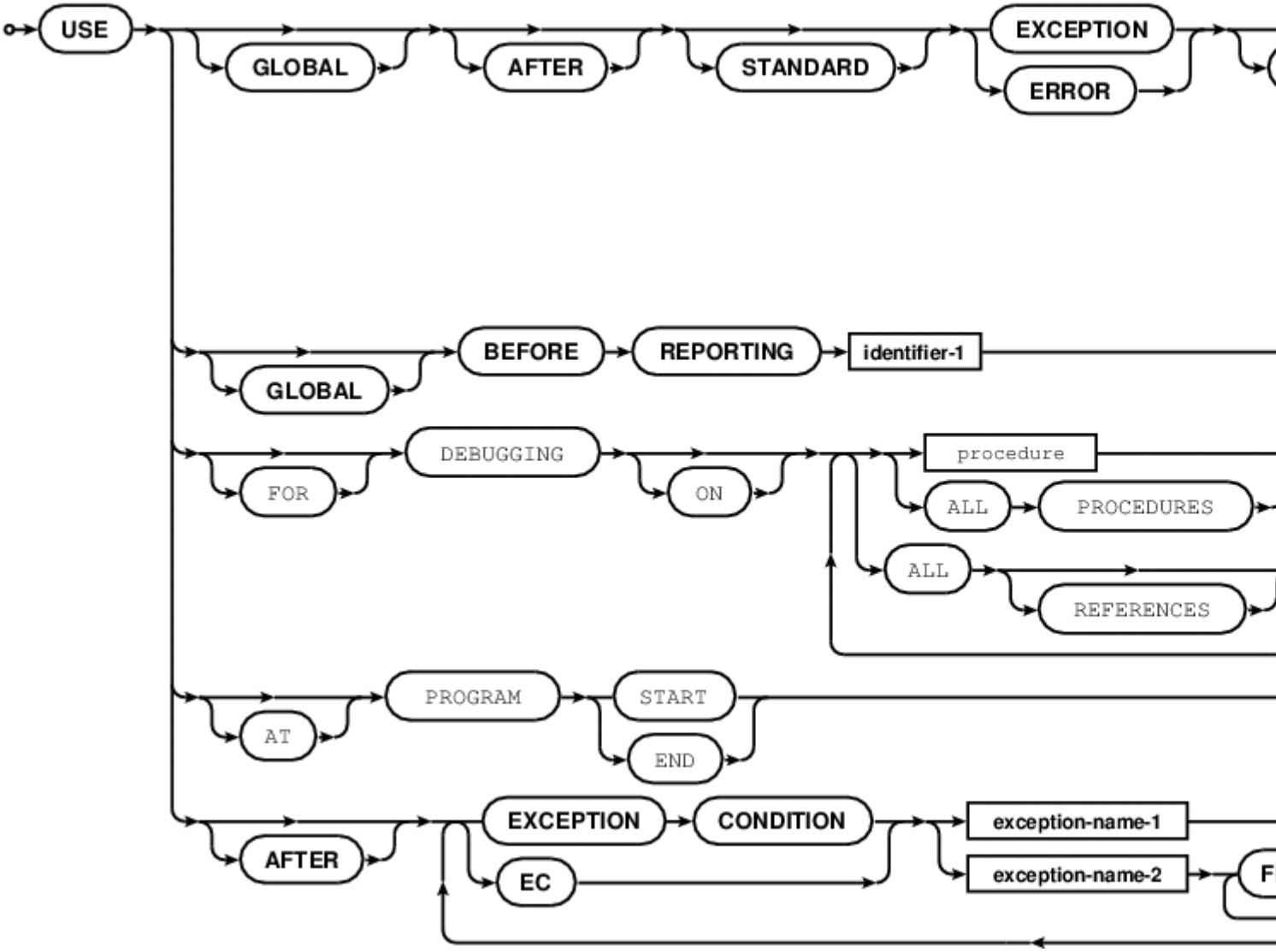
Capítulo 24: Declaración DE USO

Observaciones

La declaración `USE` especifica los procedimientos a utilizar.

- para el manejo de errores y excepciones además de los proporcionados por otras instalaciones
- antes de imprimir un grupo de informes designado
- Después de la detección de condiciones de excepción designadas

El uso obsoleto incluye la especificación de procedimientos que se utilizarán durante la `DEBUGGING` , y las extensiones incluyen la adición de procedimientos intersticiales para el inicio y el final del programa.



Examples

Declaración de uso con el escritor del informe

```
035700 PROCEDURE DIVISION.
035800
035900 DECLARATIVES.
036000
036100 DEPT-HEAD-USE SECTION. USE BEFORE REPORTING DEPT-HEAD.
036200 DEPT-HEAD-PROC.
036300     SET DE-IX TO +1.
036400     SEARCH DEPARTMENT-ENTRY
036500         WHEN DE-NUMBER (DE-IX) = PRR-DEPARTMENT-NUMBER
036600             MOVE ZEROS TO DE-GROSS (DE-IX), DE-FICA (DE-IX),
036700                           DE-FWT (DE-IX), DE-MISC (DE-IX),
036800                           DE-NET (DE-IX).
036900
037000 DEPT-HEAD-EXIT.
037100     EXIT.
037200
037300 EMPL-FOOT-USE SECTION. USE BEFORE REPORTING EMPL-FOOT.
037400 EMPL-FOOT-PROC.
037500     MOVE PRR-EMPLOYEE-KEY TO WS-EMPLOYEE-KEY.
037600
037700 EMPL-FOOT-EXIT.
037800     EXIT.
037900
038000 DEPT-FOOT-USE SECTION. USE BEFORE REPORTING DEPT-FOOT.
038100 DEPT-FOOT-PROC.
038200     MOVE DEPT-FOOT-GROSS TO DE-GROSS (DE-IX).
038300     MOVE DEPT-FOOT-FICA TO DE-FICA (DE-IX).
038400     MOVE DEPT-FOOT-FWT TO DE-FWT (DE-IX).
038500     MOVE DEPT-FOOT-MISC TO DE-MISC (DE-IX).
038600     MOVE DEPT-FOOT-NET TO DE-NET (DE-IX).
      *     SUPPRESS PRINTING.
038700
038800 DEPT-FOOT-EXIT.
038900     EXIT.
039000
039100 COMP-FOOT-USE SECTION. USE BEFORE REPORTING COMP-FOOT.
039200 COMP-FOOT-PROC.
039300     PERFORM COMP-FOOT-CALC
039400         VARYING WPCD-IX FROM +1 BY +1
039500         UNTIL WPCD-IX > +6.
039600     GO TO COMP-FOOT-EXIT.
039700
039800 COMP-FOOT-CALC.
039900     SET DE-IX TO WPCD-IX.
040000     SET WPCC-IX TO +1.
040100     COMPUTE WPC-PERCENT (WPCD-IX WPCC-IX) ROUNDED =
040200         ((DE-GROSS (DE-IX) / CO-GROSS) * 100) + .5.
040300     SET WPCC-IX TO +2.
040400     COMPUTE WPC-PERCENT (WPCD-IX WPCC-IX) ROUNDED =
040500         ((DE-FICA (DE-IX) / CO-FICA) * 100) + .5.
040600     SET WPCC-IX TO +3.
040700     COMPUTE WPC-PERCENT (WPCD-IX WPCC-IX) ROUNDED =
040800         ((DE-FWT (DE-IX) / CO-FWT) * 100) + .5.
040900     SET WPCC-IX TO +4.
041000     COMPUTE WPC-PERCENT (WPCD-IX WPCC-IX) ROUNDED =
041100         ((DE-MISC (DE-IX) / CO-MISC) * 100) + .5.
041200     SET WPCC-IX TO +5.
041300     COMPUTE WPC-PERCENT (WPCD-IX WPCC-IX) ROUNDED =
041400         ((DE-NET (DE-IX) / CO-NET) * 100) + .5.
041500
041600 COMP-FOOT-EXIT.
```

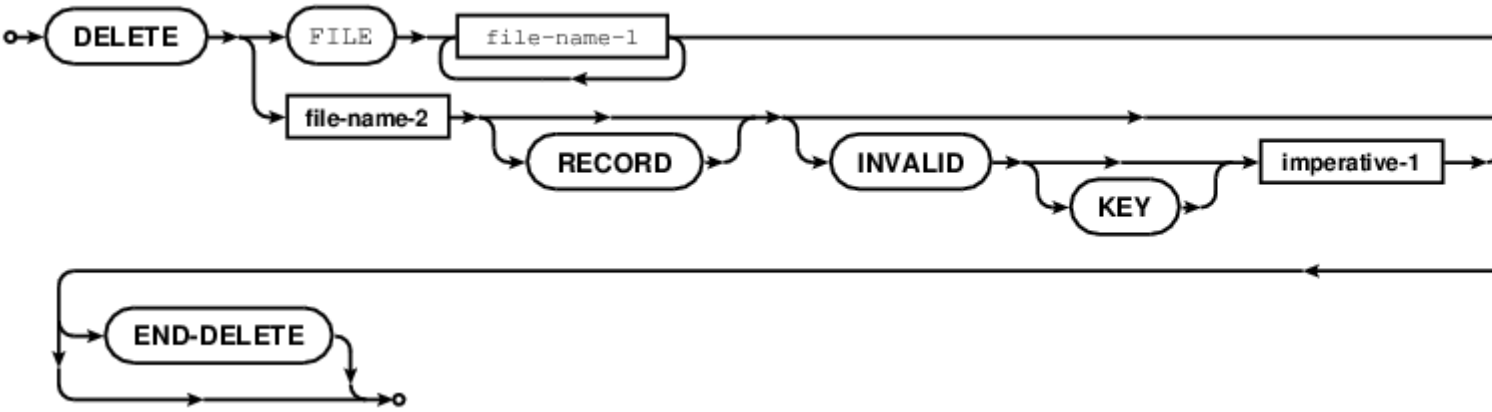
```
041700     EXIT.
041800
041900 END DECLARATIVES.
```

Lea Declaración DE USO en línea: https://riptutorial.com/es/cobol/topic/7582/declaracion-de-uso

Capítulo 25: Declaración DELETE

Observaciones

La instrucción `DELETE` elimina registros del almacenamiento masivo. Algunos compiladores permiten que la instrucción `DELETE` se use con una cláusula `FILE` , para eliminar nombres `FD` (junto con cualquier estructura de indexación asociada que pueda ser requerida por el motor de administración de la base de datos en uso).



Examples

Eliminar un registro, clave en el campo de clave principal

```
identification division.
program-id. deleting.

environment division.
configuration section.

input-output section.
file-control.
    select optional indexed-file
    assign to "indexed-file.dat"
    status is indexing-status
    organization is indexed
    access mode is dynamic
    record key is keyfield
    alternate record key is altkey with duplicates
    .

...

procedure division.

move "abcdef" to keyfield

*> Delete a record by index
delete indexed-file record
   invalid key
       display "No delete of " keyfield end-display
```

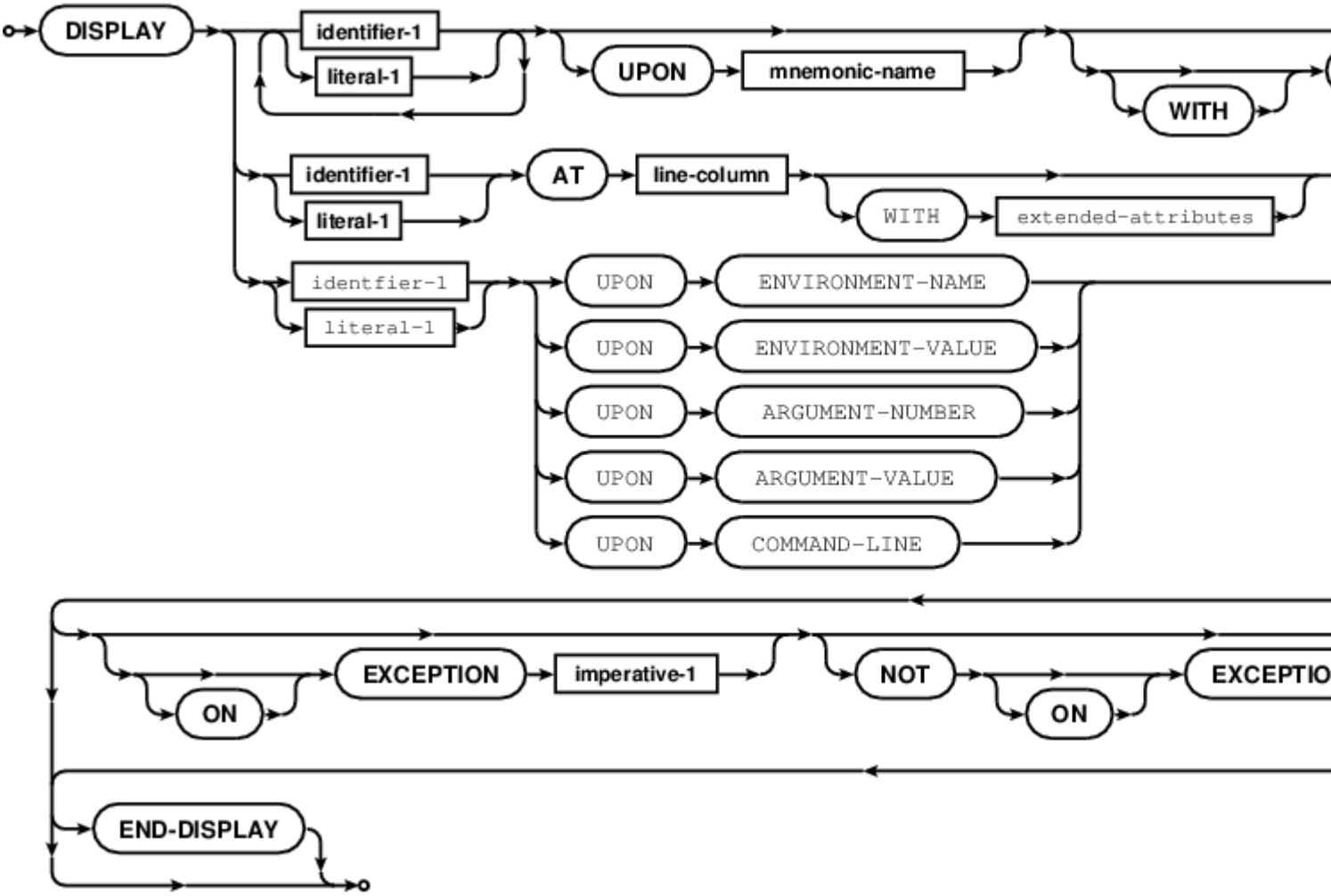
```
   not invalid key
       display "Record " keyfield " removed" end-display
end-delete

perform check-delete-status

...
```

Lea Declaración DELETE en línea: https://riptutorial.com/es/cobol/topic/7063/declaracion-delete

Capítulo 26: Declaración DISPLAY

Observaciones

La instrucción `DISPLAY` hace que los datos se transfieran al hardware o software del entorno operativo. `DISPLAY` presenta en dos formas, `UPON device` o para mostrar los datos de la `SCREEN` Las variables de entorno también se pueden configurar con `DISPLAY UPON` en algunas implementaciones de COBOL, junto con otras extensiones para la transferencia de datos de gráficos u otras necesidades específicas del dispositivo.



Examples

Mostrar en

```
DISPLAY "An error occurred with " tracked-resource UPON SYSERR

DISPLAY A, B, C UPON CONSOLE

DISPLAY group-data UPON user-device
    ON EXCEPTION
        WRITE device-exception-notice
```

```
    NOT ON EXCEPTION
        WRITE device-usage-log
END-DISPLAY
```

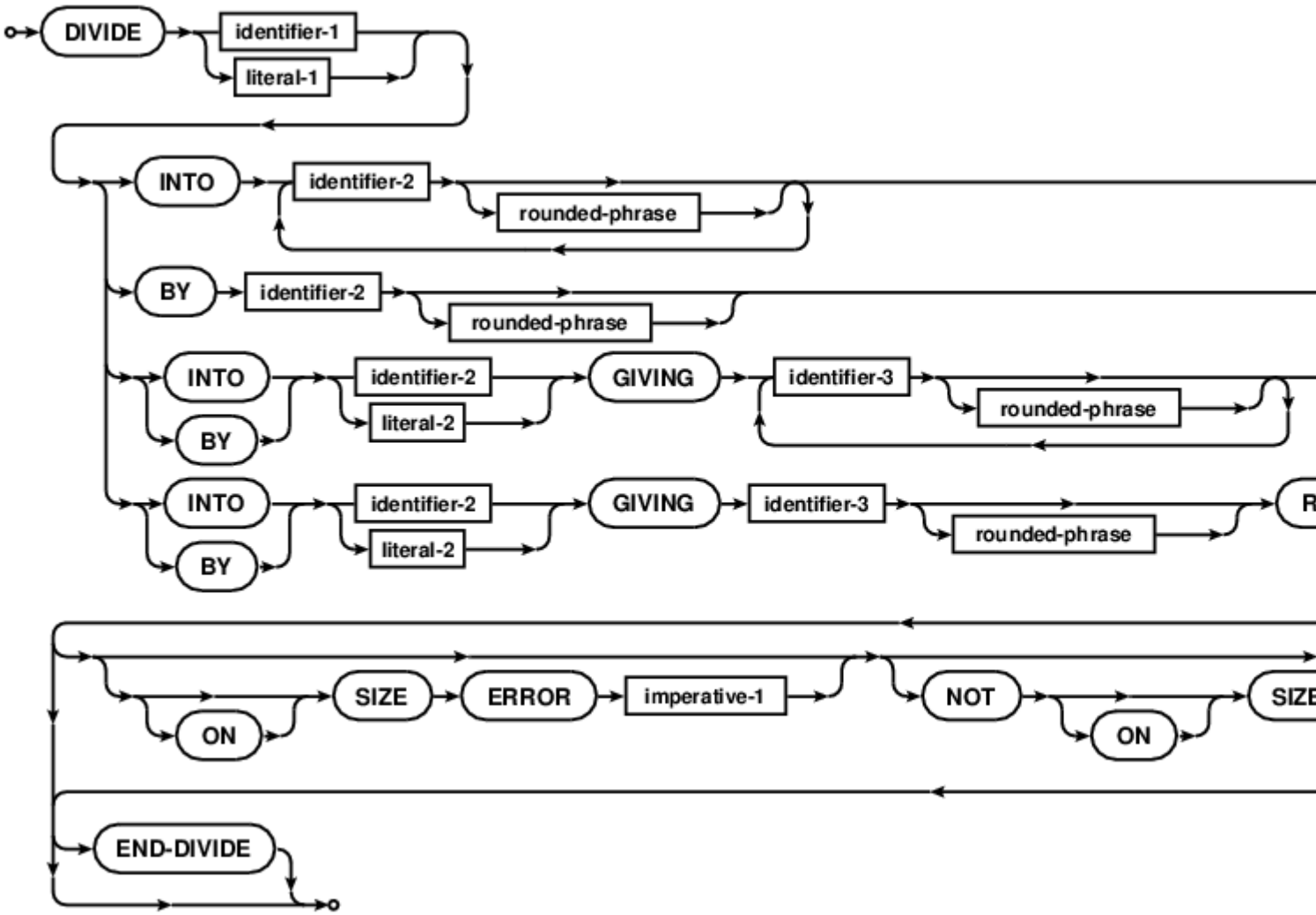
UPON CONSOLE es un valor predeterminado, raramente escrito. Los mensajes con DISPLAY son una forma de depurar el código COBOL, pero muchos programas COBOL son de naturaleza transaccional y pueden no interactuar con un operador humano una vez que se envía un trabajo.

Lea Declaración DISPLAY en línea: https://riptutorial.com/es/cobol/topic/7082/declaracion-display

Capítulo 27: Declaración divisoria

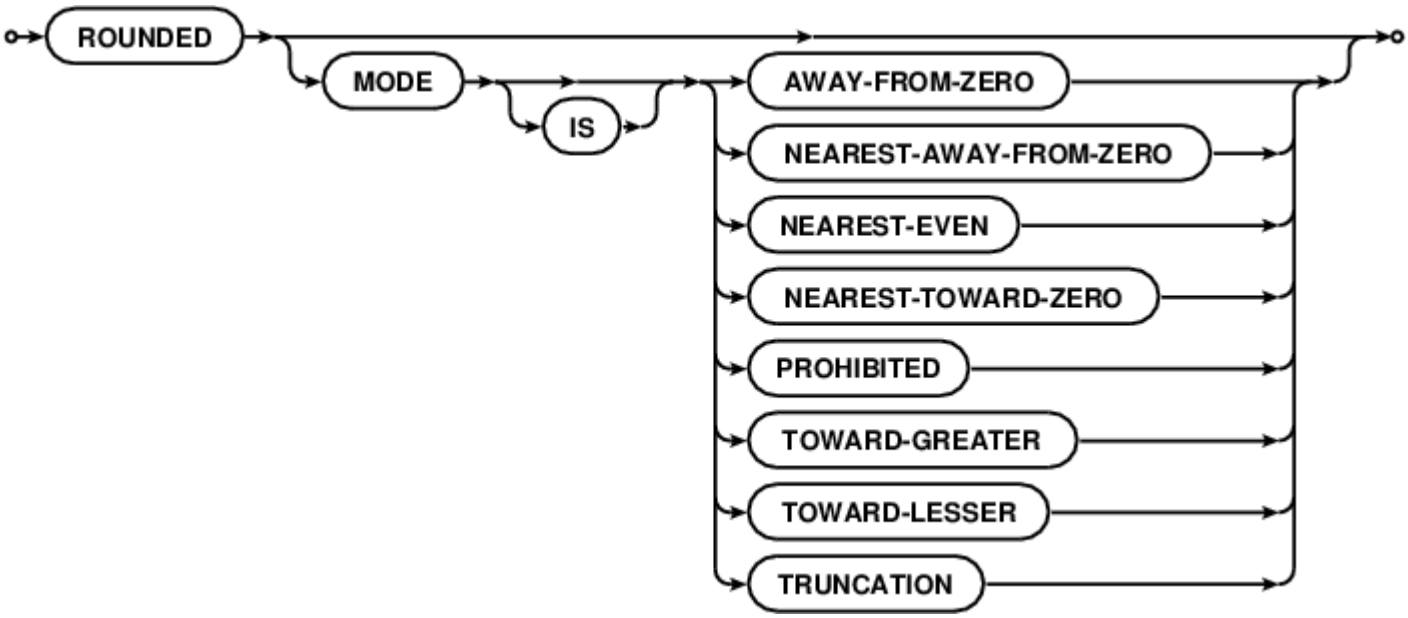
Observaciones

La instrucción COBOL `DIVIDE` divide un elemento numérico en otros, estableciendo elementos de datos en el cociente y, opcionalmente, el resto.



Frase `ROUNDED` :

El valor predeterminado es `TRUNCATION` si no se especifica una frase redondeada. El modo `ROUNDED` predeterminado es `ROUNDED NEAREST-TOWARD-ZERO` (redondeo hacia abajo) a menos que se especifique lo contrario. El llamado *redondeo del banquero* es `NEAREST-EVEN` .



Examples

DIVIDE formatos de instrucciones

```
DIVIDE a INTO b c d
```

Los elementos de datos `b` , `c` y `d` se cambian como `b/a` , `c/a` `d/a` .

```
DIVIDE a INTO b GIVING c
```

El elemento de datos `c` se cambia como `b/a` .

```
DIVIDE a BY b GIVING c
```

El elemento de datos `c` se cambia como `a/b` .

```
DIVIDE a INTO b GIVING q REMAINDER r
```

Los elementos de datos `q` y `r` se establecen con los resultados de `b/a`

```
DIVIDE a BY b GIVING q REMAINDER r
```

Los elementos de datos `q` y `r` se establecen con los resultados de `b/a`

Todos `DIVIDE` campos de resultados pueden tener `ROUNDED MODE IS` cláusulas.

Todas las declaraciones `DIVIDE` pueden incluir declaraciones declarativas `ON SIZE ERROR` y `NOT ON SIZE ERROR` para detectar resultados inválidos dado el tipo y tamaño de los campos de resultados.

Lea Declaración divisoria en línea: https://riptutorial.com/es/cobol/topic/7081/declaracion-divisoria

Capítulo 28: Declaración GENERATE

Observaciones

La instrucción COBOL `GENERATE` es una declaración opcional que se admite si el compilador incluye la función Report Writer.

Examples

GENERAR una línea de detalle

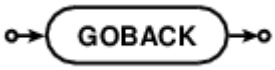
```
GENERATE detail-line
```

Lea Declaración GENERATE en línea: https://riptutorial.com/es/cobol/topic/7161/declaracion-generate

Capítulo 29: Declaración GOBACK

Observaciones

La sentencia COBOL `GOBACK` es una devolución. A diferencia de `EXIT PROGRAM` , o `STOP RUN` , `GOBACK` siempre devuelve un nivel. Si el módulo actual es "principal", `GOBACK` volverá al sistema operativo. Si el módulo actual es un subprograma, `GOBACK` volverá a la instrucción después de una llamada.



Examples

REGRESA

```
identification division.
program-id. subprog.
procedure division.
display "in subprog"
goback.

...

call "subprog"
goback.
```

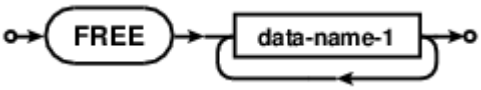
El primer `GOBACK` anterior volverá de subprog. Suponiendo que el segundo se encuentre dentro del procedimiento principal, `GOBACK` volverá al sistema operativo.

Lea Declaración GOBACK en línea: https://riptutorial.com/es/cobol/topic/7173/declaracion-goback

Capítulo 30: Declaración GRATIS

Observaciones

La instrucción `FREE` libera la memoria asignada para uno o más identificadores, ya sea por POINTER o desde un identificador de almacenamiento de trabajo BASADO. Usar después de LIBRE es ilegal.



Examples

GRATIS una asignación

```
01 field-1 PIC X(80) BASED.

ALLOCATE field-1

*> use field-1

FREE field-1

*> further use of field-1 will cause memory corruption
```

Lea Declaración GRATIS en línea: https://riptutorial.com/es/cobol/topic/7162/declaracion-gratis

Capítulo 31: Declaración IF

Observaciones

La expresión condicional y la declaración de selección. Se recomienda el uso de terminadores de alcance explícito. Las expresiones condicionales COBOL permiten formas cortas, donde el identificador actual (y condicional) se asume a través de múltiples pruebas de condición, a menos que se proporcionen explícitamente.

```
IF A = 1 OR 2 ...
```

es equivalente a

```
IF A = 1 OR A = 2 ...
```

Examples

IF con condicionales de forma corta

```
IF A = 1 OR 2 THEN
    perform miracles
END-IF

IF A = 1 OR 2 AND B = 1 THEN
    perform rites-of-passage
ELSE
    perform song-and-dance
END-IF
```

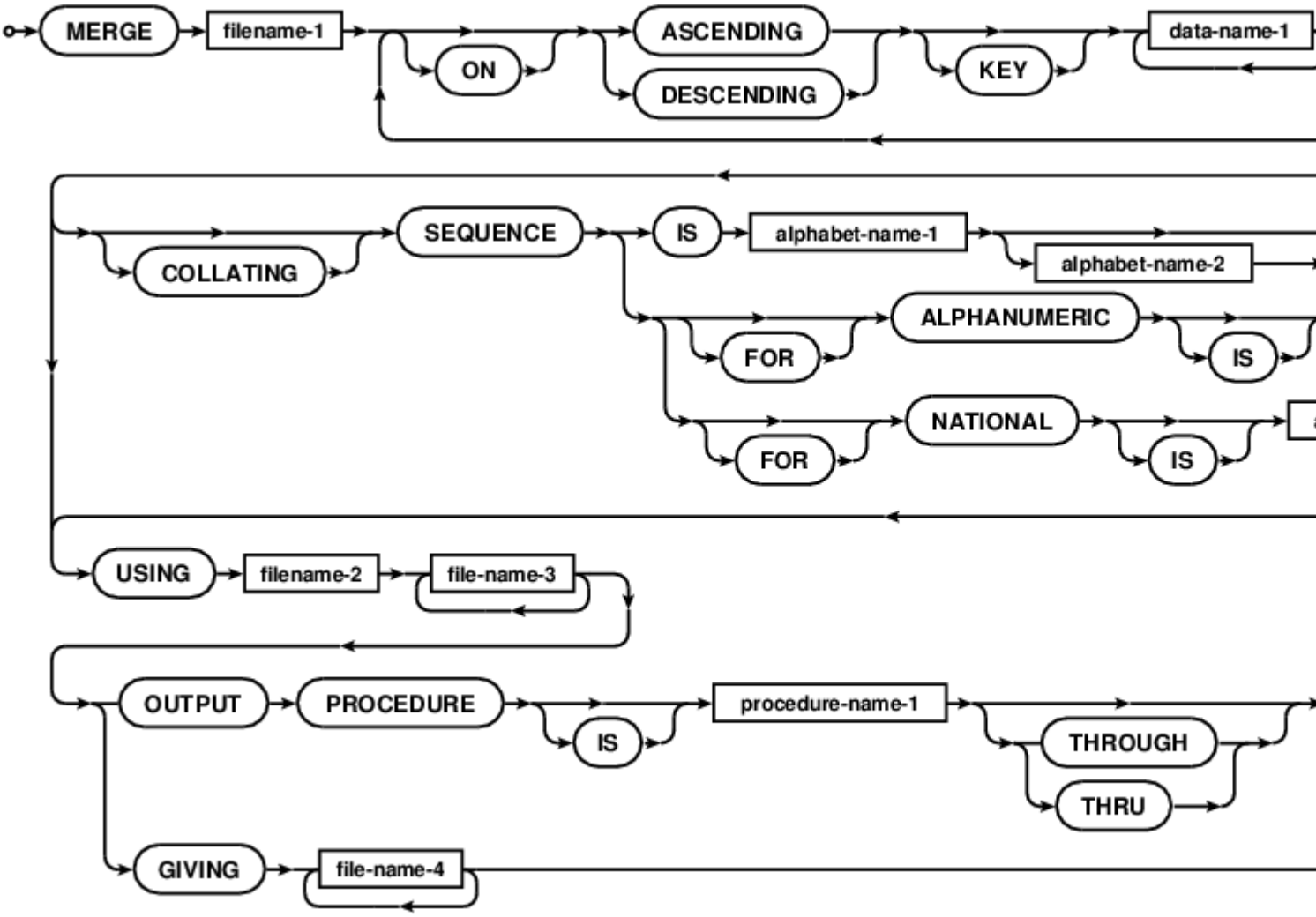
`IF` sentencias `IF` se pueden terminar con una terminación completa o un terminador de alcance explícito `END-IF` . *Ya no se recomienda el uso de períodos para la terminación del alcance.* Las paradas completas solo significan que, en el caso de IF anidadas, todas las anidaciones terminan en la primera parada completa `.` , y cualquier código posterior estará fuera del bloque IF.

Lea Declaración IF en línea: https://riptutorial.com/es/cobol/topic/7174/declaracion-if

Capítulo 32: Declaración MERGE

Observaciones

La declaración MERGE fusionará uno o más archivos de datos COBOL con formato similar en un solo archivo de salida. El programador puede asumir el control sobre el `OUTPUT PROCEDURE` , que usa la instrucción `RELEASE` , o usar mecanismos internos de tiempo de ejecución COBOL con la cláusula `GIVING` .



Examples

MERGE datos regionales en maestro

```
GCobol >>SOURCE FORMAT IS FIXED
      *> ************************************************************
      *> Purpose:   Demonstrate a merge pass
      *> Tectonics: cobc -x gnucobol-merge-sample.cob
      *> ************************************************************
       identification division.
       program-id. gnucobol-merge-sample.
```

```
       environment division.
       configuration section.
       repository.
           function all intrinsic.

files  input-output section.
       file-control.
           select master-file
               assign to "master-sample.dat"
               organization is line sequential.

           select eastern-transaction-file
               assign to "east-transact-sample.dat"
               organization is line sequential.

           select western-transaction-file
               assign to "west-transact-sample.dat"
               organization is line sequential.

           select merged-transactions
               assign to "merged-transactions.dat"
               organization is line sequential.

           select working-merge
               assign to "merge.tmp".

data   data division.
       file section.
       fd master-file.
          01 master-record     pic x(64).

       fd eastern-transaction-file.
          01 transact-rec      pic x(64).

       fd western-transaction-file.
          01 transact-rec      pic x(64).

       fd merged-transactions.
          01 new-rec           pic x(64).

       sd working-merge.
          01 merge-rec.
             02 master-key     pic 9(8).
             02 filler         pic x.
             02 action         pic xxx.
             02 filler         PIC x(52).

      *> ***************************************************************
      *> not much code
      *>     trick.  DEP, CHQ, BAL are action keywords.  They sort
      *>     descending as DEP, CHQ, BAL, so main can do all deposits,
      *>     then all withdrawals, then balance reports, for each id.
      *> ***************************************************************
code   procedure division.
       merge working-merge
           on ascending key master-key
              descending key action
           using eastern-transaction-file,
                 western-transaction-file,
                 master-file
```

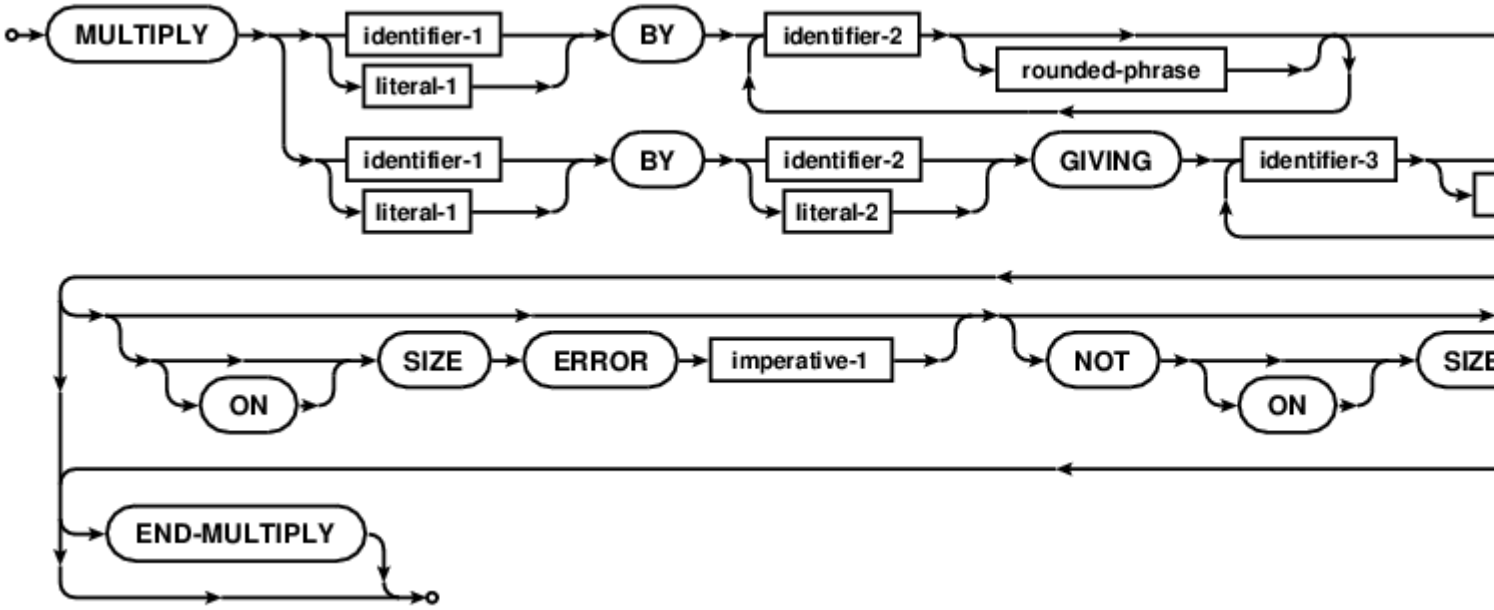
```
           giving merged-transactions
done   goback.
       end program gnucobol-merge-sample.
```

Lea Declaración MERGE en línea: https://riptutorial.com/es/cobol/topic/7183/declaracion-merge

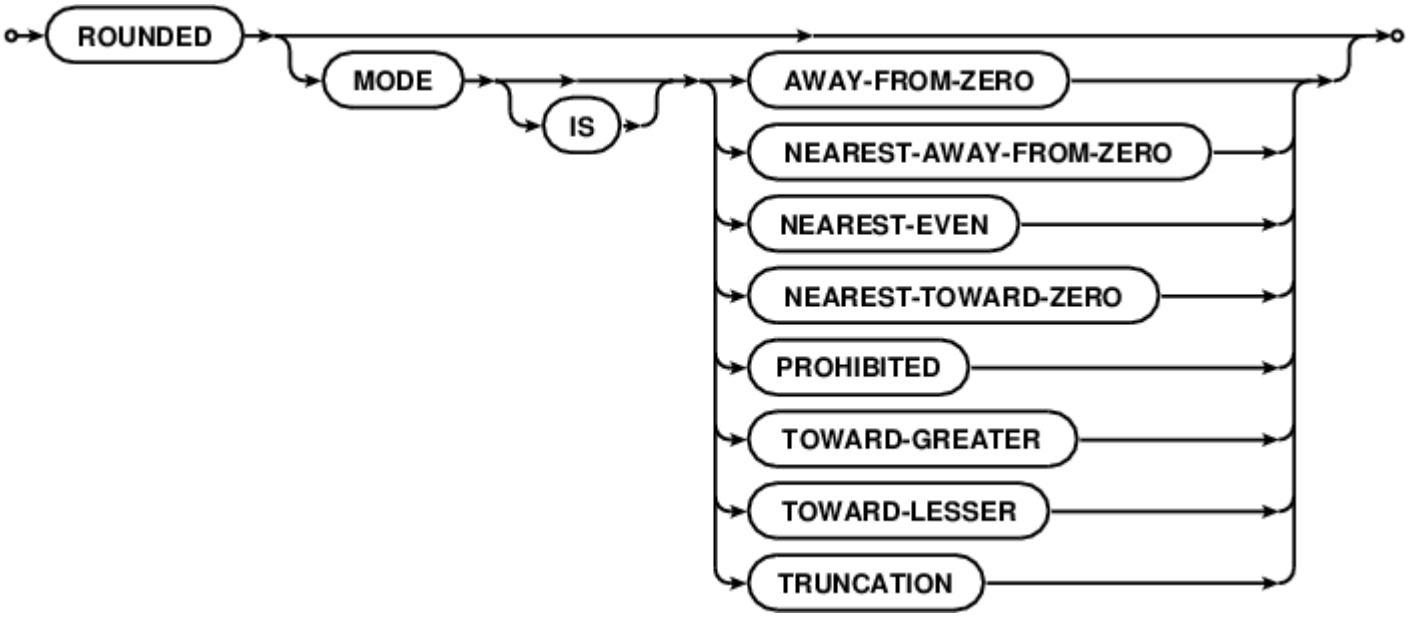
Capítulo 33: Declaración MULTIPLY

Observaciones

La instrucción `MULTIPLY` multiplica los datos numéricos configurando el resultado en uno o más identificadores de tipo numérico.



Donde `rounded-phrase` es



Examples

Algunos formatos MULTIPLICOS

```
MULTIPLY 5 BY a
```

```
MULTIPLY a BY b
    ON SIZE ERROR
        PERFORM error-handling
    NOT ON SIZE ERROR
        PERFORM who-does-that
END-MULTIPLY

MULTIPLY a BY b GIVING x ROUNDED MODE IS PROHIBITED
                       y ROUNDED MODE IS NEAREST-EVEN
                       z ROUNDED
```

Lea Declaración MULTIPLY en línea: https://riptutorial.com/es/cobol/topic/7264/declaracion-multiply

Capítulo 34: Declaración PERFORM

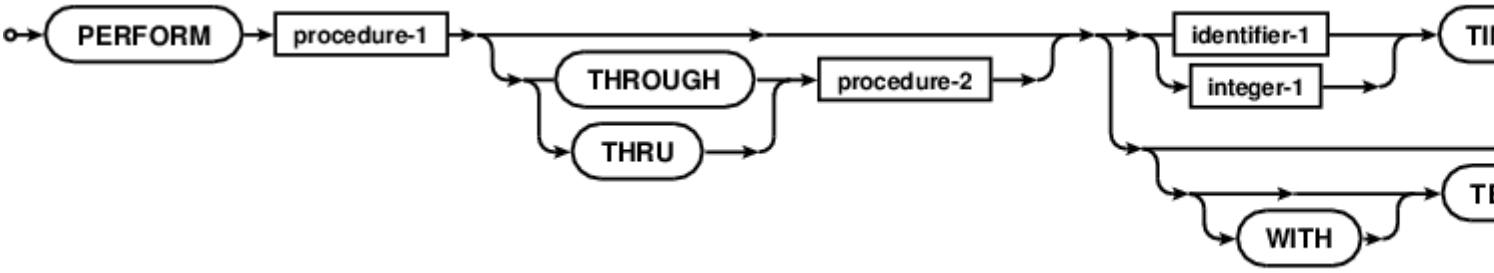
Observaciones

La instrucción PERFORM transfiere el control a uno o más procedimientos y devuelve el control implícitamente cuando se completa la secuencia. PERFORM también se puede utilizar para bucles en línea dentro del alcance de PERFORM.

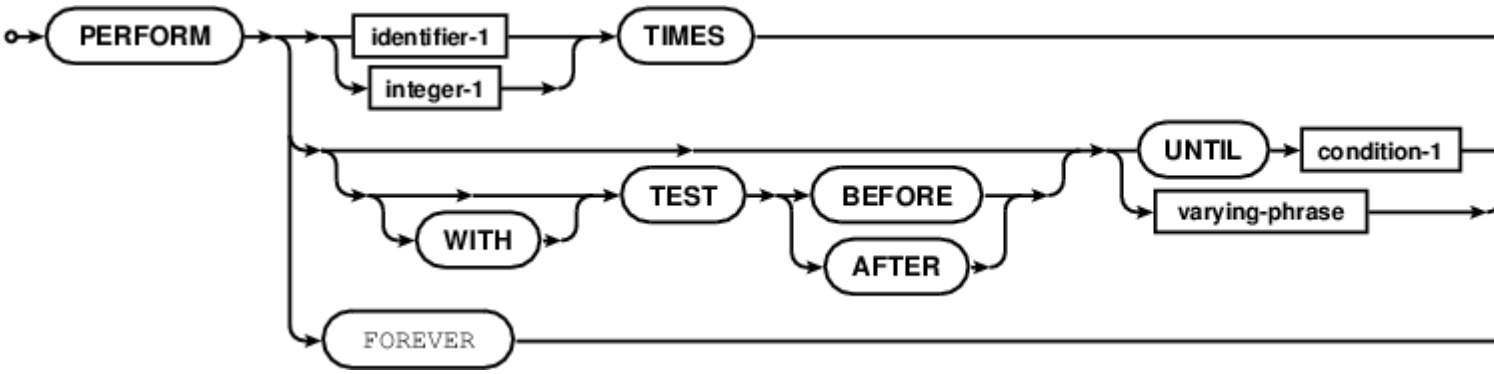
La frase `VARYING` permite anidar con una o más cláusulas `AFTER` , y la prueba condicional puede ser `BEFORE` (predeterminada) o `AFTER` cada bucle.

La cláusula `THRU` de una ejecución de procedimiento asume un flujo de control descendente secuencial desde el `procedure-1` hasta el final del `procedure-2` . *THRU* es un tema polémico, y muchos programadores prefieren `PERFORM SECTION` lugar de usar los párrafos `THRU` . Algunas tiendas pueden exigir `PERFORM THRU` con un párrafo de punto de salida explícito, otras pueden prohibir el uso de `THRU` lo que dificulta la depuración.

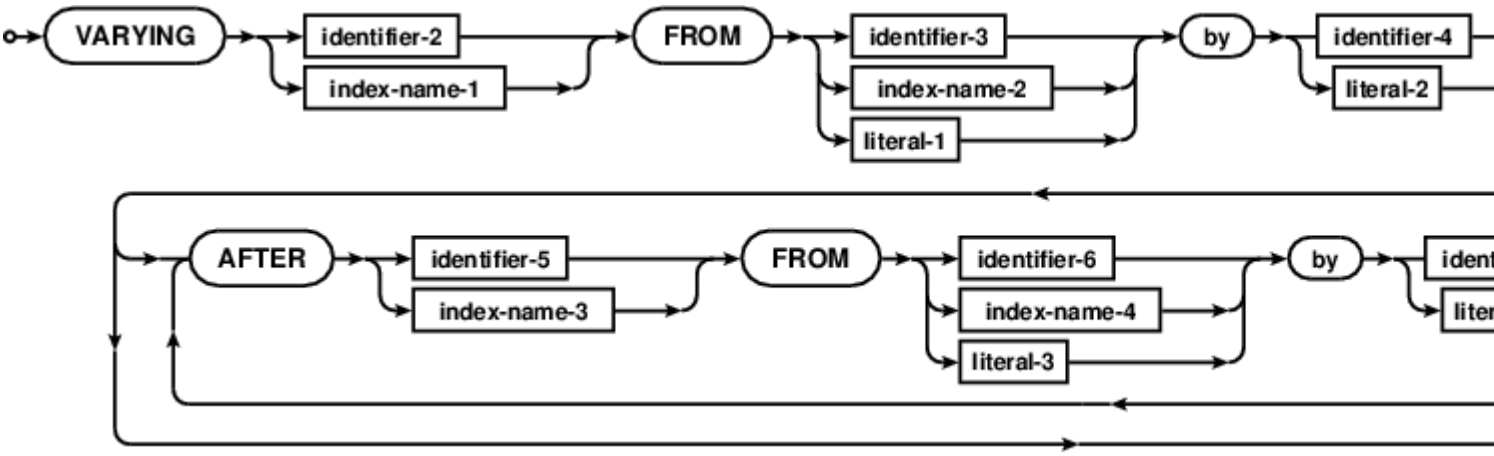
Realización procesal:



En línea realizar:



Donde `varying-phrase` es:



Examples

En línea realizar variacion

```
PERFORM VARYING TALLY FROM 1 BY 1 UNTIL TALLY > 5
    DISPLAY TALLY
END-PERFORM
```

PROCEDIMIENTO DE PROCEDIMIENTO

```
PERFORM some-paragraph
```

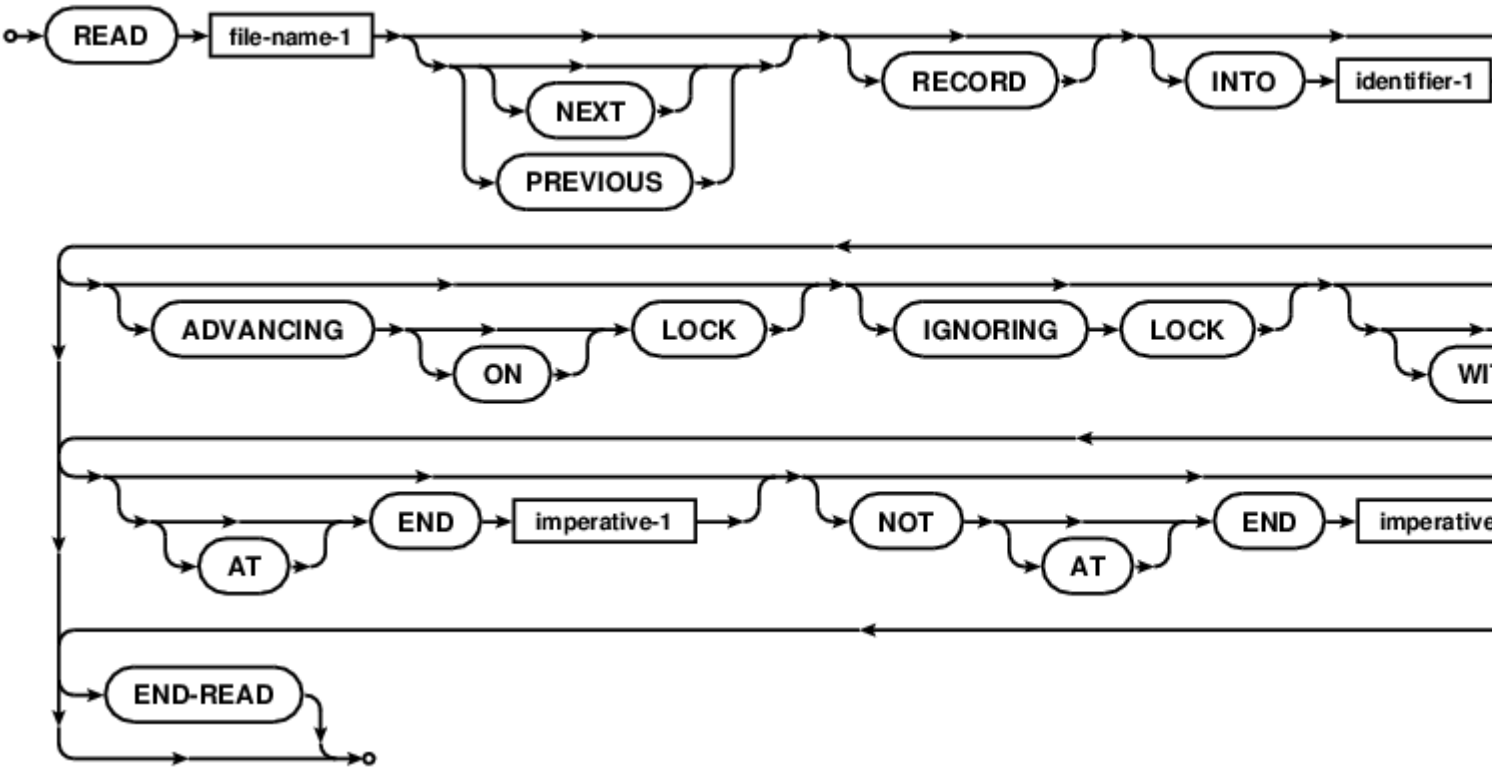
Lea Declaración PERFORM en línea: https://riptutorial.com/es/cobol/topic/7334/declaracion-perform

Capítulo 35: Declaración READ

Observaciones

La instrucción `READ` es un elemento básico de la programación de procesamiento de transacciones COBOL. Lee los datos del almacenamiento externo en la tienda de trabajo. Con o sin bloqueos o compartiendo, secuencialmente, por acceso aleatorio, o por clave. También se pueden especificar cláusulas declarativas para `AT END` , pero algunos programadores prefieren la prueba explícita de `FILE STATUS` .

Como cada recurso de archivo puede contener cualquier tipo de registro en cualquier ranura dada, COBOL es un lenguaje "leer un archivo", "escribir un registro", `READ` toma un nombre de archivo (FD) y depende del programador colocar el registro en una estructura apropiada si se guardan datos heterogéneos en el archivo.



Examples

Lectura simple de FD

```
READ data-file
```

Lea Declaración READ en línea: https://riptutorial.com/es/cobol/topic/7336/declaracion-read

Capítulo 36: Declaración SORT

Observaciones

La instrucción COBOL `SORT` se puede usar para ordenar archivos y tablas en el almacenamiento de trabajo.

Archivo SORT

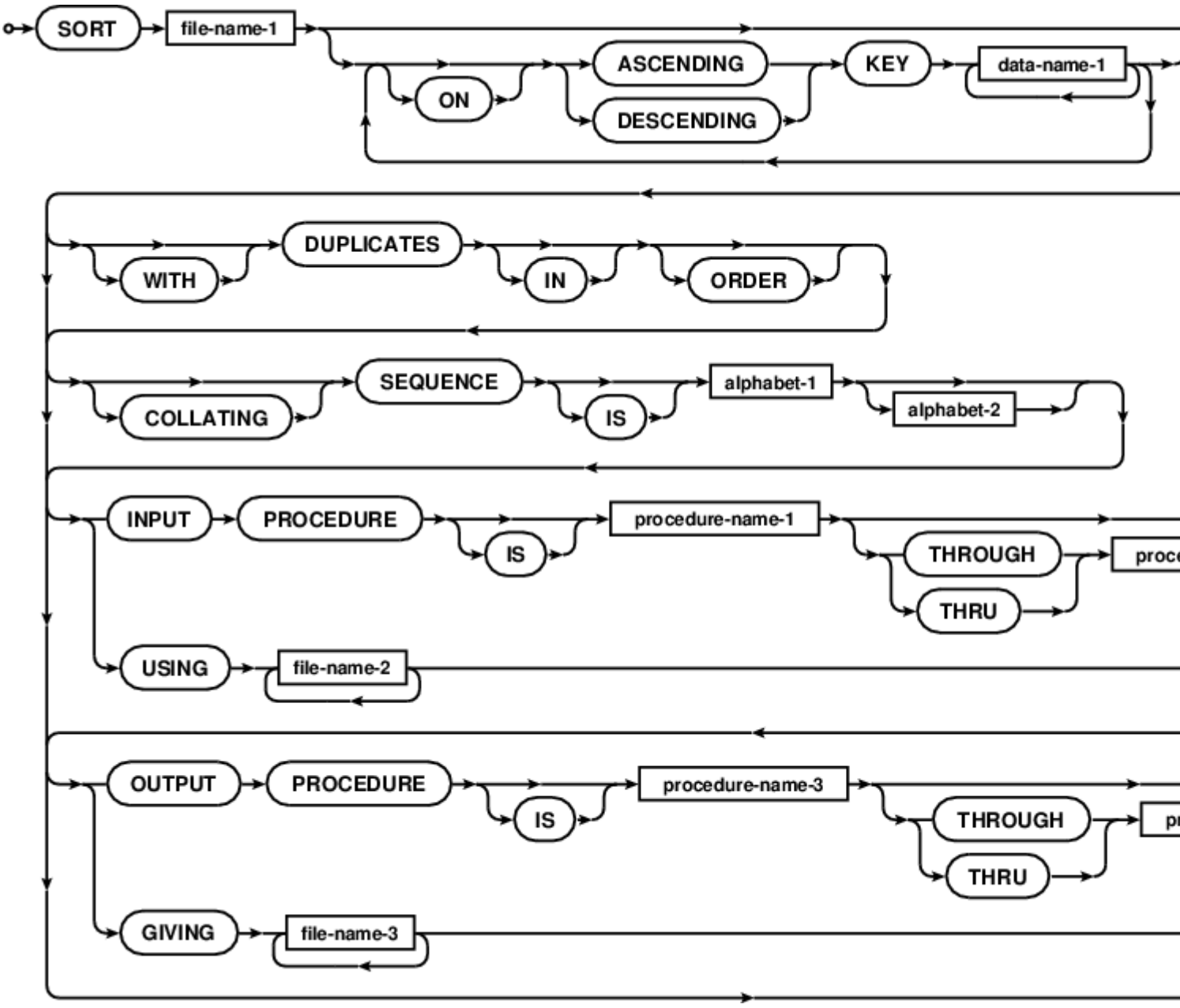
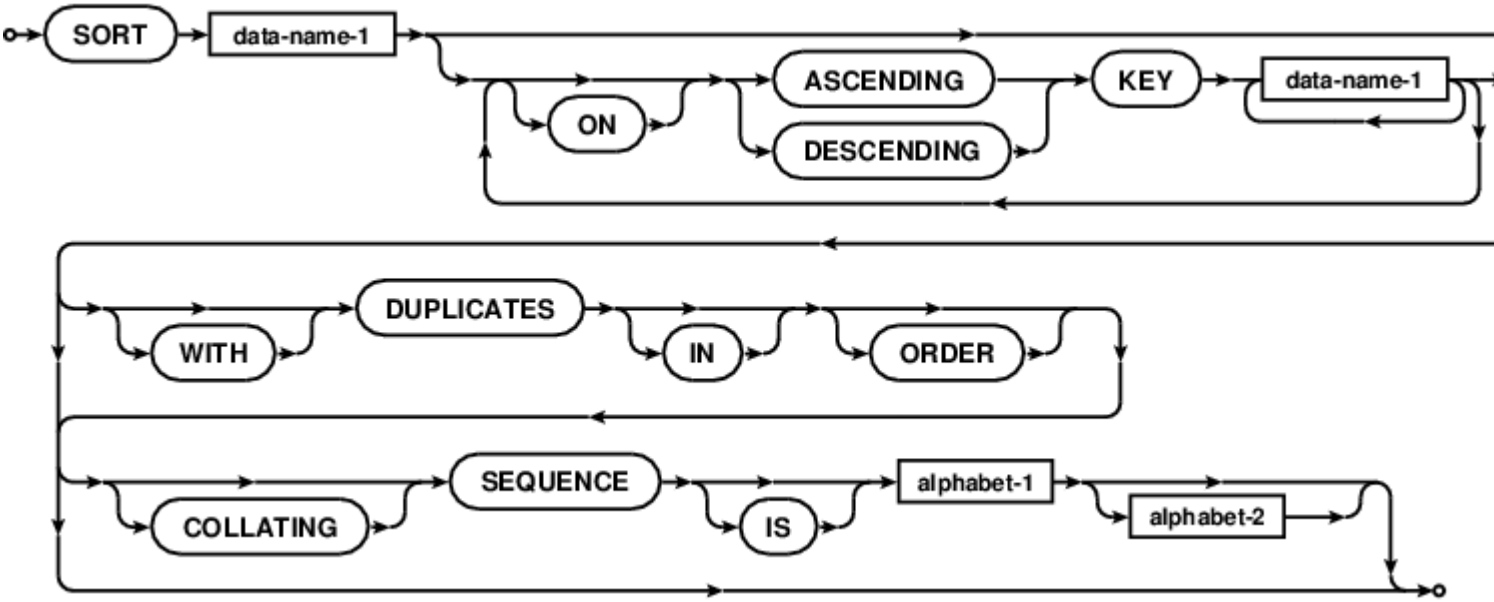


Tabla de clasificación



Examples

Clasificación estándar en estándar hacia fuera

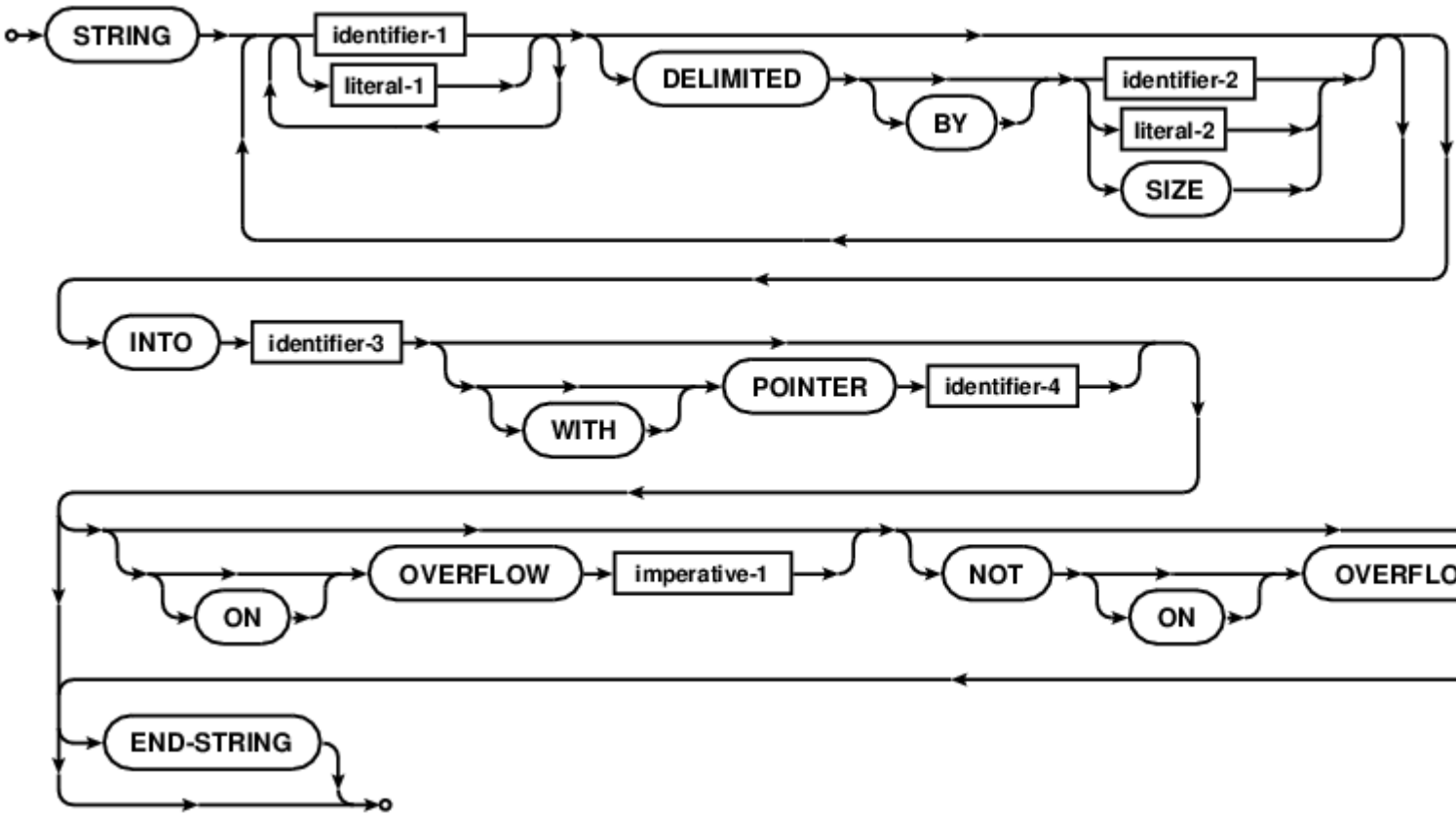
```
GCobol* GnuCOBOL SORT verb example using standard in and standard out
       identification division.
       program-id. sorting.

       environment division.
       input-output section.
       file-control.
           select sort-in
               assign keyboard
               organization line sequential.
           select sort-out
               assign display
               organization line sequential.
           select sort-work
               assign "sortwork".

       data division.
       file section.
       fd sort-in.
          01 in-rec        pic x(255).
       fd sort-out.
          01 out-rec       pic x(255).
       sd sort-work.
          01 work-rec      pic x(255).

       procedure division.
       sort sort-work
           ascending key work-rec
           using  sort-in
           giving sort-out.

       goback.
       exit program.
       end program sorting.
```

Lea Declaración SORT en línea: https://riptutorial.com/es/cobol/topic/7463/declaracion-sort

Capítulo 37: Declaración STRING

Observaciones

La instrucción `STRING` concatena los contenidos parciales o completos de múltiples campos en un solo resultado.



Examples

Ejemplo STRING para cuerdas C

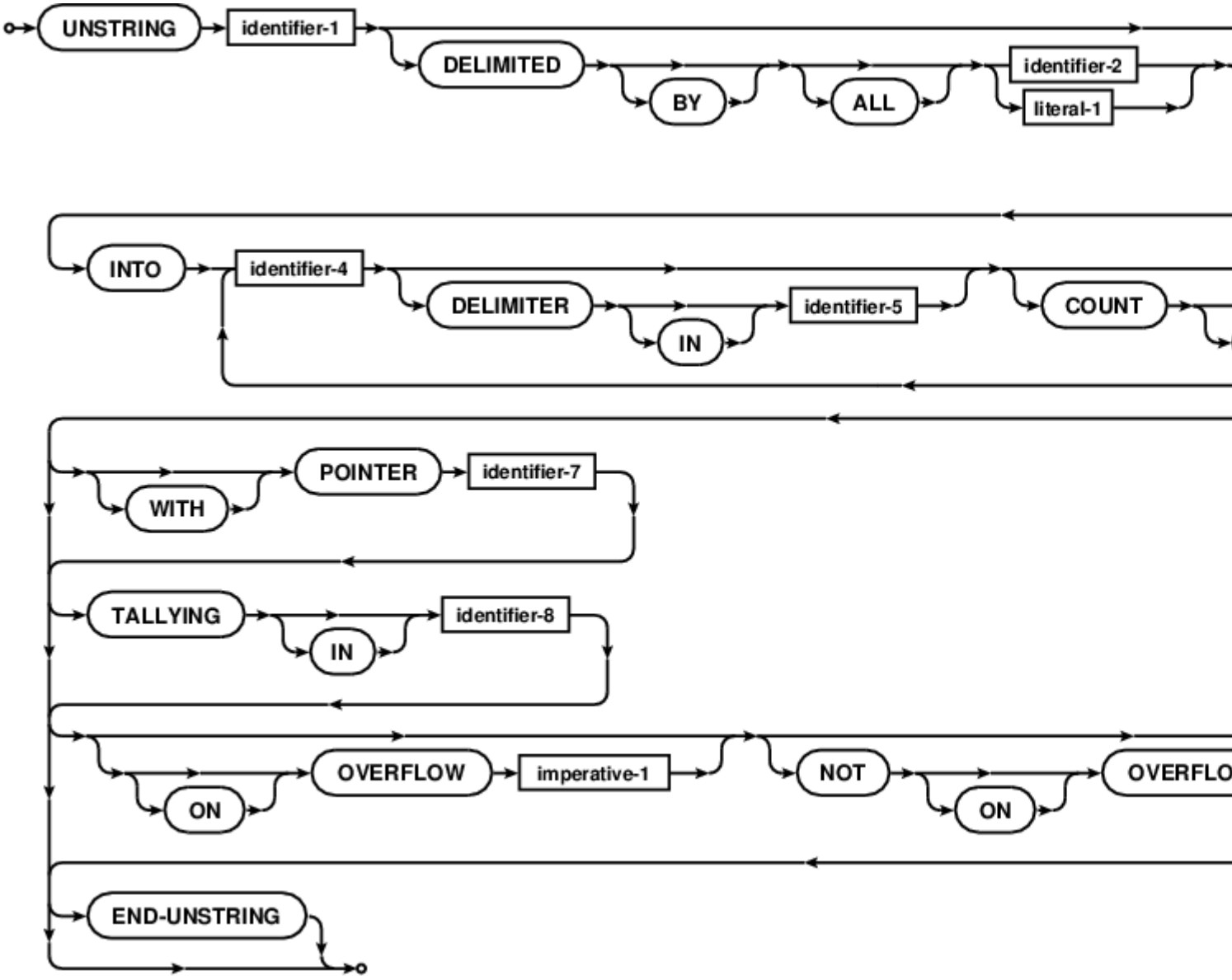
```
*> Strip off trailing zero bytes
STRING c-string DELIMITED BY LOW-VALUE INTO working-store
```

Lea Declaración STRING en línea: https://riptutorial.com/es/cobol/topic/7468/declaracion-string

Capítulo 38: Declaración UNSTRING

Observaciones

La instrucción `UNSTRING` separa un campo de envío y coloca los resultados en múltiples campos de recepción.



Examples

Ejemplo UNSTRING

```
UNSTRING Input-Address
    DELIMITED BY "," OR "/"
    INTO
        Street-Address DELIMITER D1 COUNT C1
        Apt-Number DELIMITER D2 COUNT C2
        City DELIMITER D3 COUNT C3
```

```
        State DELIMITER D4 COUNT C4
        Zip-Code DELIMITER D5 COUNT C5
    WITH POINTER ptr-1
    ON OVERFLOW
        SET more-fields TO TRUE
END-UNSTRING
```

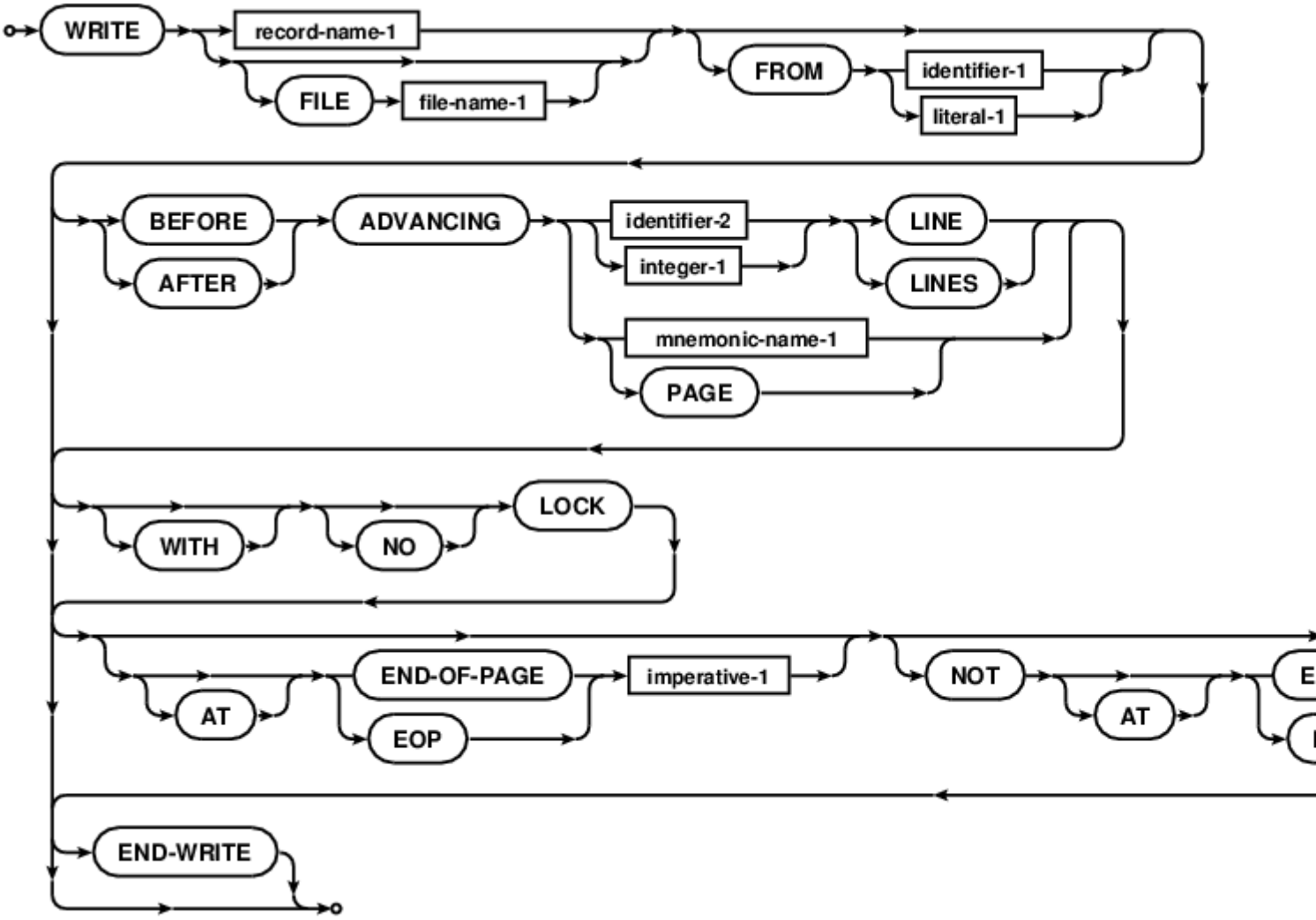
Lea Declaración UNSTRING en línea: https://riptutorial.com/es/cobol/topic/7581/declaracion-unstring

Capítulo 39: Declaración WRITE

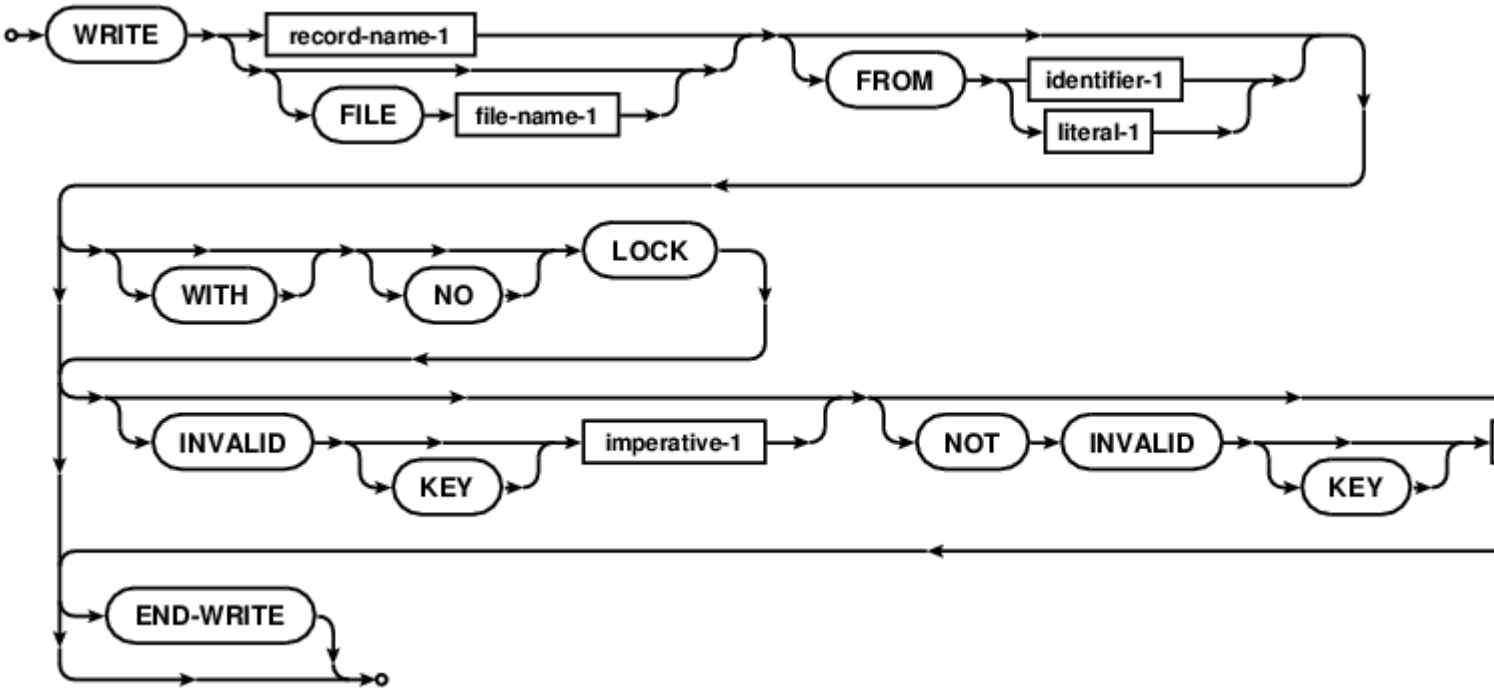
Observaciones

La instrucción `WRITE` libera registros lógicos para un recurso de almacenamiento de `output` o `input-output` , y para el posicionamiento lógico de líneas dentro de una página lógica.

ESCRIBIR secuencial



ESCRIBIR al azar



Examples

ESCRIBIR EJEMPLOS

```
WRITE record-buff

WRITE indexed-record
    WITH LOCK
    ON INVALID KEY
        DISPLAY "Key exists, REWRITING..." END-DISPLAY
        PERFORM rewrite-key
END-WRITE
IF indexed-file-status NOT EQUAL ZERO THEN
    DISPLAY "Write problem: " indexed-file-status UPON SYSERR
    END-DISPLAY
    PERFORM evasive-manoeuvres
END-IF

WRITE record-name-1 AFTER ADVANCING PAGE

WRITE record-name-1 FROM header-record-1
    AFTER ADVANCING 2 LINES
    AT END-OF-PAGE
        PERFORM write-page-header
        PERFORM write-last-detail-reminder
END-WRITE
```

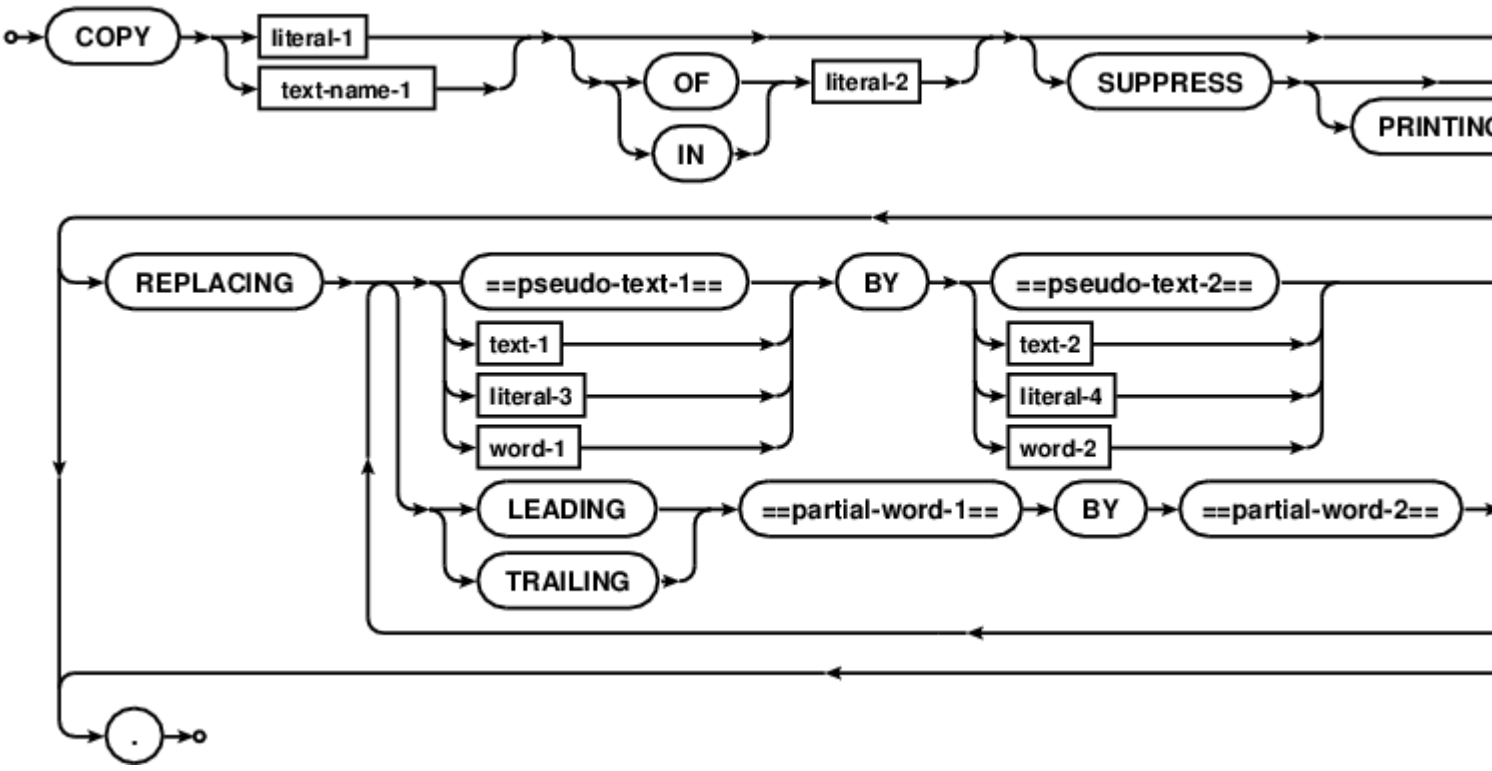
Lea Declaración WRITE en línea: https://riptutorial.com/es/cobol/topic/7583/declaracion-write

Capítulo 40: Directiva COPY

Observaciones

La versión COBOL de la directiva de preprocesador C `#include` incluye. *O, más históricamente exacto, COBOL vino primero, desarrollado unos 10 años antes.*

Debido a algunas de las decisiones de diseño en COBOL (no hay argumentos para `PERFORM` como la razón principal), muchas secuencias de acceso a la estructura de datos necesitan romper el principio DRY . Los nombres de los componentes de la estructura deben repetirse en la DIVISIÓN DE MEDIO AMBIENTE, la DIVISIÓN DE DATOS y posiblemente muchas veces en la DIVISIÓN DE PROCEDIMIENTOS. Esto generalmente se maneja agregando cuadernos. Las declaraciones de registro y el código de acceso se guardan en archivos separados y la instrucción COPY es la única fuente repetida. Un cambio en el libro de copia mantiene todos los usos de la ortografía de nombres y el diseño de los datos sincronizados, en lugar de requerir múltiples ediciones a múltiples archivos cuando ocurre un cambio.



Examples

Copiar el diseño de registro.

programa uno

```
FD important-file.
01 file-record.
   COPY record-layout.
```

```
DATA DIVISION.
01 memory-record.
   COPY record-layout.

PROCEDURE DIVISION.
   ...
   COPY record-move.
   ...
   COPY record-move.
```

programa dos

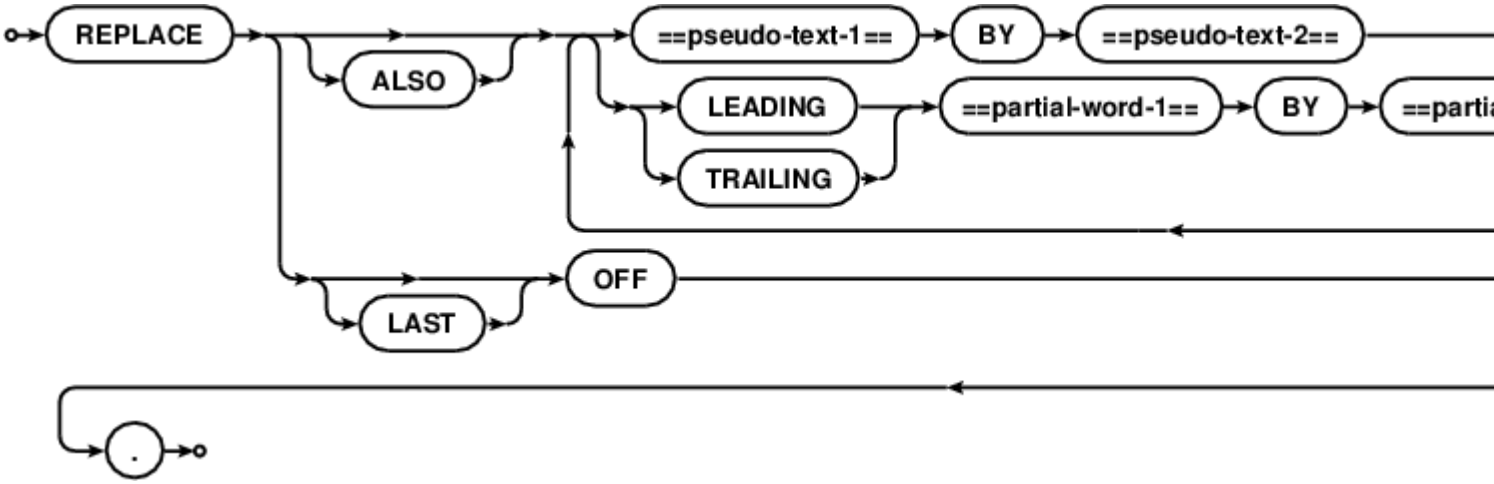
```
   DATA DIVISION.

   01 print-record.
      COPY record-layout.
   ...

   PROCEDURE DIVISION.
   ...
   print-line.
       COPY record-move.
```

Lea Directiva COPY en línea: https://riptutorial.com/es/cobol/topic/6982/directiva-copy

Capítulo 41: Directiva de reemplazo

Observaciones

La directiva `REPLACE` es parte del preprocesador estándar COBOL. Los reemplazos se hacen antes de que comience la compilación.



Examples

REEMPLAZAR muestra de manipulación de texto

```
REPLACE ==magic-number== BY ==65535==.
```

Lea Directiva de reemplazo en línea: https://riptutorial.com/es/cobol/topic/7459/directiva-de-reemplazo

Capítulo 42: División de datos

Introducción

DIVISIÓN DE DATOS es una de las cuatro partes que conforman un programa COBOL. Contiene declaraciones que describen los datos utilizados por el programa. Consta de cuatro secciones: SECCIÓN DE ARCHIVO, SECCIÓN DE ALMACENAMIENTO DE TRABAJO, SECCIÓN DE ALMACENAMIENTO LOCAL y SECCIÓN DE ENLACE.

Examples

Secciones en la división de datos

Las SECCIONES en COBOL pueden ser requeridas u opcionales, según la DIVISIÓN en la que se encuentren.

```
DATA DIVISION.
FILE SECTION.
FD SAMPLE-FILE
01 FILE-NAME PIC X(20).
WORKING-STORAGE SECTION.
01 WS-STUDENT PIC A(10).
01 WS-ID PIC 9(5).
LOCAL-STORAGE SECTION.
01 LS-CLASS PIC 9(3).
LINKAGE SECTION.
01 LS-ID PIC 9(5).
```

En el ejemplo anterior, 01 son números de nivel.

Número de nivel

El número de nivel se utiliza para especificar el nivel de datos en un registro. Se utilizan para diferenciar entre elementos elementales y elementos de grupo. Los elementos elementales se pueden agrupar para crear elementos grupales.

- 01: Registro de entrada de descripción. Número de nivel de grupo es siempre 01.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-NAME               PIC X(25).   ---> ELEMENTARY ITEM
01 WS-SURNAME            PIC X(25).   ---> ELEMENTARY ITEM
01 WS-ADDRESS.                        ---> GROUP ITEM
   05 WS-HOUSE-NUMBER    PIC 9(3).    ---> ELEMENTARY ITEM
   05 WS-STREET          PIC X(15).   ---> ELEMENTARY ITEM
```

- 02 a 49: Artículos elementales

- 66: Renombrar artículos
- 77: Elementos que no pueden ser subdivididos.
- 88: El nivel 88 es un número de nivel especial que se utiliza para mejorar la legibilidad de los programas COBOL y para mejorar las pruebas de FI. Un nivel 88 parece un nivel debajo de otra variable, pero no lo es. No tiene una IMAGEN, pero tiene un valor. Un nivel 88 siempre está asociado con otra variable y es un nombre de condición para esa variable.

```
01 YES-NO PIC X.
88 ANSWER-IS-YES VALUE "Y".
```

Las dos condiciones siguientes comprueban si YES-NO es igual a "Y":

```
IF YES-NO = "Y"
IF ANSWER-IS-YES
```

Se puede utilizar un nombre de condición de nivel 88 para una variable alfanumérica o numérica.

Cláusula de imagen

La CLÁUSULA DE IMAGEN define dos cosas acerca de una variable: el tamaño de la variable (el número de bytes utilizados en la memoria para el valor) y el tipo de datos que se pueden almacenar en la variable.

Lea División de datos en línea: https://riptutorial.com/es/cobol/topic/10859/division-de-datos

Capítulo 43: Funciones intrínsecas

Introducción

Las funciones intrínsecas se incluyen en el estándar COBOL como un conjunto de funciones que devuelven valores de un algoritmo específico, dado cero o más argumentos. Estas funciones intrínsecas se proporcionan como una facilidad del compilador y del sistema de tiempo de ejecución. Los elementos devueltos son campos COBOL temporales y pueden ser datos de caracteres, campos de bits o valores numéricos.

Los ejemplos incluyen funciones trigonométricas, rutinas de fecha y hora, conversiones de tipos de datos, desviación estándar y otros algoritmos de soporte.

Observaciones

COBOL 2014 enumera las siguientes funciones intrínsecas estándar:

```
======================================== ==========
Intrinsic Function                       Parameters
======================================== ==========
FUNCTION ABS                             1
FUNCTION ACOS                            1
FUNCTION ANNUITY                         2
FUNCTION ASIN                            1
FUNCTION ATAN                            1
FUNCTION BOOLEAN-OF-INTEGER              2
FUNCTION BYTE-LENGTH                     1
FUNCTION CHAR                            1
FUNCTION CHAR-NATIONAL                   1
FUNCTION COMBINED-DATETIME               2
FUNCTION COS                             1
FUNCTION CURRENCY-SYMBOL                 0
FUNCTION CURRENT-DATE                    0
FUNCTION DATE-OF-INTEGER                 1
FUNCTION DATE-TO-YYYYMMDD                Variable
FUNCTION DAY-OF-INTEGER                  1
FUNCTION DAY-TO-YYYYDDD                  Variable
FUNCTION DISPLAY-OF                      Variable
FUNCTION E                               0
FUNCTION EXCEPTION-FILE                  0
FUNCTION EXCEPTION-FILE-N                0
FUNCTION EXCEPTION-LOCATION              0
FUNCTION EXCEPTION-LOCATION-N            0
FUNCTION EXCEPTION-STATEMENT             0
FUNCTION EXCEPTION-STATUS                0
FUNCTION EXP                             1
FUNCTION EXP10                           1
FUNCTION FACTORIAL                       1
FUNCTION FORMATTED-CURRENT-DATE          1
FUNCTION FORMATTED-DATE                  2
FUNCTION FORMATTED-DATETIME              Variable
FUNCTION FORMATTED-TIME                  Variable
FUNCTION FRACTION-PART                   1
```

```
FUNCTION HIGHEST-ALGEBRAIC              1
FUNCTION INTEGER                        1
FUNCTION INTEGER-OF-BOOLEAN             1
FUNCTION INTEGER-OF-DATE                1
FUNCTION INTEGER-OF-DAY                 1
FUNCTION INTEGER-OF-FORMATTED-DATE      2
FUNCTION INTEGER-PART                   1
FUNCTION LENGTH                         1
FUNCTION LENGTH-AN                      1
FUNCTION LOCALE-COMPARE                 Variable
FUNCTION LOCALE-DATE                    2
FUNCTION LOCALE-TIME                    2
FUNCTION LOCALE-TIME-FROM-SECONDS       2
FUNCTION LOG                            1
FUNCTION LOG10                          1
FUNCTION LOWER-CASE                     1
FUNCTION LOWEST-ALGEBRAIC               1
FUNCTION MAX                            Variable
FUNCTION MEAN                           Variable
FUNCTION MEDIAN                         Variable
FUNCTION MIDRANGE                       Variable
FUNCTION MIN                            Variable
FUNCTION MOD                            2
FUNCTION MODULE-CALLER-ID               0
FUNCTION MODULE-DATE                    0
FUNCTION MODULE-FORMATTED-DATE          0
FUNCTION MODULE-ID                      0
FUNCTION MODULE-PATH                    0
FUNCTION MODULE-SOURCE                  0
FUNCTION MODULE-TIME                    0
FUNCTION MONETARY-DECIMAL-POINT         0
FUNCTION MONETARY-THOUSANDS-SEPARATOR   0
FUNCTION NATIONAL-OF                    Variable
FUNCTION NUMERIC-DECIMAL-POINT          0
FUNCTION NUMERIC-THOUSANDS-SEPARATOR    0
FUNCTION NUMVAL                         1
FUNCTION NUMVAL-C                       2
FUNCTION NUMVAL-F                       1
FUNCTION ORD                            1
FUNCTION ORD-MAX                        Variable
FUNCTION ORD-MIN                        Variable
FUNCTION PI                             0
FUNCTION PRESENT-VALUE                  Variable
FUNCTION RANDOM                         Variable
FUNCTION RANGE                          Variable
FUNCTION REM                            2
FUNCTION REVERSE                        1
FUNCTION SECONDS-FROM-FORMATTED-TIME    2
FUNCTION SECONDS-PAST-MIDNIGHT          0
FUNCTION SIGN                           1
FUNCTION SIN                            1
FUNCTION SQRT                           1
FUNCTION STANDARD-COMPARE               Variable
FUNCTION STANDARD-DEVIATION             Variable
FUNCTION STORED-CHAR-LENGTH             1
FUNCTION SUM                            Variable
FUNCTION TAN                            1
FUNCTION TEST-DATE-YYYYMMDD             1
FUNCTION TEST-DAY-YYYYDDD               1
FUNCTION TEST-FORMATTED-DATETIME        2
FUNCTION TEST-NUMVAL                    1
```

```
FUNCTION TEST-NUMVAL-C                  2
FUNCTION TEST-NUMVAL-F                  1
FUNCTION TRIM                           2
FUNCTION UPPER-CASE                     1
FUNCTION VARIANCE                       Variable
FUNCTION WHEN-COMPILED                  0
FUNCTION YEAR-TO-YYYY                   Variable
======================================= ==========
```

GnuCOBOL añade

```
======================================= ==========
FUNCTION CONCATENATE                    Variable
FUNCTION SUBSTITUTE                     Variable
FUNCTION SUBSTITUTE-CASE                Variable
======================================= ==========
```

La palabra clave `FUNCTION` es obligatoria a menos que la fuente (o la opción de tiempo de compilación) incluya

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION ALL INTRINSIC.
```

Donde `ALL INTRINSIC` puede ser una lista de funciones que se usarán sin el prefijo `FUNCTION` en las declaraciones de `PROCEDURE DIVISION` .

La función `LENGTH` tiene un historial ordenado. Algunos compiladores incluyen una palabra reservada de `LENGTH` . Para GnuCOBOL, esta palabra reservada solo se reconoce cuando se usa en la frase `LENGTH OF` , el token `OF` se requiere para desambiguar la función de la extensión de palabra reservada más antigua.

Examples

Ejemplo de FUNCTION TRIM

```
01 some-string PIC X(32).

...

MOVE "    a string literal" TO some-string

DISPLAY ":" some-string ":"
DISPLAY ":" FUNCTION TRIM(some-string) ":"
DISPLAY ":" FUNCTION TRIM(some-string LEADING) ":"
DISPLAY ":" FUNCTION TRIM(some-string TRAILING) ":"
```

Demostración

```
:    a string literal           :
:a string literal:
```

```
:a string literal            :
:    a string literal:
```

Mayúsculas

```
MOVE FUNCTION UPPER-CASE("Hello World!") TO SOME-FIELD
DISPLAY SOME-FIELD
```

Salida

```
HELLO WORLD!
```

Función LOWER-CASE

```
MOVE FUNCTION LOWER-CASE("HELLO WORLD!") TO SOME-FIELD
DISPLAY SOME-FIELD
```

Salida

```
hello world!
```

Lea Funciones intrínsecas en línea: https://riptutorial.com/es/cobol/topic/7580/funciones-intrinsecas

Capítulo 44: Instalación de GnuCOBOL con GNU / Linux

Examples

Instalación de GNU / Linux

Para la mayoría de las distribuciones de GNU / Linux, una versión de `GnuCOBOL` está disponible en los repositorios. `GnuCOBOL` fue originalmente `OpenCOBOL` , renombrado cuando el proyecto se convirtió en un proyecto oficial de GNU. Muchos repositorios siguen utilizando `open-cobol` como nombre del paquete (a partir de agosto de 2016).

Para Fedora y otros gestores de paquetes basados en RPM

```
sudo yum install open-cobol
```

Para paquetes basados en Debian, Ubuntu y APT.

```
sudo apt install open-cobol
```

Esta suele ser la versión 1.1 del conjunto de compiladores, y se ocupará de las dependencias de tiempo de compilación y de tiempo de ejecución necesarias cuando se usa GnuCOBOL.

De la fuente (que se encuentra en SourceForge en https://sourceforge.net/projects/open-cobol/) necesitará.

- Paquete de compilación de CA; `build-essential` (o similar)
- Cabeceras de desarrollo BerkeleyDB y BerkelyDB; `libdb` , `libdb-dev` (o nombres similares)
- Biblioteca numérica de precisión múltiple de GNU; `libgmp` , `libgmp-dev`
- Una versión de las `curses` ; `ncurses` , `ncurses-dev`
- El kit de origen, `gnucobol-1.1.tar.gz` (o mejor, `gnucobol-2.0.tar.gz`)
- (Para cambiar las fuentes del compilador, también se requieren herramientas `GNU Autoconf`).

De un directorio de trabajo, de su elección:

```
prompt$ tar xvf gnucobol.tar.gz
prompt$ cd gnucobol
```

Para ver las posibles opciones de configuración, utilice:

```
prompt$ ./configure --help
```

Entonces

```
prompt$ ./configure
```

```
prompt$ make
```

Suponiendo que las dependencias están en su lugar y la compilación se realiza correctamente, verifique la preinstalación con

```
prompt$ make check
```

o

```
prompt$ make checkall
```

Eso ejecuta las comprobaciones internas del compilador (`make check`) y, opcionalmente, ejecuta pruebas en el conjunto de verificación NIST COBOL85 (`make checkall`). La versión 1.1 de OpenCOBOL cubre unas 9100 pruebas NIST, las versiones recientes cubren más de 9700 pases de prueba. *El testuite NIST COBOL85 ya no se mantiene, pero es un conjunto de pruebas muy completo y respetable. COBOL es altamente compatible con versiones anteriores, por intención de diseño, pero las nuevas características COBOL 2002 y COBOL 2014 no son parte del conjunto de verificación NIST.*

Los controles internos cubren unas 500 pruebas y compila el código de muestra.

Si todo está bien, el último paso es

```
prompt$ sudo make install
```

O, para sistemas sin `sudo` , conviértase en el usuario root de `make install` o use un prefijo `./configure` que no requiera permisos de superusuario. El prefijo predeterminado para las compilaciones de origen es `/usr/local` .

Si se ha producido más de una compilación en la máquina y se vuelven a instalar las bibliotecas locales, esto debe ser seguido por

```
prompt$ sudo ldconfig
```

Para asegurarse de que la memoria caché `ld` cargador de vinculador se actualice correctamente para que coincida con la nueva instalación del compilador.

`cobc` estará listo para su uso.

`cobc --help` para obtener ayuda rápida, `info open-cobol` (o `info gnucobol`) para obtener ayuda más `info gnucobol` , y visite http://open-cobol.sourceforge.net/ para obtener enlaces a la Guía del programador y al documento de preguntas frecuentes de más de 1200 páginas.

Los problemas de instalación, los problemas o las preguntas generales se pueden publicar en el espacio del proyecto GnuCOBOL, en las páginas de discusión de `Help getting started` la `Help getting started` en SourceForge.

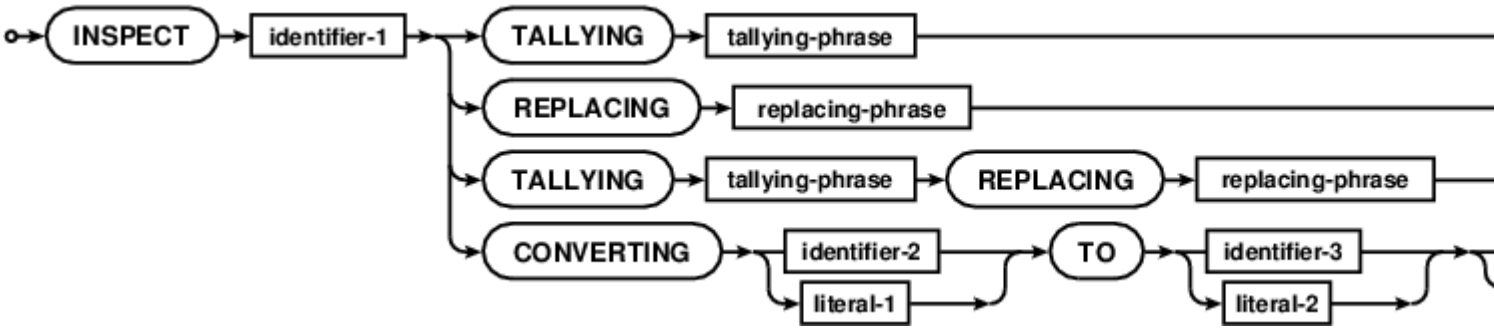
Lea Instalación de GnuCOBOL con GNU / Linux en línea:

https://riptutorial.com/es/cobol/topic/5446/instalacion-de-gnucobol-con-gnu---linux

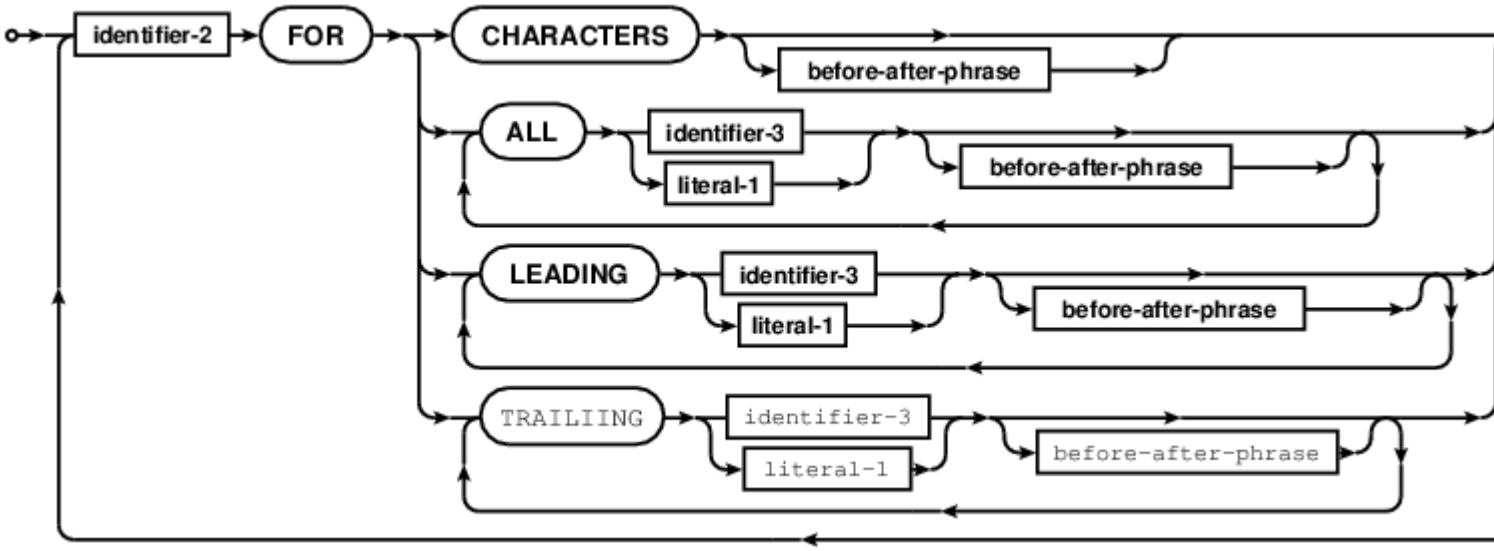
Capítulo 45: Instrucción INSPECT

Observaciones

La instrucción `INSPECT` es un verbo de exploración y reemplazo en COBOL.



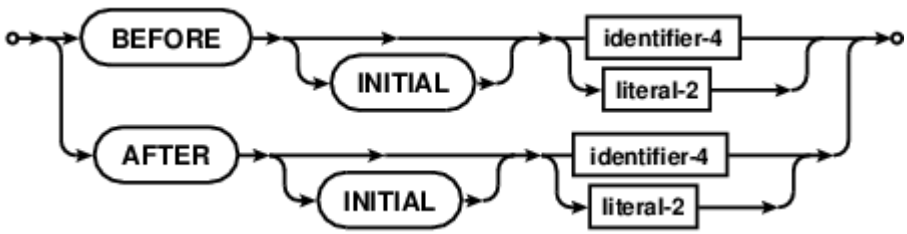
Donde `tallying-phrase` es:



`replacing-phrase` es:

```
missing image
```

`before-after-phrase` es:



Examples

INSPECCIONE reformatear una línea de fecha

```
GCobol identification division.
       program-id. inspecting.

       data division.
       working-storage section.
       01  ORIGINAL            pic XXXX/XX/XXBXX/XX/XXXXXXX/XX.
       01  DATEREC             pic XXXX/XX/XXBXX/XX/XXXXXXX/XX.

       procedure division.

       move function when-compiled to DATEREC ORIGINAL

       INSPECT DATEREC REPLACING ALL "/" BY ":" AFTER INITIAL SPACE

       display "Formatted function WHEN-COMPILED " ORIGINAL
       display " after INSPECT REPLACING         " DATEREC

       goback.
       end program inspecting.
```

Dando:

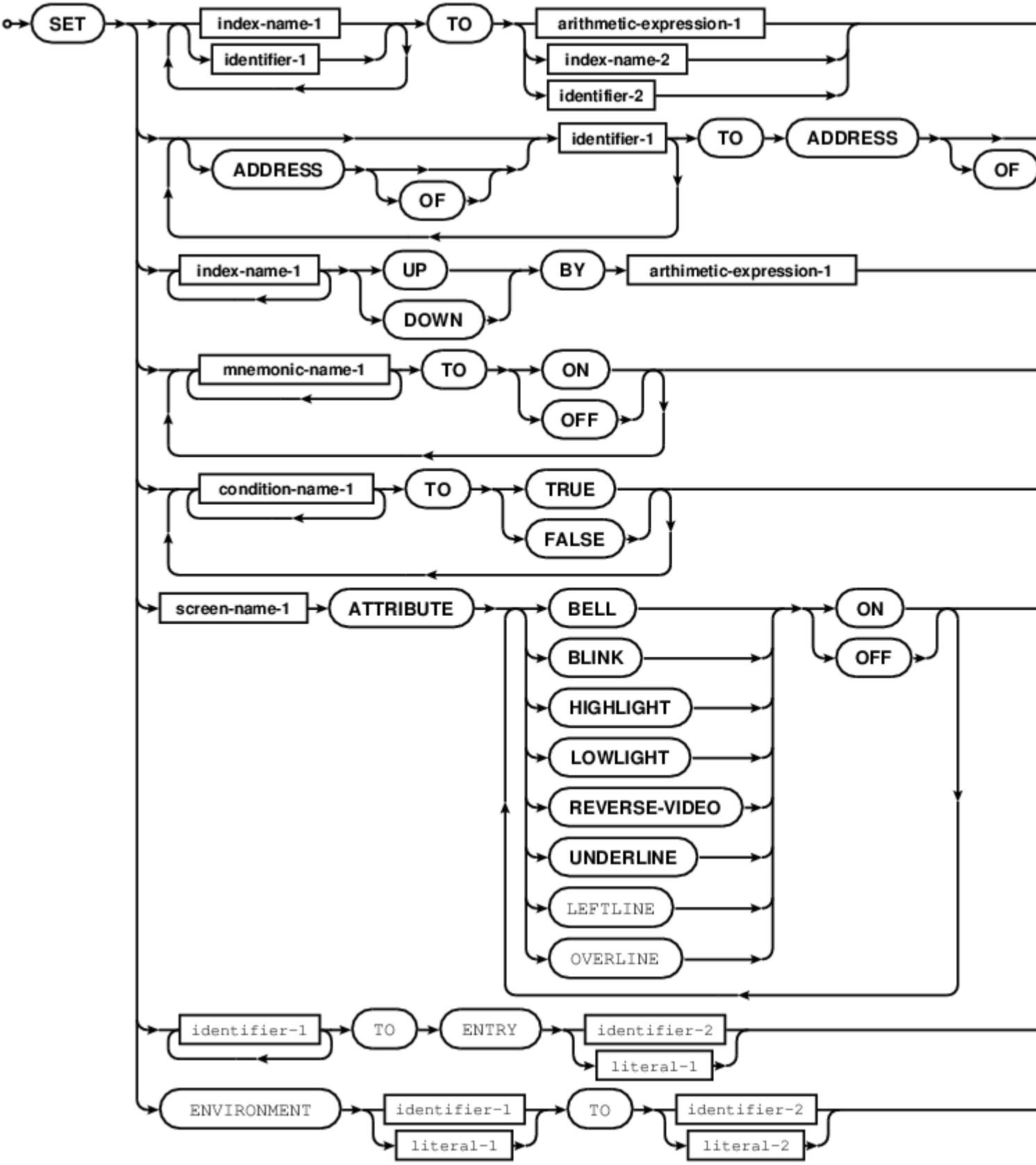
```
Formatted function WHEN-COMPILED 2010/03/25 23/05/0900-04/00
 after INSPECT REPLACING         2010/03/25 23:05:0900-04:00
```

Lea Instrucción INSPECT en línea: https://riptutorial.com/es/cobol/topic/7182/instruccion-inspect

Capítulo 46: Instrucción SET

Observaciones

La instrucción COBOL `SET` establece valores y datos del entorno operativo. Se puede argumentar que `SET` fue utilizado en exceso por el comité, ya que tiene más de una docena de formatos de sintaxis documentados.



Examples

Ejemplo de puntero SET

```
SET handle TO returned-pointer
```

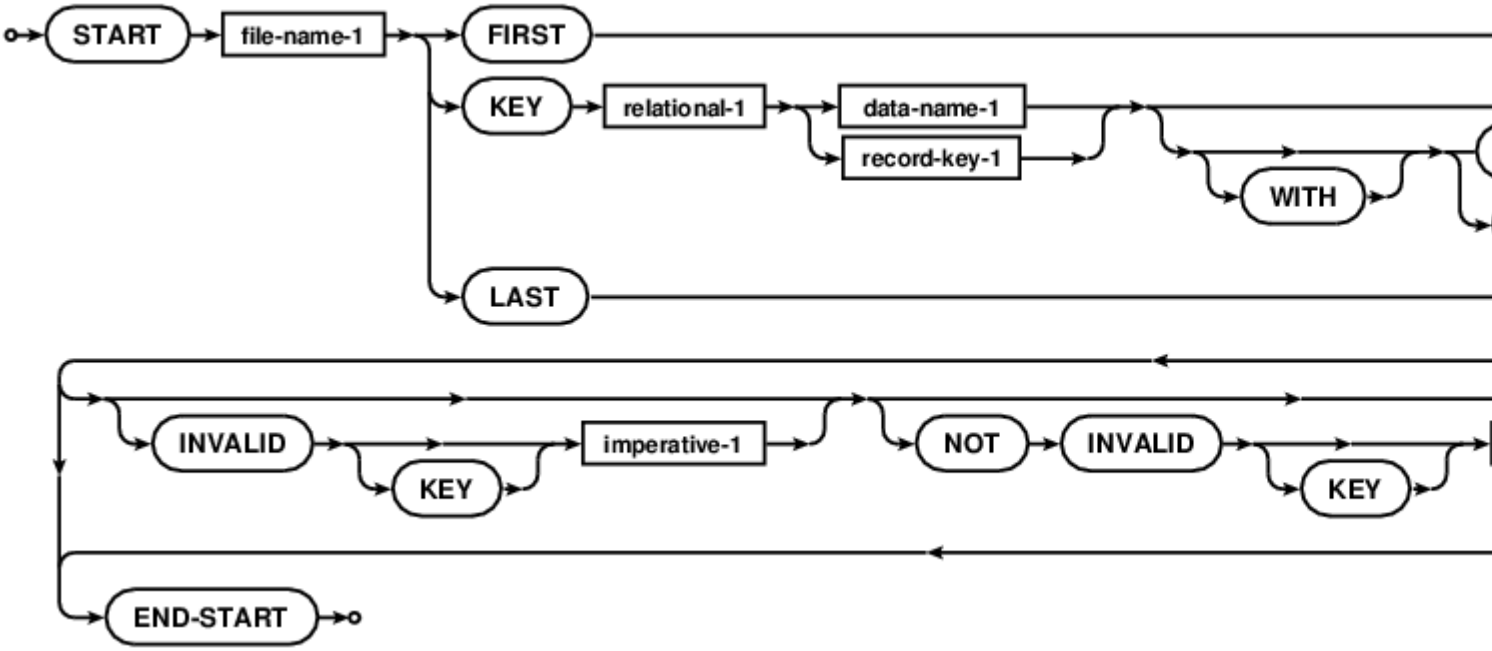
```
SET handle UP BY LENGTH(returned-pointer)
SET ADDRESS OF buffer-space TO handle
MOVE buffer-space TO work-store
DISPLAY "Second element is " work-store
```

Lea Instrucción SET en línea: https://riptutorial.com/es/cobol/topic/7461/instruccion-set

Capítulo 47: Instrucción START

Observaciones

La instrucción `START` proporciona una manera de posicionar una lectura en un archivo para su posterior recuperación secuencial (por clave).



La clave relacional puede incluir (pero no se limita a):

- LA CLAVE ES MAYOR QUE
- La clave es>
- La clave es menos que
- La clave es <
- LA LLAVE ES IGUAL A
- La clave es =
- LA LLAVE NO ES MAYOR QUE
- LA CLAVE NO ES>
- La clave no es menos que
- La clave no es <
- LA LLAVE NO ES IGUAL A
- LA CLAVE NO ES =

- La clave es <>
- LA CLAVE ES MAYOR O IGUAL A
- LA CLAVE ES> =
- LA CLAVE ES MENOS DE O IGUAL A
- LA CLAVE ES <=

Examples

Ejemplo de START

```
start indexing
   key is less than
       keyfield of indexing-record
   invalid key
       display "bad start: " keyfield of indexing-record
       set no-more-records to true
   not invalid key
       read indexing previous record
           at end set no-more-records to true
       end-read
end-start
```

Lea Instrucción START en línea: https://riptutorial.com/es/cobol/topic/7464/instruccion-start

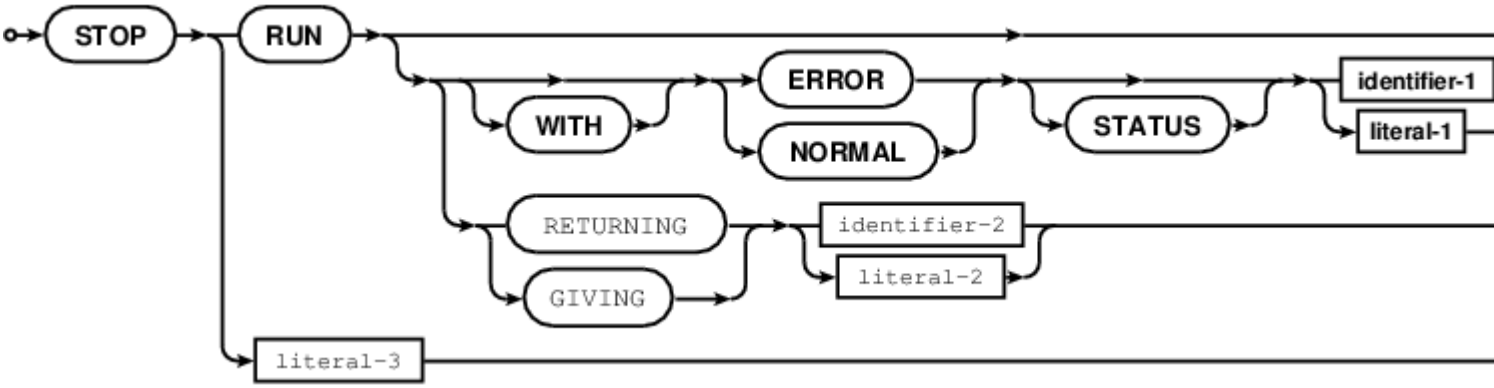
Capítulo 48: Instrucción STOP

Observaciones

La instrucción `STOP` termina la unidad de ejecución actual.

Una extensión ahora obsoleta de `STOP RUN` es `STOP literal` , que pausará un programa hasta que se dé una respuesta de la consola, en la que se reanudará la ejecución del punto. Esto podría ser útil para cosas como "ve a buscar la gran caja de papel y carga la impresora especial".

`STOP` es un programa difícil de terminar, `GOBACK` es una forma un poco mejor de regresar al sistema operativo o al módulo de llamada, especialmente en subrutinas que pueden no tener un negocio que termine una ejecución.



Examples

STOP RUN

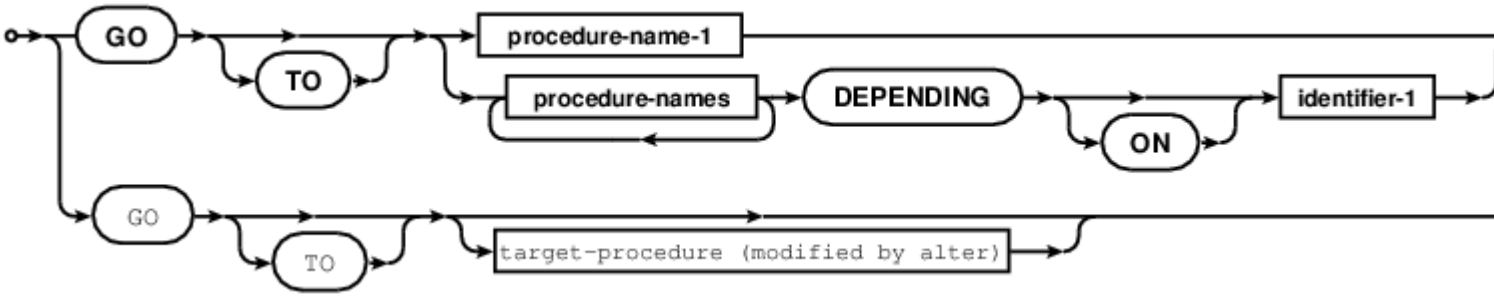
```
STOP RUN
```

Lea Instrucción STOP en línea: https://riptutorial.com/es/cobol/topic/7466/instruccion-stop

Capítulo 49: IR a la declaración

Observaciones

Los muy queridos `GO TO` . COBOL incluye párrafos y secciones con nombre, junto con otras etiquetas, y cualquiera de ellos puede ser el objetivo de una declaración `GO` .



Examples

Declaración GO

```
GO TO label

GO TO label-1 label-2 label-3 DEPENDING ON identifier-1

GO TO label OF section

GO.
```

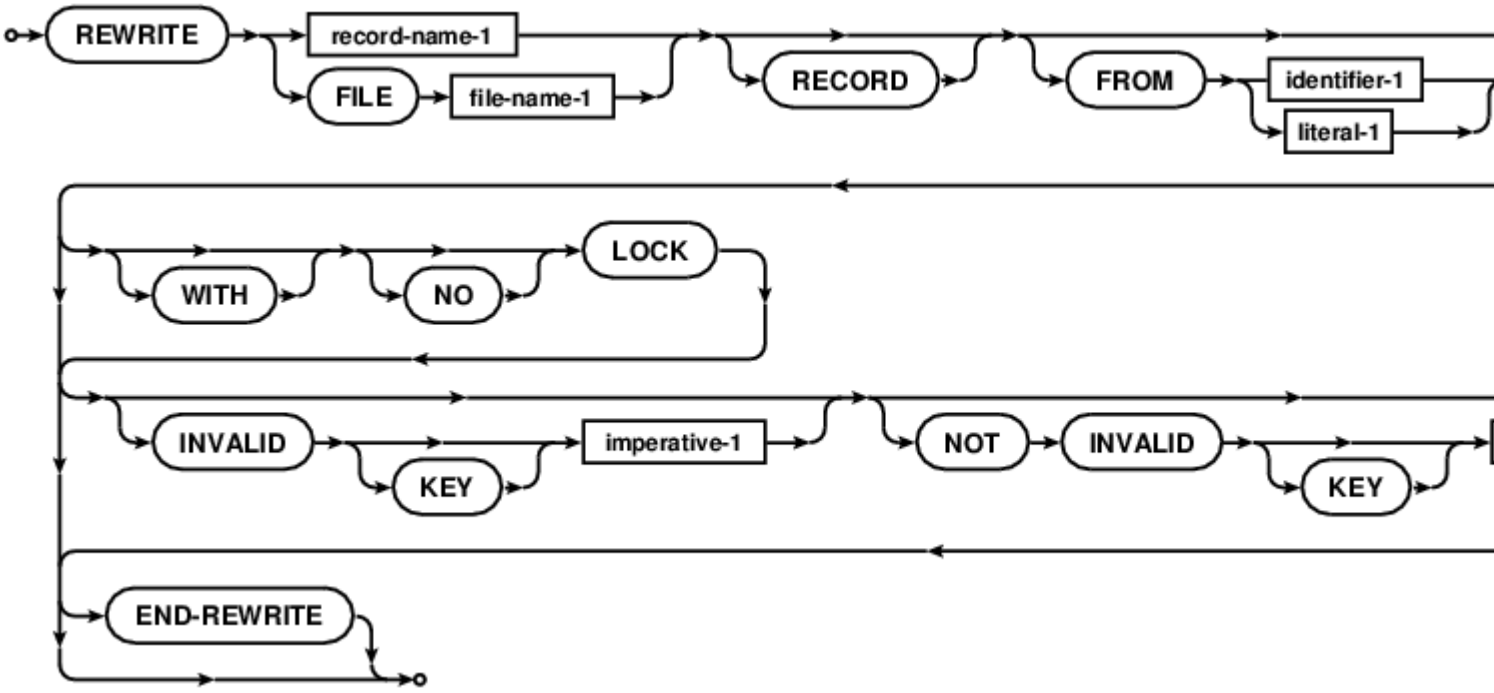
El último ejemplo de línea indica que una instrucción `ALTER` está en juego, y otra parte del código especificará qué `label` real es el objetivo del salto.

Lea IR a la declaración en línea: https://riptutorial.com/es/cobol/topic/7163/ir-a-la-declaracion

Capítulo 50: Reescribir la declaración

Observaciones

La instrucción REWRITE reemplaza lógicamente los registros existentes en el almacenamiento masivo.



Examples

ESCRIBIR de registros en un archivo de acceso RELATIVO

```
GCobol >>SOURCE FORMAT IS FIXED
      *> ****************************************************************
      *> Purpose:   RELATIVE file organization  REWRITE example
      *> Tectonics: cobc -g -debug -W -x relatives.cob
      *> ****************************************************************
       identification division.
       program-id. relatives.

       environment division.
       configuration section.
       repository.
           function all intrinsic.

       input-output section.
       file-control.
           select optional relatives
               assign to "relatives.dat"
               file status is filestatus
               organization is relative
               access mode is dynamic
               relative key is nicknum.
```

```
 data division.
 file section.
 fd relatives.
    01 person.
       05 firstname      pic x(48).
       05 lastname       pic x(64).
       05 relationship   pic x(32).

 working-storage section.
 77 filestatus pic 9(2).
    88 ineof value 1 when set to false is 0.

 77 satisfaction pic 9.
    88 satisfied value 1 when set to false is 0.

 77 nicknum   pic 9(2).

 77 title-line pic x(34).
    88 writing-names value "Adding, Overwriting.  00 to finish".
    88 reading-names value "Which record?         00 to quit".
 77 problem   pic x(80).

 screen section.
 01 detail-screen.
    05           line 1 column 1  from title-line erase eos.
    05           line 2 column 1  value "Record: ".
    05 pic 9(2)  line 2 column 16 using nicknum.
    05           line 3 column 1  value "First name: ".
    05 pic x(48) line 3 column 16 using firstname.
    05           line 4 column 1  value "Last name: ".
    05 pic x(64) line 4 column 16 using lastname.
    05           line 5 column 1  value "Relation: ".
    05 pic x(32) line 5 column 16 using relationship.
    05 pic x(80) line 6 column 1  from problem.

 01 show-screen.
    05           line 1 column 1  from title-line erase eos.
    05           line 2 column 1  value "Record: ".
    05 pic 9(2)  line 2 column 16 using nicknum.
    05           line 3 column 1  value "First name: ".
    05 pic x(48) line 3 column 16 from firstname.
    05           line 4 column 1  value "Last name: ".
    05 pic x(64) line 4 column 16 from lastname.
    05           line 5 column 1  value "Relation: ".
    05 pic x(32) line 5 column 16 from relationship.
    05 pic x(80) line 6 column 1  from problem.

*> -*********-*********-*********-*********-*********-**********-**
 procedure division.
 beginning.

*> Open the file and find the highest record number
*> which is a sequential read operation after START
     open input relatives

     move 99 to nicknum
     start relatives key is less than or equal to nicknum
         invalid key
             move concatenate('NO START' space filestatus)
               to problem
```

```
                move 00 to nicknum
            not invalid key
                read relatives next end-read
        end-start

    *> Close and open for i-o
        close relatives
        open i-o relatives

    *> Prompt for numbers and names to add until 00
        set writing-names to true
        set satisfied to false
        perform fill-file through fill-file-end
            until satisfied

        close relatives

    *> Prompt for numbers to view names of until 00
        open input relatives

        set reading-names to true
        set satisfied to false
        perform record-request through record-request-end
            until satisfied

        perform close-shop
     .
     ending.
        goback.

    *> get some user data to add
     fill-file.
        display detail-screen.
        accept detail-screen.
        move spaces to problem
        if nicknum equal 0
            set satisfied to true
            go to fill-file-end
        end-if.
     .
     write-file.
        write person
            invalid key
                move concatenate("overwriting: " nicknum) to problem
                REWRITE person
                    invalid key
                        move concatenate(
                            exception-location() space nicknum
                            space filestatus)
                        to problem
                END-REWRITE
        end-write.
        display detail-screen

     .
     fill-file-end.
     .

    *> get keys to display
     record-request.
        display show-screen
```

```
           accept show-screen
           move spaces to problem
           if nicknum equals 0
               set satisfied to true
               go to record-request-end
           end-if
       .

      *> The magic of relative record number reads
       read-relation.
           read relatives
               invalid key
                   move exception-location() to problem
               not invalid key
                   move spaces to problem
           end-read
           display show-screen
       .

       record-request-end.
       .

      *> get out <*
       close-shop.
           close relatives.
           goback.
       .
       end program relatives.
```

Lea Reescribir la declaración en línea: https://riptutorial.com/es/cobol/topic/7460/reescribir-la-declaracion

Capítulo 51: Sentencia ALLOCATE

Observaciones

Asigne almacenamiento de trabajo para un elemento BASADO, o asigne un tamaño dado de almacenamiento de almacenamiento dinámico.

Vea también: Declaración GRATIS

Examples

Sentencia ALLOCATE

```
01 pointer-var         usage POINTER.
01 character-field     pic x(80) BASED value "Sample".

ALLOCATE 1024 characters returning pointer-var
ALLOCATE character-field
ALLOCATE character-field INITIALIZED RETURNING pointer-var
```

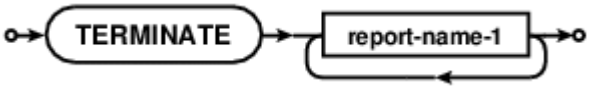
Consulte http://open-cobol.sourceforge.net/faq/index.html#allocate para obtener más detalles.

Lea Sentencia ALLOCATE en línea: https://riptutorial.com/es/cobol/topic/5556/sentencia-allocate

Capítulo 52: Sentencia TERMINATE

Observaciones

La sentencia `TERMINATE` es una característica de COBOL Report Writer. Finaliza el procesamiento en los nombres de informe dados.



Examples

Ejemplo de finalización

```
TERMINATE report-1 report-2 report-summary
```

Lea Sentencia TERMINATE en línea: https://riptutorial.com/es/cobol/topic/7467/sentencia-terminate

Creditos

S. No	Capítulos	Contributors
1	Empezando con Cobol	4444, Abhishek Jain, Bharat Anand, Brian Tiffin, Community, Joe Zitzelberger, ncmathsadist
2	¿Cómo funciona el computacional en cobol?	Bruce Martin, Bulut Colak
3	Cuerda	Jeffrey Ranney, Michael Simpson
4	Declaración ABIERTA	Brian Tiffin
5	Declaración ACCEPT	Brian Tiffin
6	Declaración ADD	Brian Tiffin
7	Declaración ALTER	Brian Tiffin
8	Declaración CALL	4444, Bill Woodger, Brian Tiffin, infoRene, Jeffrey Ranney, Joe Zitzelberger, Simon Sobisch
9	Declaración CANCEL	Brian Tiffin
10	Declaración COMPUTE	Brian Tiffin
11	Declaración CONTINUAR	Brian Tiffin
12	Declaración de búsqueda	Brian Tiffin
13	Declaración de compromiso	Brian Tiffin
14	Declaración de desbloqueo	Brian Tiffin
15	Declaración de evaluación	Brian Tiffin

16	Declaración de INICIACIÓN	Brian Tiffin
17	Declaración de inicialización	Brian Tiffin
18	Declaración de LIBERACIÓN	Brian Tiffin
19	Declaración de MOVE	Brian Tiffin
20	Declaración de réplica	Brian Tiffin
21	Declaración de retorno	Brian Tiffin
22	Declaración de salida	Brian Tiffin
23	Declaración de supresión	Brian Tiffin
24	Declaración DE USO	Brian Tiffin
25	Declaración DELETE	Brian Tiffin
26	Declaración DISPLAY	Brian Tiffin
27	Declaración divisoria	Brian Tiffin
28	Declaración GENERATE	Brian Tiffin
29	Declaración GOBACK	Brian Tiffin
30	Declaración GRATIS	Brian Tiffin
31	Declaración IF	Brian Tiffin
32	Declaración MERGE	Brian Tiffin
33	Declaración MULTIPLY	Brian Tiffin
34	Declaración PERFORM	Brian Tiffin

35	Declaración READ	Brian Tiffin
36	Declaración SORT	Brian Tiffin
37	Declaración STRING	Brian Tiffin
38	Declaración UNSTRING	Brian Tiffin
39	Declaración WRITE	Brian Tiffin
40	Directiva COPY	Brian Tiffin
41	Directiva de reemplazo	Brian Tiffin
42	División de datos	Bulut Colak
43	Funciones intrínsecas	Brian Tiffin, MC Emperor
44	Instalación de GnuCOBOL con GNU / Linux	Brian Tiffin
45	Instrucción INSPECT	Brian Tiffin
46	Instrucción SET	Brian Tiffin
47	Instrucción START	Brian Tiffin
48	Instrucción STOP	Brian Tiffin
49	IR a la declaración	Brian Tiffin
50	Reescribir la declaración	Brian Tiffin
51	Sentencia ALLOCATE	Brian Tiffin
52	Sentencia TERMINATE	Brian Tiffin