EBook Gratuito

APPRENDIMENTO cobol

Free unaffiliated eBook created from
Stack Overflow contributors.

#cobol

Sommario

Di

Capitolo 1: Iniziare con cobol

Osservazioni

COBOL è il **B** usiness **CO** mmon **O** riented programmazione **L** anguage.

Anche se è diventato un nome pronunciabile, COBOL è ancora trattato come un acronimo dal comitato degli standard e COBOL è l'ortografia preferita dagli organismi di standard ISO e INCITS.

Specifiche standard

La specifica attuale è

ISO / IEC 1989: 2014 Tecnologia dell'informazione - Linguaggi di programmazione, relativi ambienti e interfacce software di sistema - Linguaggio di programmazione COBOL

Tale documento è stato pubblicato nel maggio 2014 e può essere acquistato da vari rami di enti standard, ufficialmente ospitati presso

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

Campo principale di utilizzo

Orientato al business. Questo di solito significa elaborazione delle transazioni. Le attività bancarie, le agenzie governative e il settore assicurativo sono aree principali delle implementazioni di applicazioni COBOL. I sistemi mainframe IBM di solito hanno un compilatore COBOL installato. Esistono più di 300 dialetti COBOL, con forse una decina di versioni che fanno la parte del leone delle distribuzioni. La maggior parte di questi compilatori sono sistemi proprietari, ma è disponibile anche il software gratuito COBOL.

Categoria

COBOL è un linguaggio di programmazione procedurale, imperativo, compilato. A partire dalla specifica COBOL 2002, le funzionalità orientate agli oggetti sono state aggiunte allo standard.

Per intento progettuale, COBOL è un linguaggio di programmazione molto prolisso. Sebbene sia consentita la forma algebrica:

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

l'intento iniziale era quello di utilizzare parole complete per le descrizioni computazionali e la manipolazione dei dati:

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

Questa decisione di progettazione ha sia campioni che detrattori. Alcuni ritengono che sia troppo prolisso, mentre altri sostengono che la sintassi consenta una maggiore leggibilità in un ambiente aziendale.

Matematica decimale

COBOL è progettato intorno all'aritmetica decimale, a differenza della maggior parte delle lingue che usano una rappresentazione interna binaria. Le specifiche COBOL richiedono calcoli decimali a virgola fissa molto precisi, un aspetto della lingua che è stato ben considerato nei settori finanziari. *COBOL consente anche BASE UTENTE, ma si appoggia alle rappresentazioni decimali (base 10).*

Storia

COBOL risale alla fine degli anni '50, con le prime implementazioni pubblicate nel 1960.

L'ammiraglio posteriore della Marina degli Stati Uniti, Grace Hopper, è spesso associato a COBOL e si difende a nome della lingua durante le prime fasi di sviluppo. Non è stata l'unica persona coinvolta nella progettazione e nello sviluppo di COBOL, con qualsiasi mezzo, ma viene spesso indicata come la madre di COBOL.

A causa del sostegno anticipato da parte dei governi e delle grandi società, COBOL è stato ampiamente utilizzato per molti decenni. Rimane un punto di orgoglio per alcuni, e una spina per gli altri, che lo vedono come obsoleto. La verità probabilmente sta da qualche parte tra queste visioni estreme. Quando applicato all'elaborazione della transazione, COBOL è a casa. Se applicato a schermate web moderne e applicazioni di rete, potrebbe non essere comodo.

Struttura

I programmi COBOL sono scritti in quattro divisioni separate.

- DIVISIONE DI IDENTIFICAZIONE
- DIVISIONE AMBIENTE
- DATA DIVISION
- DIVISIONE PROCEDURA

Descrizioni dei dati

Essendo progettato per gestire i dati decimali, COBOL consente la descrizione dei dati basata su PICTURE, in gerarchie raggruppate.

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

Ciò definisce il `balance` come un valore a otto cifre firmato con due cifre assunte dopo il punto

decimale. `rate` è di tre cifre prima e tre cifre dopo un punto decimale assunto. `show-balance` è un campo di modifica numerica che avrà un segno di dollaro in testa, sette cifre (soppressione dello zero) con almeno una cifra mostrata prima di due cifre dopo un punto decimale.

`balance` può essere utilizzato nei calcoli, lo `show-balance` è solo a scopo di visualizzazione e non può essere utilizzato nelle istruzioni di calcolo.

Dichiarazioni procedurali

COBOL è un linguaggio pesante con parole chiave riservate. SPOSTARE, COMPARE, MOLTIPLICAMENTE, PERFORMARE le parole in stile lungo formano la maggior parte delle specifiche standard. Oltre 300 parole chiave e 47 dichiarazioni operative nelle specifiche COBOL 2014. Molte implementazioni del compilatore aggiungono ancora di più all'elenco delle parole riservate.

Examples

Ciao mondo

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

I tempi del layout della scheda perforata e degli input solo maiuscoli sono molto indietro. La maggior parte delle implementazioni COBOL continua a gestire lo stesso layout di codice. Anche le implementazioni correnti seguono lo stesso (spesso anche in maiuscolo), compilate e in produzione.

Un'implementazione moderna ben formattata potrebbe essere simile a:

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

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

Con alcune implementazioni di COBOL, questo può essere abbreviato in:

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

Questo formato di solito richiede opzioni di tempo di compilazione per mettere un compilatore COBOL in una modalità di sintassi rilassata, poiché mancano alcune delle istruzioni `DIVISION` normalmente obbligatorie.

COBOL assume le origini del formato FISSO per impostazione predefinita, anche nelle specifiche correnti.

COBOL pre-2002

Colonna	La zona
1-6	Area del numero di sequenza
7	Area Indicatore
8-12	Area A
12-72	Area B
73-80	Area Nome Programma

Gli editor di testo mainframe IBM sono ancora configurati per questo modulo in alcuni casi.

Dopo il 2002 e in COBOL 2014, le aree A e B sono state unite e estese alla colonna 255 e l'area del nome del programma è stata eliminata.

Colonna	La zona
1-6	Area del numero di sequenza
7	Area Indicatore
8-	Area del testo del programma

La colonna 8 attraverso una colonna definita dall'implementazione *Margine R* , di solito è ancora limitata alla colonna 72, ma è consentita dalla specifica di essere eseguita fino alla colonna 255.

COBOL 2002 ha introdotto il testo sorgente `FORMAT FREE` . Non vi è alcuna *area del numero di sequenza* , nessuna *area dell'indicatore* e le linee di origine possono essere di qualsiasi lunghezza (fino a un limite di *Margin R* definito dall'implementazione, in genere inferiore a 2048 caratteri per riga, comunemente 255).

Ma il compilatore inizia in modalità FORMATO FISSO per impostazione predefinita. Di solito è presente un comando di compilazione o un'istruzione *Facility Directive del compilatore* prima che venga riconosciuta l'origine del formato libero.

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

Dove `bbbbbb` rappresenta 6 spazi vuoti o altri caratteri. (Questi vengono ignorati come parte della modalità di default predefinita Sequence Number Area).

Installa gnu-cobol su Mac OS X

gnu-cobol è disponibile tramite il sistema homebrew.

Apri una finestra di terminale da `/Applications/Utilities/Terminal` o usa il tasto `Command+Space` e digita `"Terminal"` .

Se non hai installato il sistema homebrew, aggiungilo digitando, o copiando e incollando nel tuo terminale:

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

Al termine del comando, digita:

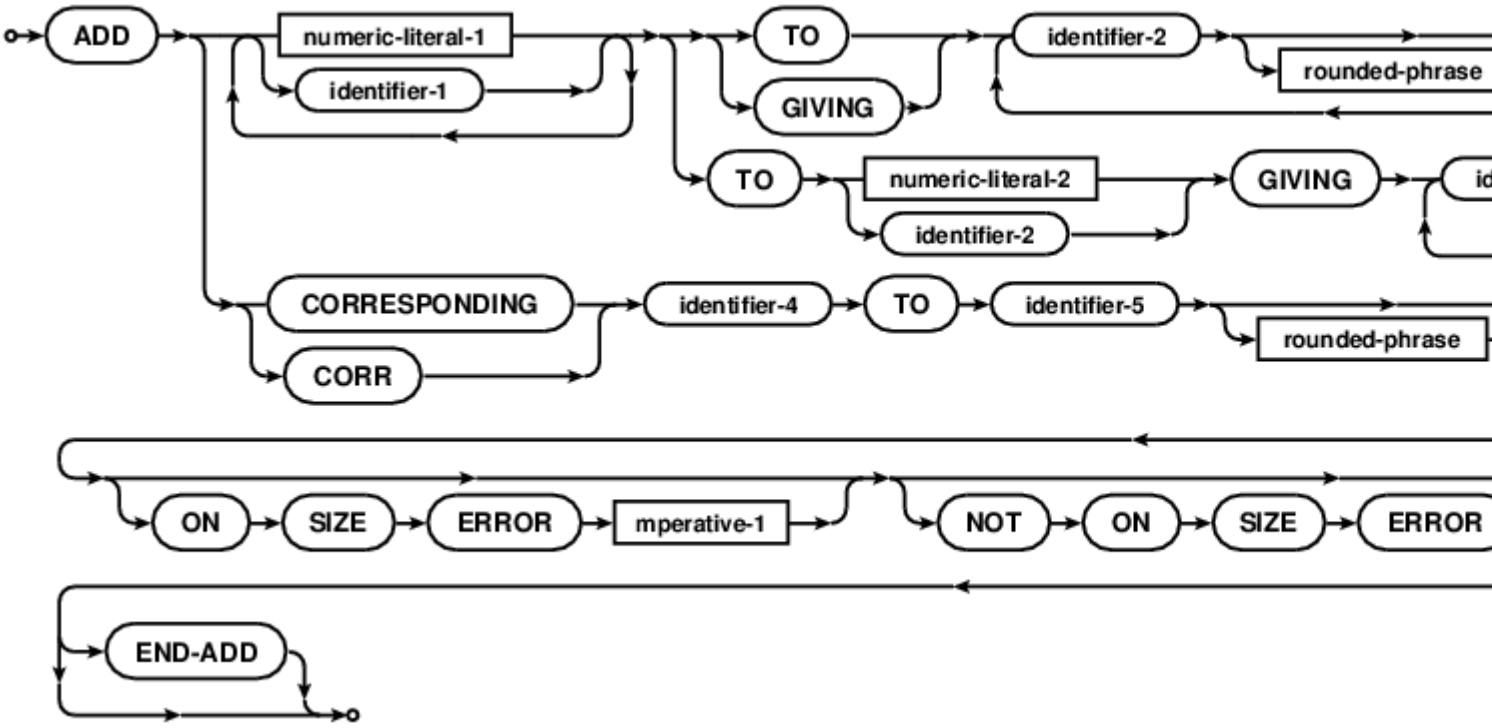
```
brew install gnu-cobol
```

Cioè, ora puoi compilare programmi Cobol sul tuo Mac.

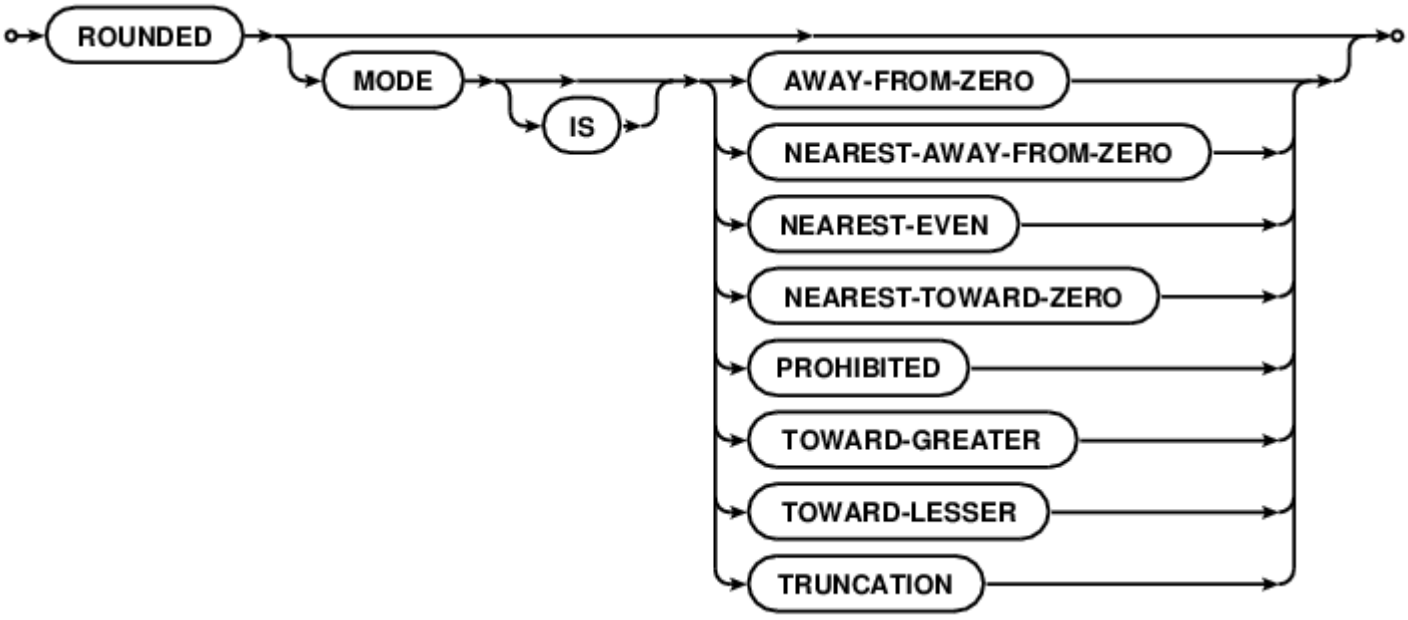
Leggi Iniziare con cobol online: https://riptutorial.com/it/cobol/topic/4728/iniziare-con-cobol

Capitolo 2: ADD dichiarazione

Osservazioni



Dove è arrotondata la fase



Examples

ADD dichiarazione

```
ADD 1 TO cobol
```

Questo modifica la variabile `cobol` . Overflow silenziosamente ignorato.

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

Questo non modifica `cobol` , il risultato `GnuCOBOL` viene memorizzato in `GnuCOBOL` . Ancora una volta, l'overflow dell'assegnazione dello storage viene silenziosamente ignorata (il campo rimarrà al suo vecchio valore sugli errori di dimensione e non ci saranno eccezioni).

```
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
```

Sono consentiti più ingressi, con test delle dimensioni di archiviazione esplicito. COBOL ha una `FUNCTION E` intrinseca `FUNCTION E` , quindi non è una scelta saggia per un identificatore di singola lettera.

`SIZE ERROR` in COBOL dipende dal tipo e / o `PICTURE` . Un campo `PIC 9` memorizzerà solo valori da 0 a 9, un risultato intermedio di 10 innescherebbe la frase `ON SIZE ERROR` in quel caso.

Leggi ADD dichiarazione online: https://riptutorial.com/it/cobol/topic/5533/add-dichiarazione

Capitolo 3: ALLOCATE dichiarazione

Osservazioni

Assegna memoria di lavoro per un articolo BASATO o assegna una dimensione di archiviazione heap.

Vedi anche: dichiarazione GRATUITA

Examples

ALLOCATE dichiarazione

```
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
```

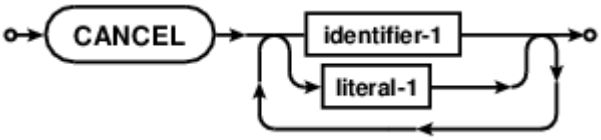
Vedi http://open-cobol.sourceforge.net/faq/index.html#allocate per maggiori dettagli.

Leggi ALLOCATE dichiarazione online: https://riptutorial.com/it/cobol/topic/5556/allocate-dichiarazione

Capitolo 4: ANNULLA dichiarazione

Osservazioni

L'istruzione CANCEL garantisce che un programma referenziato si trovi in uno stato iniziale alla successiva chiamata e che scarichi tutte le risorse per il modulo.



Examples

ANNULLA dichiarazione

```
CALL "submodule"
CALL "submodule"

CANCEL "submodule"
CALL "submodule"
```

Tutti i dati statici nel working set del `submodule` saranno in uno stato iniziale sull'ultima istruzione `CALL` cui sopra. Il secondo `CALL` avrà qualsiasi valore iniziale impostato come rimasto dalla prima `CALL` .

I compilatori COBOL possono supportare l'annullamento fisico (oggetto scaricato dalla memoria) e / o l'annullamento virtuale (garantire uno stato iniziale, ma lasciare l'oggetto disponibile all'ambiente operativo host). Questo è un dettaglio di implementazione.

Vedi http://open-cobol.sourceforge.net/faq/index.html#cancel per maggiori dettagli.

Leggi ANNULLA dichiarazione online: https://riptutorial.com/it/cobol/topic/5600/annulla-dichiarazione

Capitolo 5: Come funziona il calcolo in cobol?

introduzione

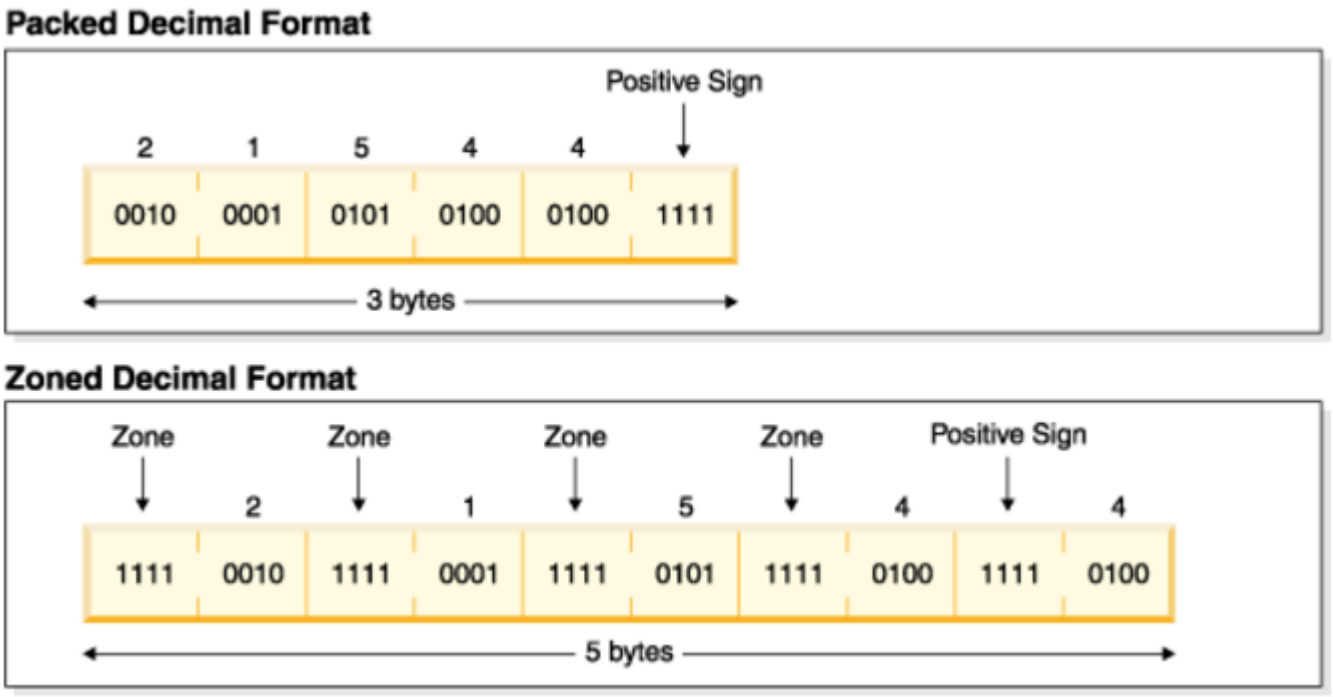
La clausola computazionale viene utilizzata per descrivere il tipo di memoria utilizzata in COBOL. È utilizzato per 3 modi: COMP-1, COMP-2 e COMP-3. La forma più comune di calcolo è COMP-3. Spesso viene chiamato semplicemente "COMP" dai programmatori.

Examples

COMP-3

L'elemento di dati viene memorizzato in formato decimale compresso in COMP-3. Il formato decimale compresso indica che ogni byte di memoria (tranne il byte di ordine inferiore) può contenere due numeri decimali. Il byte di ordine inferiore contiene una cifra nella parte più a sinistra e il segno (positivo o negativo) nella parte più a destra.

"Formato decimale suddiviso in zone" nell'immagine sotto è la memoria predefinita per un numero in COBOL.




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

L'archiviazione computazionale viene spesso utilizzata per ridurre le dimensioni di un file.

Implementazioni comuni

Come comp, comp-1 ... comp-5 sono implementati dipende dall'implementazione.

```
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
```

I compilatori IBM normalmente supportano Comp, Comp-4, Comp-5 in dimensioni di 2,4,8 byte. Dimensioni del supporto GNU GNU di 1,2,4,8.

Comp-1, Comp-2 campi sono definiti senza una clausola di immagine:

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

Per gli altri Comp sono inserite le immagini:

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

Leggi Come funziona il calcolo in cobol? online: https://riptutorial.com/it/cobol/topic/10873/come-funziona-il-calcolo-in-cobol-

Capitolo 6: CONTINUA la dichiarazione

Osservazioni

L'istruzione CONTINUE fa proseguire il flusso di controllo alla successiva istruzione. Non proprio un no-op, in quanto può influenzare il flusso di controllo quando si trovano all'interno di sequenze di istruzioni composte, in particolare IF / THEN / ELSE.

Un utile? l'esempio è durante lo sviluppo e la costruzione iniziali con e senza l'ausilio di debug.

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

Quel codice, sebbene costoso, consentirà il dump della memoria formattata quando il modulo `CBL_OC_DUMP` è collegato nell'eseguibile, ma fallirà in modo innocuo quando non lo è. * Quel trucco è applicabile solo durante le prime fasi di sviluppo. La spesa di un errore di ricerca dinamico non è qualcosa da lasciare nel codice attivo e quelle linee dovrebbero essere rimosse dall'origine non appena tutte le preoccupazioni iniziali sono soddisfatte nel test alfa. Il primo giorno di programmazione, può essere un aiuto pratico. La codifica del secondo giorno ON ECCEPTION CONTINUE dovrebbe essere cancellata.

Examples

segnaposto

Questo è inventato; ma alcuni programmatori COBOL potrebbero preferire la chiarezza positiva, rispetto all'uso di `NOT` in espressioni condizionali (specialmente con l'errore logico incline `var NOT = value OR other-value`).

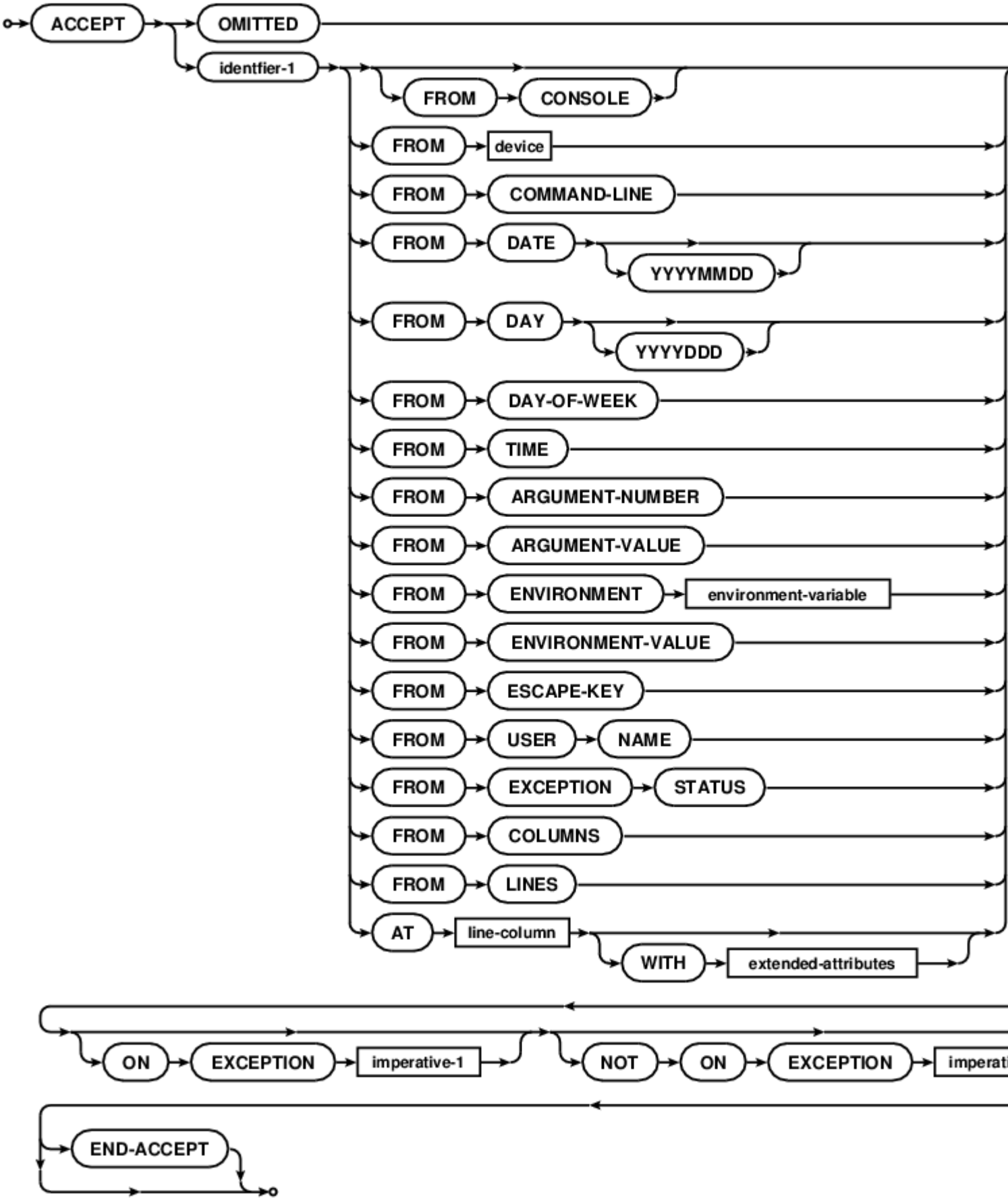
```
if action-flag = "C" or "R" or "U" or "D"
    continue
else
    display "invalid action-code" upon syserr
    perform report-exception
    exit section
end-if
```

Leggi CONTINUA la dichiarazione online: https://riptutorial.com/it/cobol/topic/6981/continua-la-dichiarazione

Capitolo 7: Dichiarazione ACCEPT

Osservazioni

L'istruzione COBOL ACCEPT viene utilizzata per recuperare i dati dal sistema.



Examples

Dichiarazione 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.
```

Vedi http://open-cobol.sourceforge.net/faq/index.html#accept per maggiori dettagli.

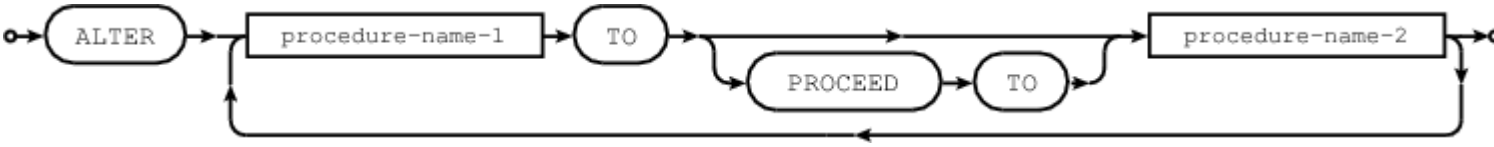
Leggi Dichiarazione ACCEPT online: https://riptutorial.com/it/cobol/topic/5512/dichiarazione-accept

Capitolo 8: Dichiarazione ALTER

Osservazioni

La tanto amata dichiarazione ALTER. Cambia l'obiettivo di un paragrafo GO TO.

Non fa più parte dello standard COBOL, ancora supportato da molti compilatori per ragioni di compatibilità con le versioni precedenti. *(Il diagramma della sintassi è disattivato per mostrare che questo non è più COBOL standard).*



Examples

Un esempio forzato 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 un campione di prova di

```
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$
```

Vedi http://open-cobol.sourceforge.net/faq/index.html#alter per maggiori dettagli.

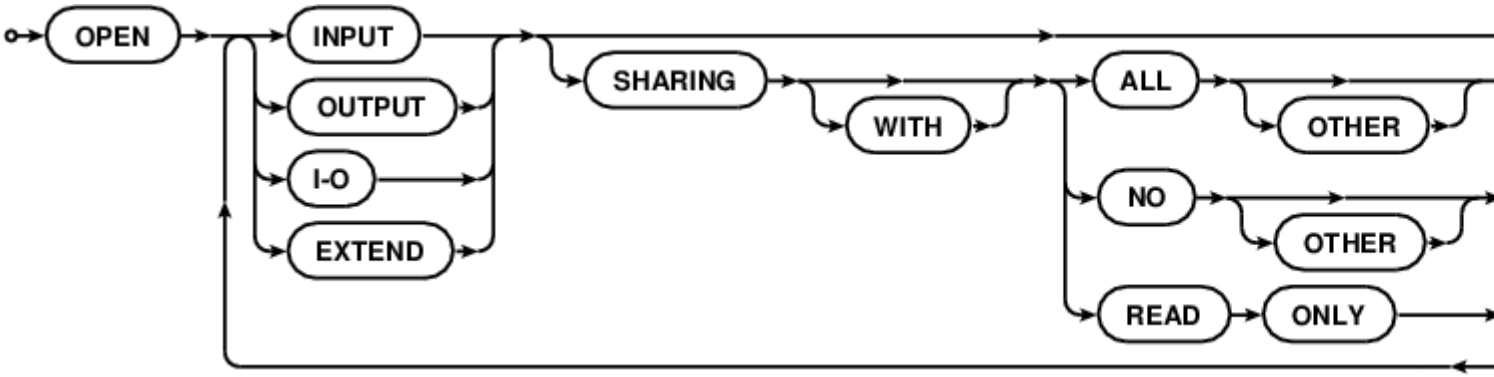
Leggi Dichiarazione ALTER online: https://riptutorial.com/it/cobol/topic/5584/dichiarazione-alter

Capitolo 9: Dichiarazione APERTA

Osservazioni

L'istruzione COBOL `OPEN` avvia l'elaborazione dei file. Le risorse file in COBOL sono definite nella `ENVIRONMENT DIVISION` , denominata nei paragrafi `FD` (File Descriptor). Questi nomi fd vengono utilizzati per accedere ai file del disco fisico e varie opzioni sono specificate in una clausola `SELECT` nel paragrafo `FILE-CONTROL` della sezione `INPUT-OUTPUT SECTION` . Si prevede che un programmatore verifichi un identificatore di `FILE STATUS` per i codici di stato e di errore.

Le modalità includono `INPUT` , `OUTPUT` , `IO` e `EXTEND` .



Examples

Esempio APERTO, con mini report 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
```

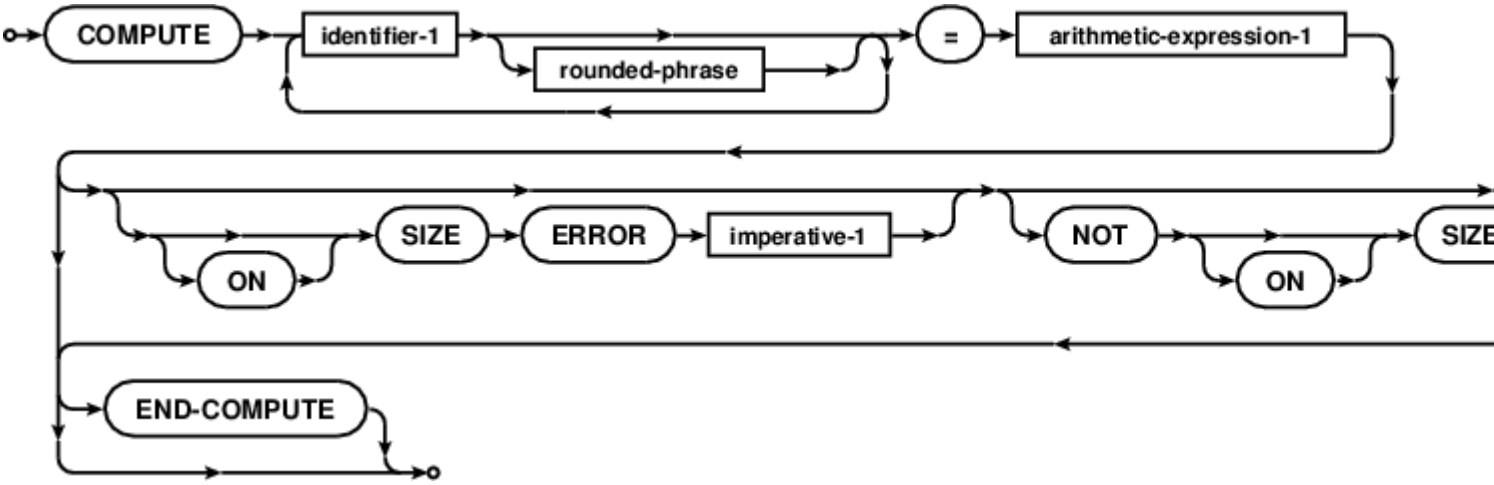
```
           after advancing page
       end-write.

       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.
```

Leggi Dichiarazione APERTA online: https://riptutorial.com/it/cobol/topic/7288/dichiarazione-aperta

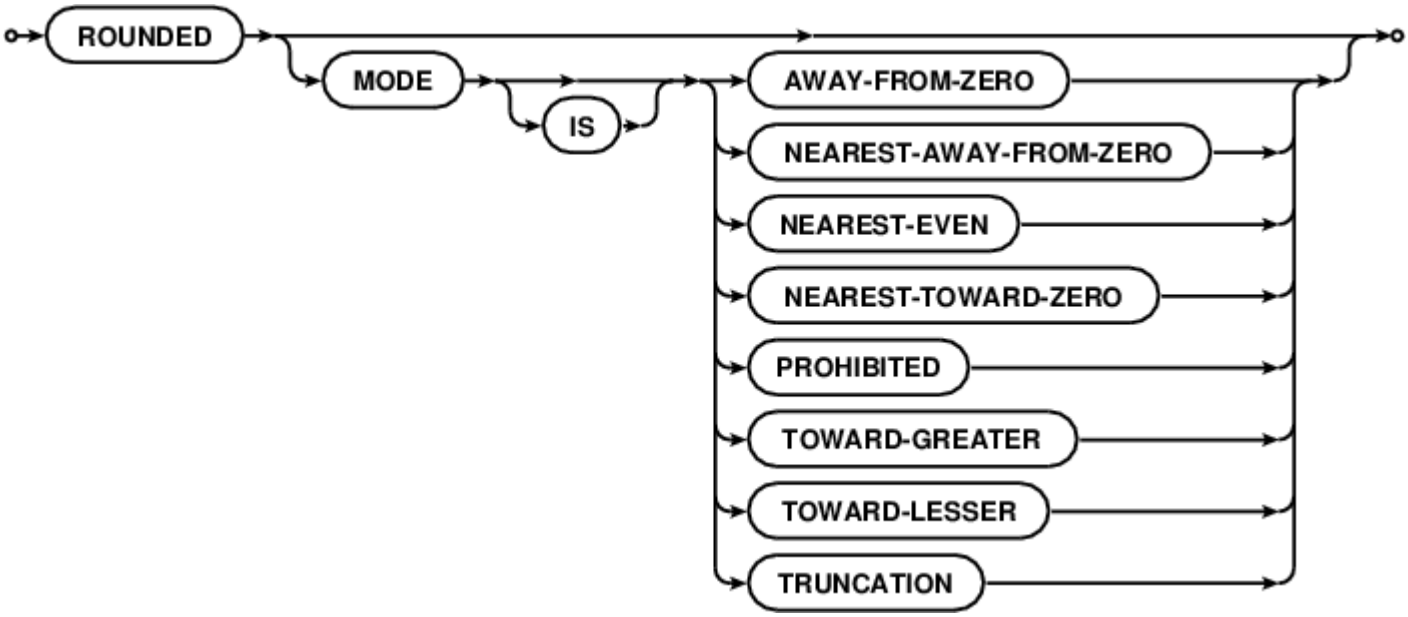
Capitolo 10: Dichiarazione COMPUTE

Osservazioni

L'istruzione COMPUTE consente espressioni di calcolo algebrico.



La frase arrotondata è



Examples

Consiglio: utilizza gli spazi attorno a tutti i componenti

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

Questo è un riferimento alla variabile `var-1` e non a `var - 1` .

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

Consigliato, *opinione* .

Leggi Dichiarazione COMPUTE online: https://riptutorial.com/it/cobol/topic/6726/dichiarazione-compute

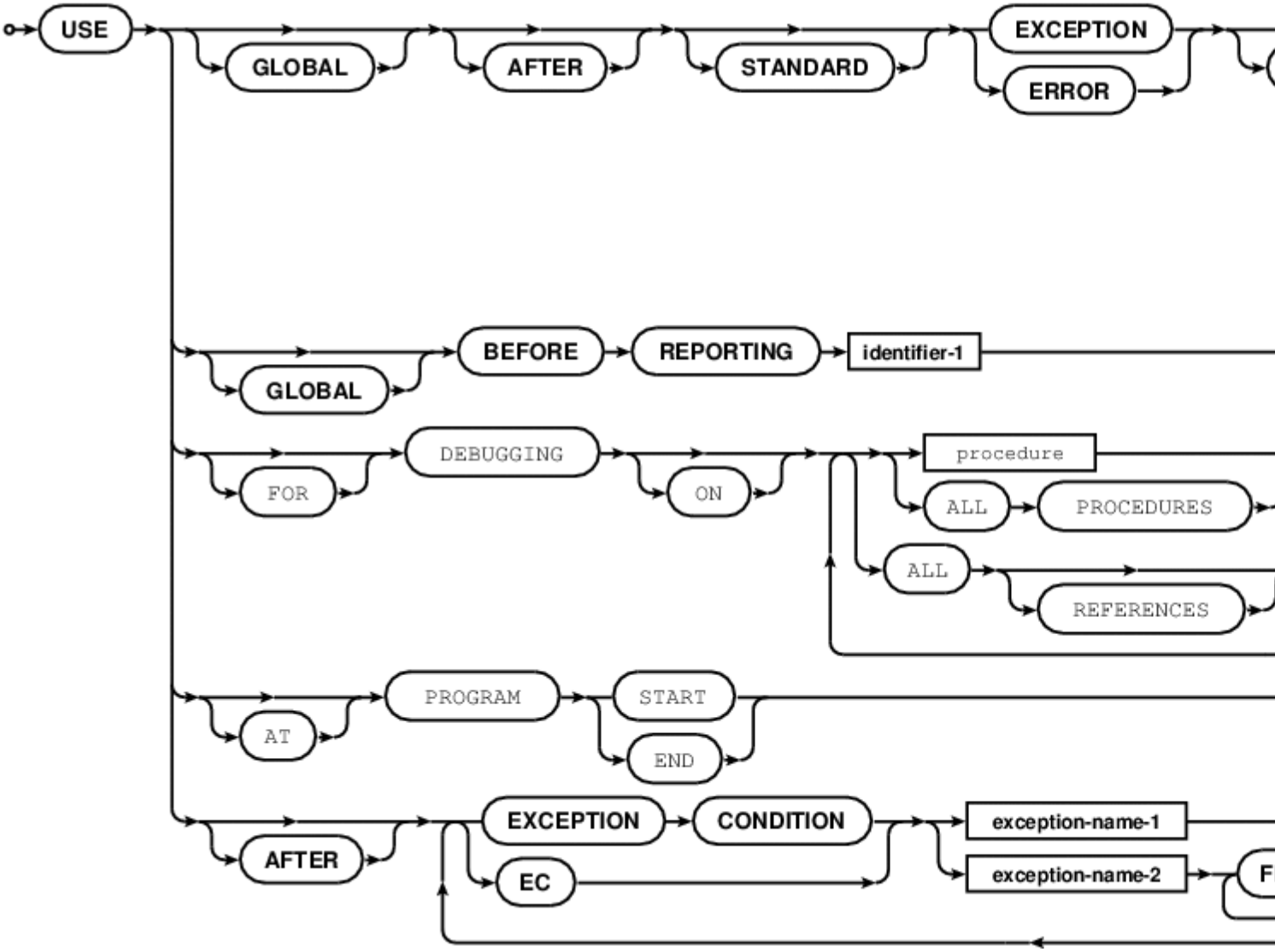
Capitolo 11: Dichiarazione d'uso

Osservazioni

L'istruzione `USE` specifica le procedure da utilizzare

- per errori e gestione delle eccezioni oltre a quelli forniti da altre strutture
- prima di stampare un gruppo di report designato
- dopo il rilevamento delle condizioni di eccezione designate

L'utilizzo obsoleto include la specifica delle procedure da utilizzare durante il `DEBUGGING` e le estensioni includono l'aggiunta di procedure interstiziali per l'inizio e la fine del programma.



Examples

Dichiarazione di USE con Report Writer DECLARATIVES

```
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.
```

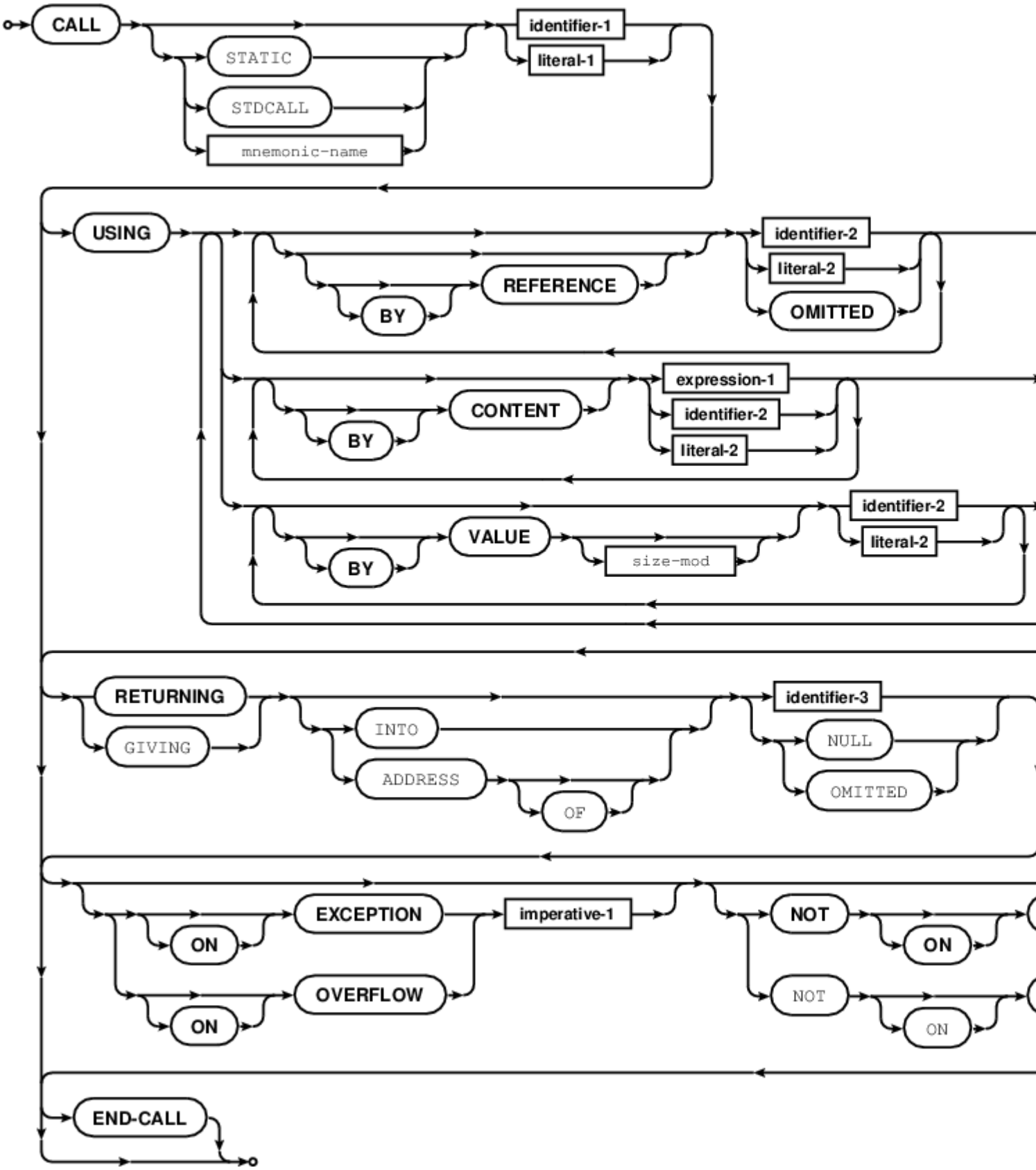
```
041700     EXIT.
041800
041900 END DECLARATIVES.
```

Leggi Dichiarazione d'uso online: https://riptutorial.com/it/cobol/topic/7582/dichiarazione-d-uso

Capitolo 12: Dichiarazione di CHIAMATA

Osservazioni

L'istruzione COBOL CALL fornisce l'accesso alle routine di libreria compilate.



Examples

Dichiarazione di CHIAMATA

COBOL può utilizzare il collegamento statico per la seguente dichiarazione. GnuCOBOL utilizza il collegamento dinamico per impostazione predefinita per tutti i simboli esterni noti al momento

della compilazione, anche quando il simbolo è letterale:

```
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
```

Questa affermazione impone la risoluzione di modifica del collegamento al tempo di compilazione. *(Estensione standard, non sintassi)* :

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

I campi in COBOL possono essere passati `BY REFERENCE` (l'impostazione predefinita, fino a quando non viene sovrascritta - le sostituzioni sono `sticky` in un ordine da sinistra a destra), `BY CONTENT` (una copia viene passata DA RIFERIMENTO), o in alcuni casi direttamente `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 è progettato per essere un linguaggio `BY REFERENCE` , pertanto l'utilizzo di `BY VALUE` può presentare problemi. Ad esempio, i valori numerici letterali non hanno un tipo esplicito e la specifica COBOL non ha regole di promozione del tipo esplicito. Pertanto gli sviluppatori devono preoccuparsi della configurazione del frame di chiamata con `BY VALUE` di letterali.

Vedi http://open-cobol.sourceforge.net/faq/index.html#call per maggiori dettagli.

TEMPO DI DORMIRE

CALL è anche un modo per estendere la funzionalità COBOL e anche per consentire la riusabilità del codice. Può anche dare accesso alle funzionalità di "sistema".

Questo esempio illustra come fornire funzionalità "sleep" ai COBOL IBM Mainframe. Tieni presente che l'obbligo di farlo è raro nella misura in cui, di solito, quando qualcuno pensa di aver bisogno di "dormire" per qualche motivo, è la cosa sbagliata da fare.

ILBOWAT0 proviene dalla vecchia era di runtime COBOL specifica su mainframe. BXP1SLP e BXP4SLP sono routine di Unix System Services (USS) che possono essere utilizzate da qualsiasi lingua. Effettivamente sono richieste di "sonno" Unix.

L'attuale IBM Mainframe Runtime (Language Environment (LE)) prevede la comunicazione tra le lingue ei servizi CEE3DLY LE sono mostrati in un altro esempio, utilizzando il servizio di ritardo del thread Ambiente linguaggio z / OS .

ILBOWAT0 è in circolazione da moltissimo tempo (forse più di 40 anni) e potresti ancora incontrarlo. Il suo uso dovrebbe essere sostituito da CEE3DLY o BXP1SLP, a seconda di quale sia il più appropriato per il particolare requisito.

A volte è necessario far dormire un programma o far dormire un lavoro per un po '(dopo un passaggio FTP o NDM), che di solito vengono eseguiti come processi separati e sarà necessario eseguire il sleep / loop alla ricerca dei set di dati risultanti.

Ecco un simpatico piccolo programma COBOL per svolgere tale compito, chiamando i programmi di sospensione COBOL disponibili in OS / VS e forse in altri ambienti operativi legacy e attuali del mainframe.

```
       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     .
```

modo microfocus

Per Microfocus, utilizza l'API "SleepEx". Come esempio;

```
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.
```

Utilizzo del servizio di ritardo del thread Ambiente linguaggio z / OS

È possibile chiamare il servizio CEE3DLY in modalità 24- 31 o 64 bit per ritardare un'attività al secondo più vicino. È il salvataggio di CICS e ritarderà solo il thread.

Un esempio:

```
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.
```

Puoi vedere più dettagli qui:

IBM Language Environment Callable Services - Sleep

Leggi Dichiarazione di CHIAMATA online: https://riptutorial.com/it/cobol/topic/5601/dichiarazione-di-chiamata

Capitolo 13: Dichiarazione di COMMIT

Osservazioni

Scarica TUTTI i blocchi correnti, sincronizzando i buffer I / O dei file.

Questa è un'estensione non standard, disponibile con alcune implementazioni COBOL che supportano le funzionalità di `ROLLBACK` .

Examples

Dichiarazione di COMMIT

```
WRITE record
COMMIT
```

Leggi Dichiarazione di COMMIT online: https://riptutorial.com/it/cobol/topic/6357/dichiarazione-di-commit

Capitolo 14: Dichiarazione di PERFORM

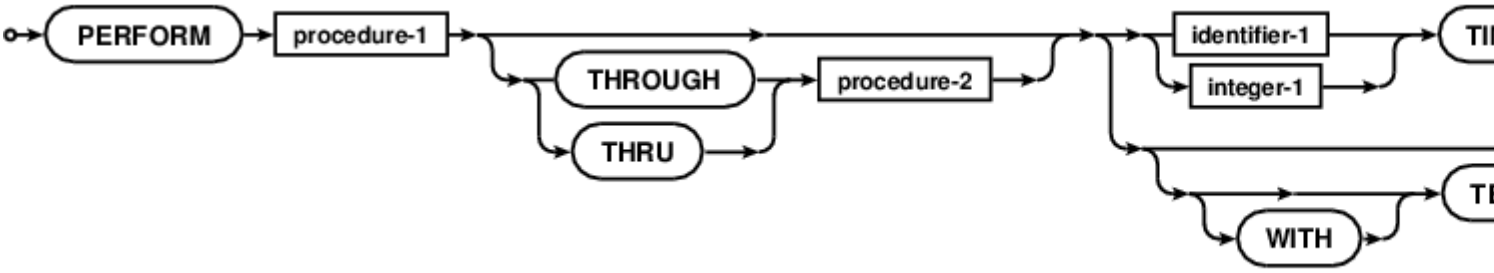
Osservazioni

L'istruzione PERFORM trasferisce il controllo a una o più procedure e restituisce il controllo implicitamente al termine della sequenza. PERFORM può essere utilizzato anche per loop in linea nell'ambito della PERFORM.

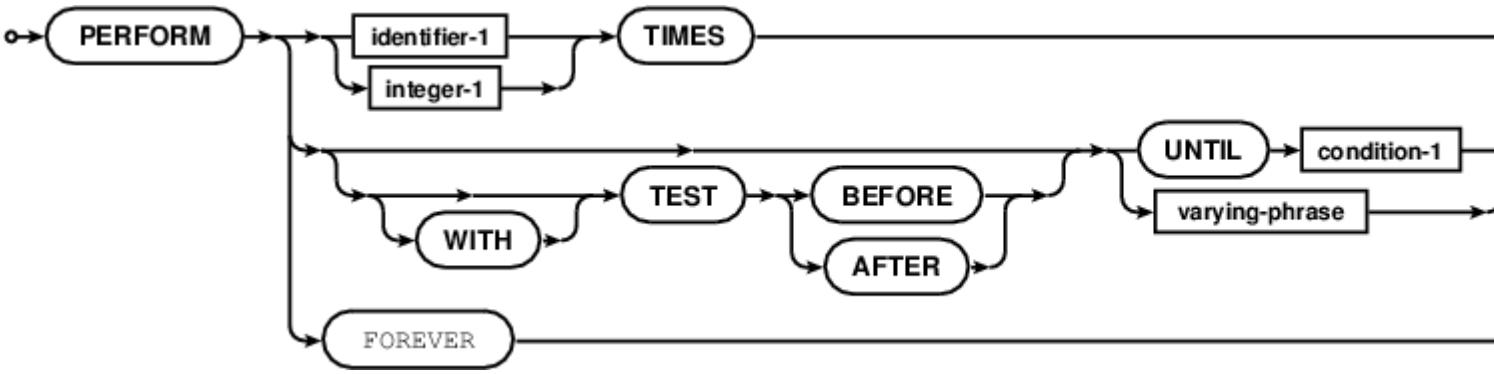
La frase `VARYING` consente l'annidamento con una o più clausole `AFTER` e il test condizionale può essere `BEFORE` (predefinito) o `AFTER` ogni ciclo.

La clausola `THRU` un'esecuzione procedurale presuppone un controllo sequenziale dall'alto verso il basso dalla `procedure-1` alla fine della `procedure-2` . *THRU* è un problema polemico e molti programmatori preferiscono `PERFORM` per `SECTION` piuttosto che usare i paragrafi `THRU` . Alcuni negozi possono imporre `PERFORM THRU` con un paragrafo di punto di uscita esplicito, altri possono vietare l'uso di `THRU` trovando più difficile eseguire il debug.

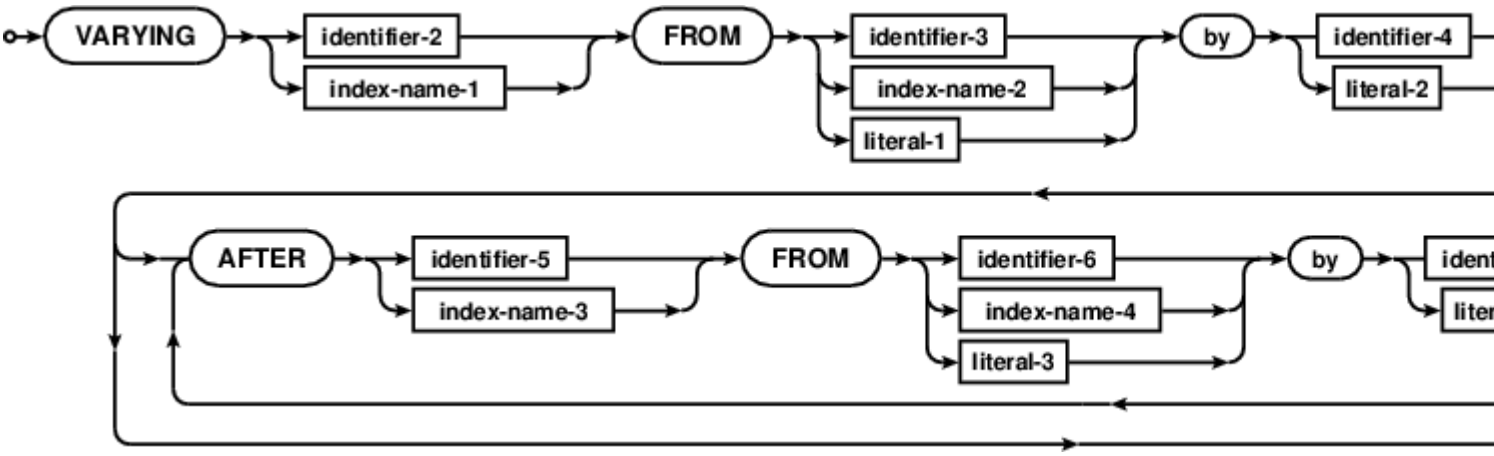
Esecuzione procedurale:



Esecuzione in linea:



Dove la `varying-phrase` è:



Examples

In linea PERFORMARE VARIAZIONE

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

PERFORM PROCESSO

```
PERFORM some-paragraph
```

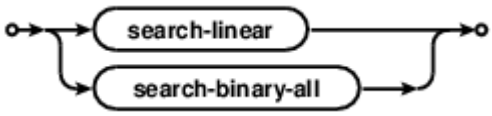
Leggi Dichiarazione di PERFORM online: https://riptutorial.com/it/cobol/topic/7334/dichiarazione-di-perform

Capitolo 15: Dichiarazione di ricerca

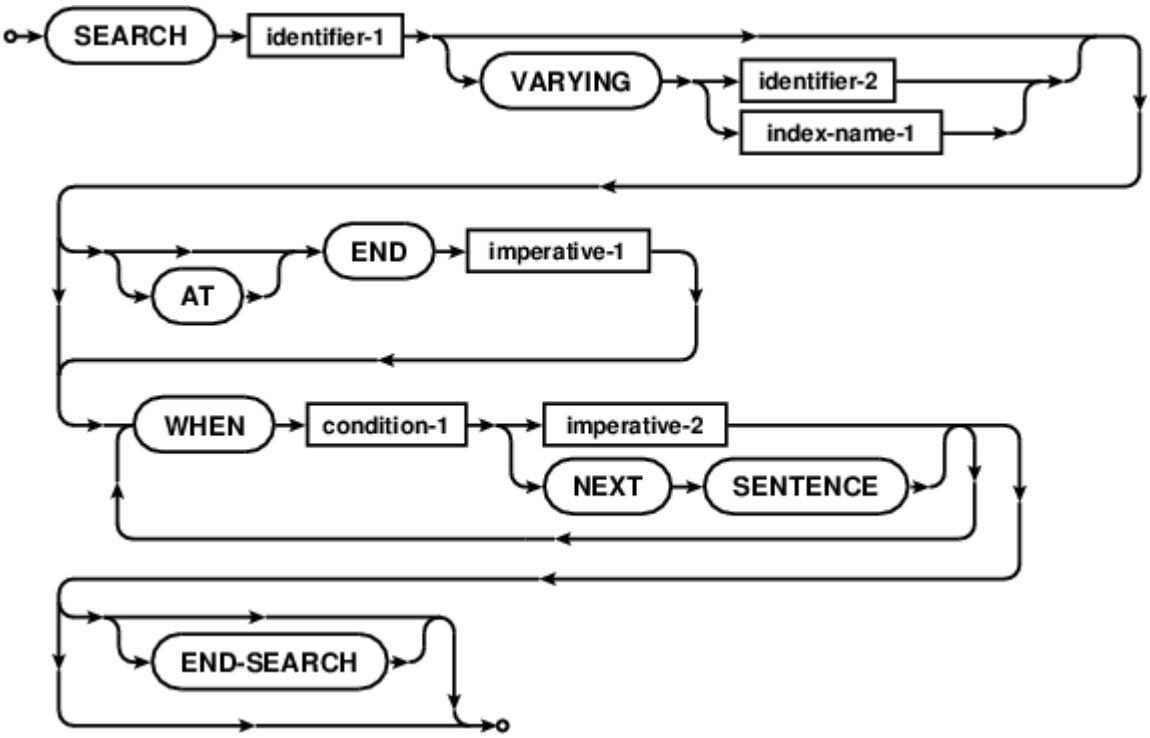
Osservazioni

L'istruzione COBOL `SEARCH` due forme. `SEARCH` e un algoritmo binario `SEARCH ALL` lineare. Binary SEARCH ALL presuppone una tabella ordinata adatta per una ricerca binaria senza elementi fuori ordine.

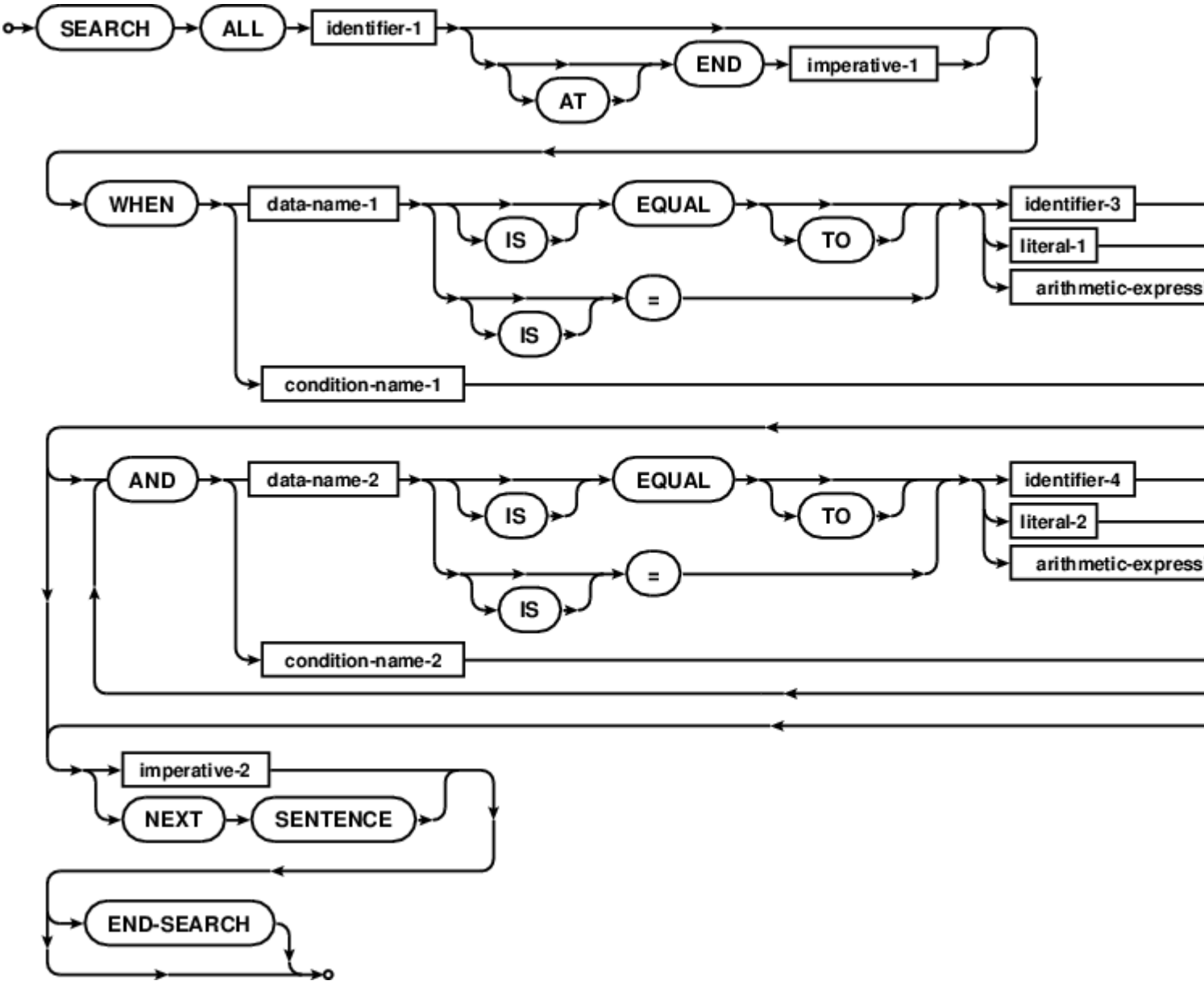
Dichiarazione di ricerca



RICERCA lineare



Binario CERCA TUTTO



Examples

RICERCA lineare

```
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 CERCA TUTTO

```
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
```

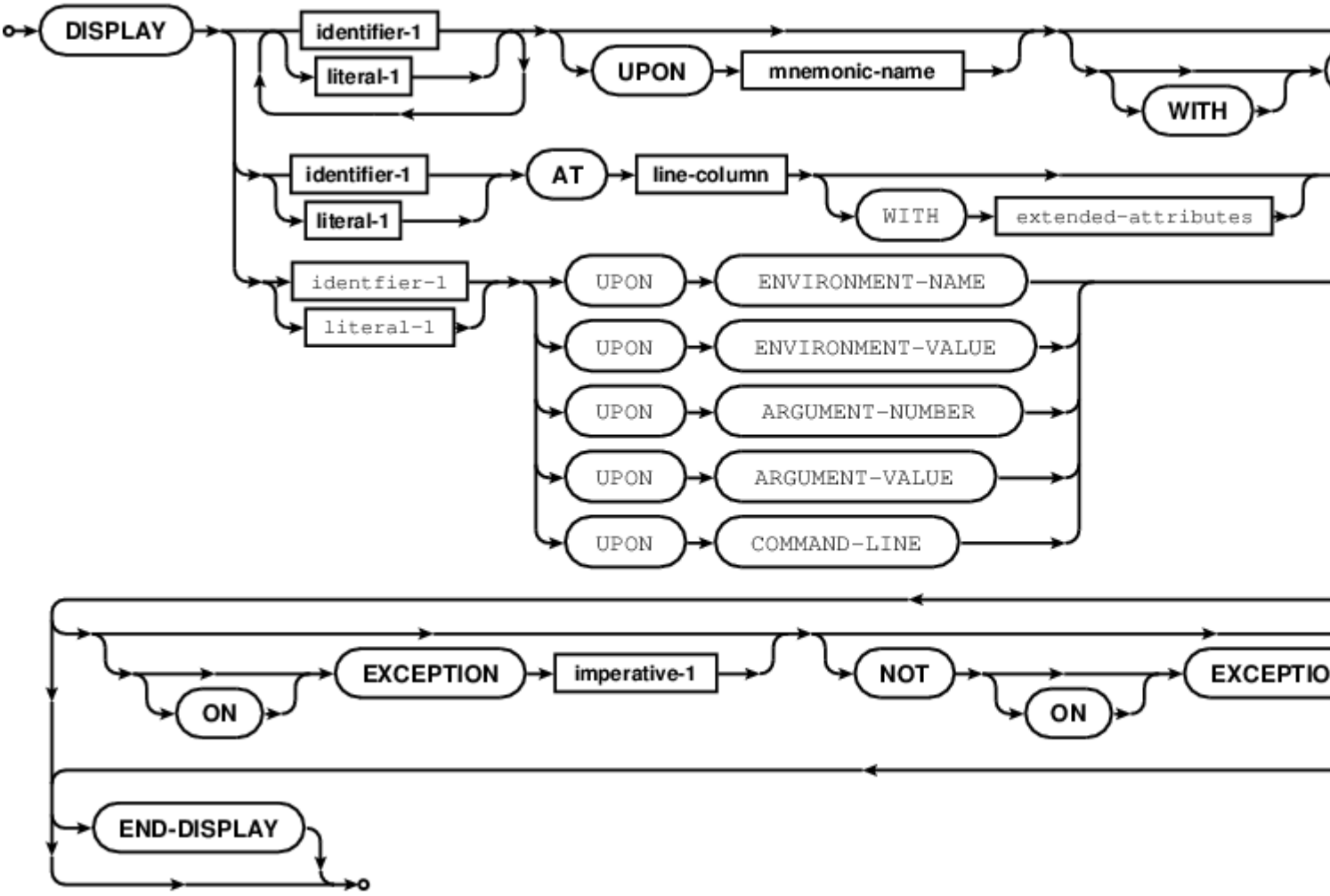
```
       read wordfile
           at end set at-eof to true
       end-read

       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.
```

Leggi Dichiarazione di ricerca online: https://riptutorial.com/it/cobol/topic/7462/dichiarazione-di-ricerca

Capitolo 16: Dichiarazione DISPLAY

Osservazioni

L'istruzione `DISPLAY` fa sì che i dati vengano trasferiti all'hardware o al software dell'ambiente operativo. `DISPLAY` disponibile in due forme, `UPON device` o per la visualizzazione di dati `SCREEN` . Le variabili di ambiente possono anche essere impostate con `DISPLAY UPON` in alcune implementazioni di COBOL, insieme ad altre estensioni per il trasferimento di dati di grafica o altre esigenze specifiche del dispositivo.



Examples

VISUALIZZA SU

```
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
```

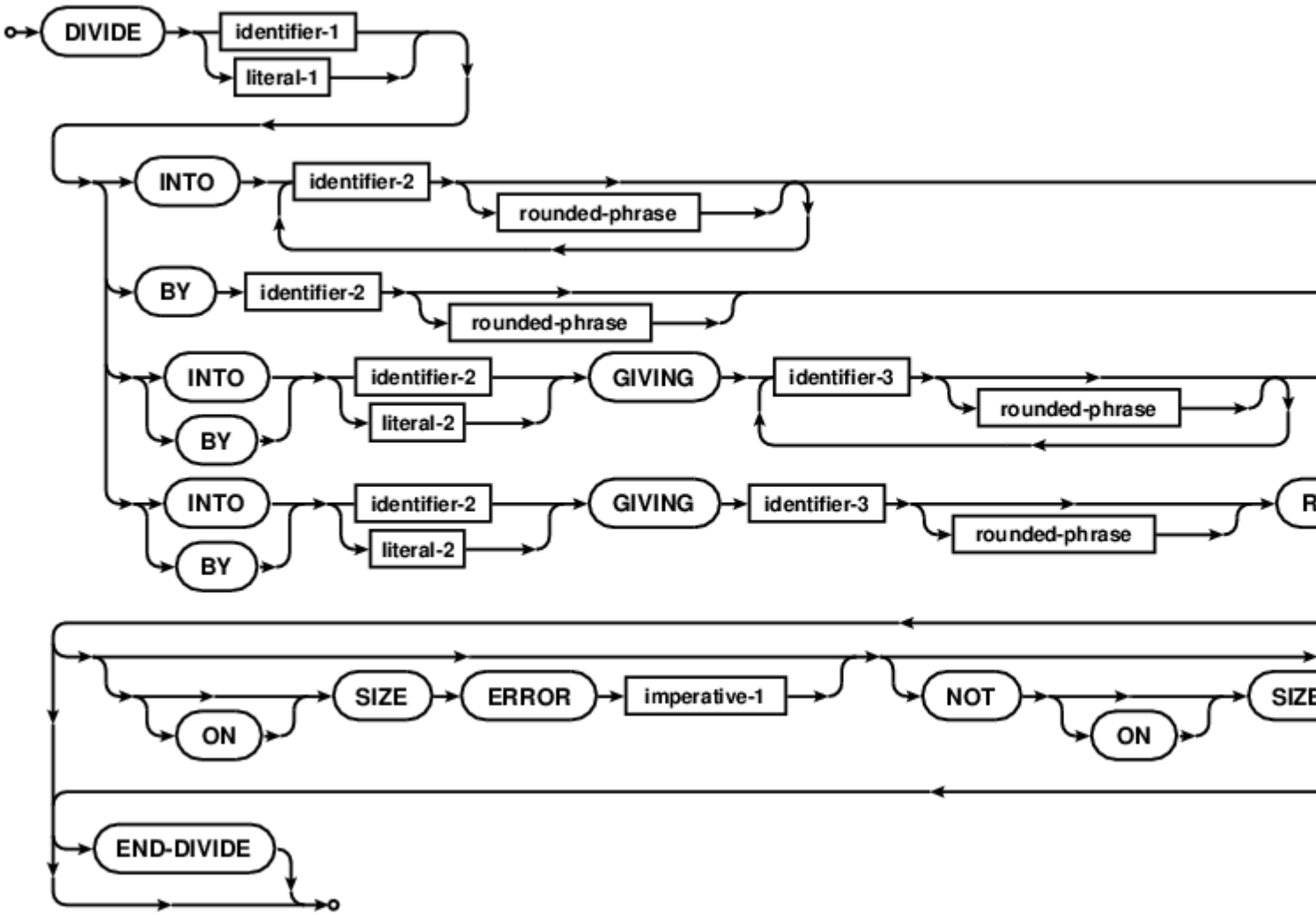
UPON CONSOLE è un valore predefinito, raramente scritto. I messaggi con DISPLAY sono un modo per eseguire il debug del codice COBOL, ma molti programmi COBOL sono di natura transazionale e potrebbero non interagire mai con un operatore umano una volta inviato un lavoro.

Leggi Dichiarazione DISPLAY online: https://riptutorial.com/it/cobol/topic/7082/dichiarazione-display

Capitolo 17: Dichiarazione DIVIDE

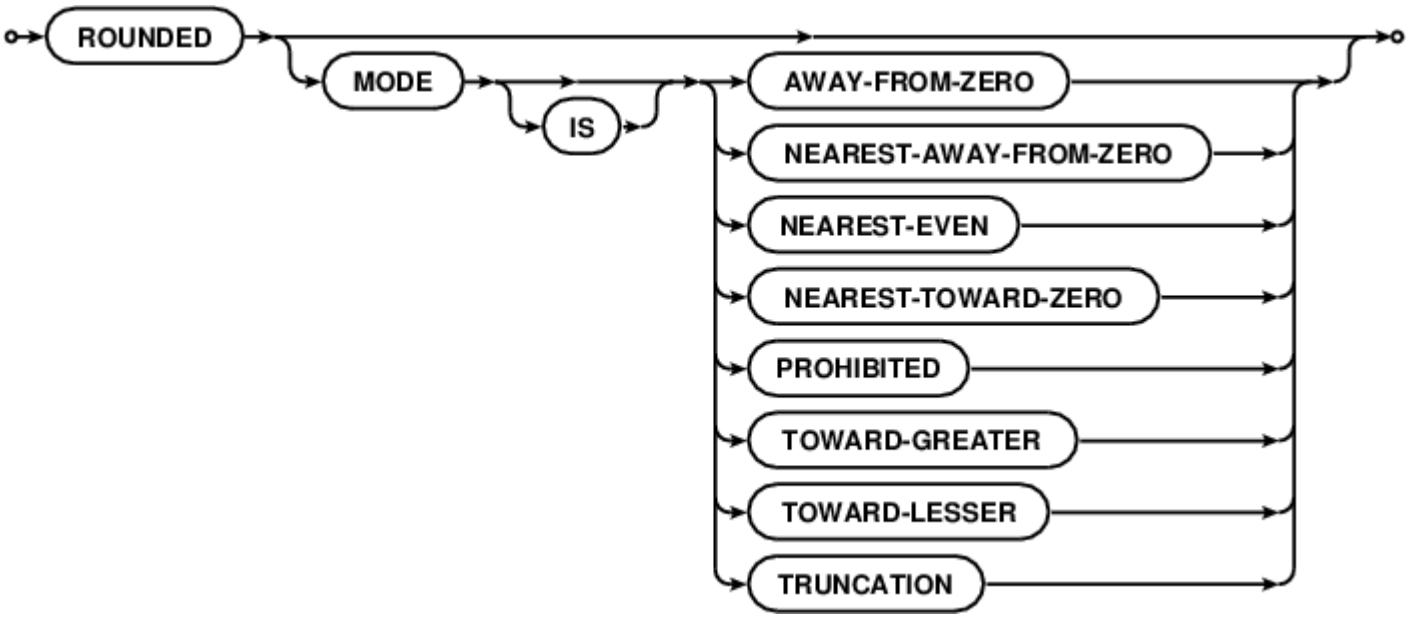
Osservazioni

L'istruzione COBOL `DIVIDE` divide un elemento numerico in altri che impostano gli elementi di dati nel quoziente e, facoltativamente, il resto.



Frase `ROUNDED` :

L'impostazione predefinita è `TRUNCATION` se non è `TRUNCATION` specificata una frase arrotondata. La modalità `ROUNDED` predefinita è `NEAREST-TOWARD-ZERO` (arrotondando per difetto) a meno che non sia specificato diversamente. Il cosiddetto *arrotondamento del Banker* è `NEAREST-EVEN` .



Examples

Formati di istruzioni DIVIDE

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

I dati `b` , `c` , e `d` sono cambiati come `b/a` , `c/a` `d/a` .

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

L'elemento dati `c` è cambiato come `b/a` .

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

L'elemento di dati `c` è cambiato come `a/b` .

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

Gli elementi di dati `q` e `r` sono impostati con i risultati di `b/a`

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

Gli elementi di dati `q` e `r` sono impostati con i risultati di `b/a`

Tutti i campi dei risultati `DIVIDE` possono avere clausole `ROUNDED MODE IS` .

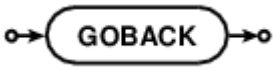
Tutte le istruzioni `DIVIDE` possono avere `ON SIZE ERROR` e `NOT ON SIZE ERROR` dichiarazioni dichiarative incluse per rilevare risultati non validi, dati il tipo e la dimensione dei campi dei risultati.

Leggi Dichiarazione DIVIDE online: https://riptutorial.com/it/cobol/topic/7081/dichiarazione-divide

Capitolo 18: Dichiarazione GOBACK

Osservazioni

L'istruzione COBOL `GOBACK` è un ritorno. A differenza di `EXIT PROGRAM` o `STOP RUN` , `GOBACK` restituisce sempre un livello. Se il modulo corrente è "principale", `GOBACK` ritornerà al sistema operativo. Se il modulo corrente è un sottoprogramma, `GOBACK` ritornerà all'istruzione dopo una chiamata.



Examples

TORNA INDIETRO

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

...

call "subprog"
goback.
```

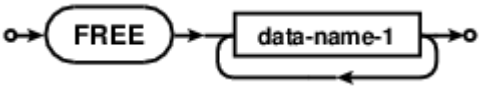
Il primo `GOBACK` sopra ritornerà da sottoprog. Supponendo che il secondo sia all'interno della procedura principale, `GOBACK` ritornerà al sistema operativo.

Leggi Dichiarazione GOBACK online: https://riptutorial.com/it/cobol/topic/7173/dichiarazione-goback

Capitolo 19: Dichiarazione GRATUITA

Osservazioni

L'istruzione `FREE` libera memoria allocata per uno o più identificatori, tramite POINTER o da un identificativo di memoria di lavoro BASED. L'uso dopo FREE è illegale.



Examples

GRATIS un'assegnazione

```
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
```

Leggi Dichiarazione GRATUITA online: https://riptutorial.com/it/cobol/topic/7162/dichiarazione-gratuita

Capitolo 20: Dichiarazione IF

Osservazioni

L'espressione condizionale e la dichiarazione di selezione. Si consiglia l'utilizzo di terminatori di ambito espliciti. Le espressioni condizionali COBOL consentono lo shortform, in cui l'identificatore corrente (e condizionale) viene assunto attraverso test di condizioni multiple, a meno che non venga esplicitamente indicato.

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

è equivalente a

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

Examples

IF con condizionali di forma breve

```
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
```

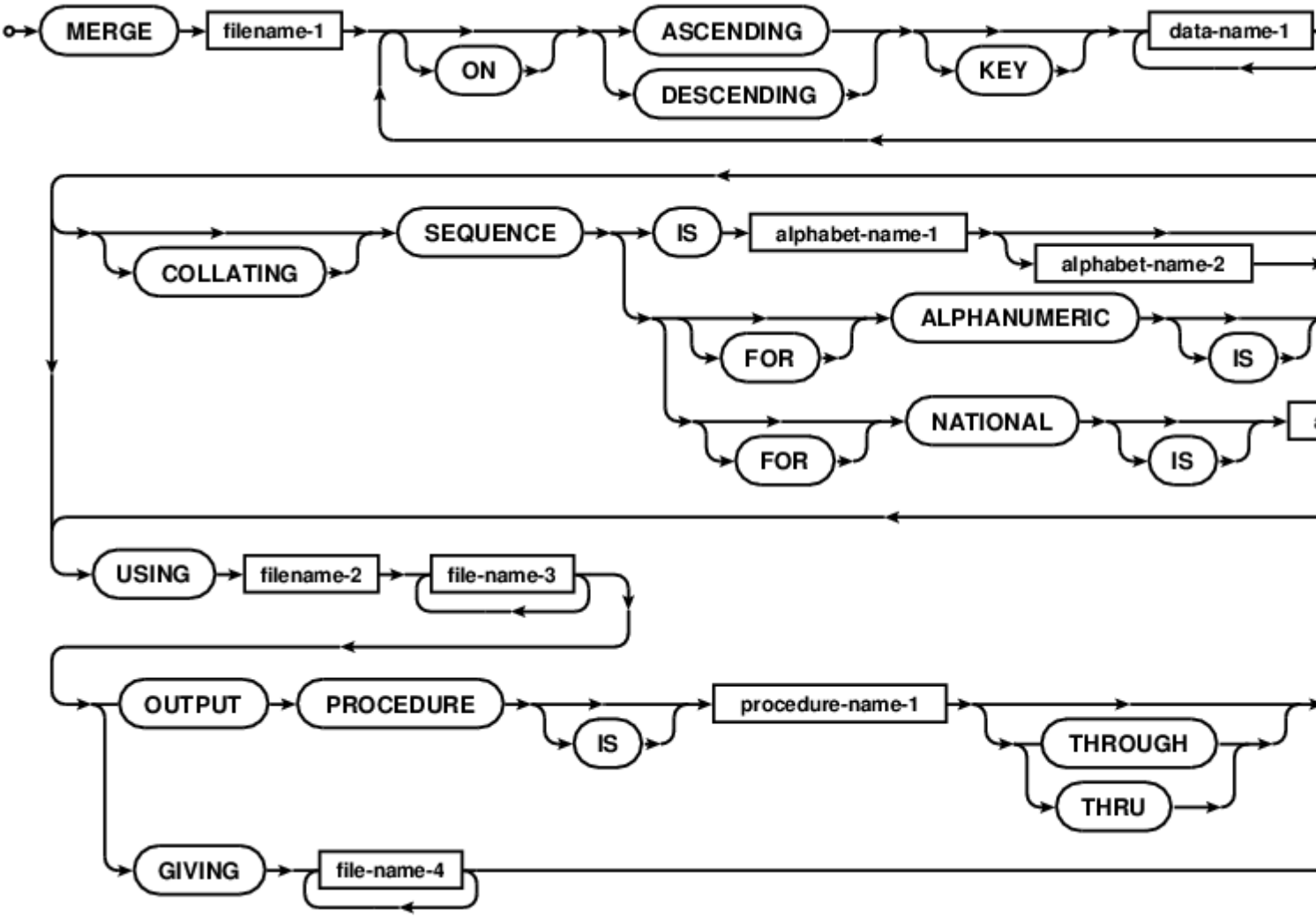
`IF` istruzioni `IF` possono essere terminate con terminazione completa o terminatore di ambito esplicito `END-IF` . *Non è più raccomandato l'uso di periodi per la chiusura dello scope.* Gli arresti completi significano solo che nel caso dell'IF annidato, tutti gli annidamenti vengono interrotti al primo punto `.` e qualsiasi codice successivo sarà al di fuori del blocco IF.

Leggi Dichiarazione IF online: https://riptutorial.com/it/cobol/topic/7174/dichiarazione-if

Capitolo 21: Dichiarazione MERGE

Osservazioni

L'istruzione MERGE unirà uno o più file di dati COBOL formattati come in un singolo file di output. Il programmatore può assumere il controllo della `OUTPUT PROCEDURE` , che utilizza l'istruzione `RELEASE` o utilizza meccanismi di runtime COBOL interni con la clausola `GIVING` .



Examples

MERGE i dati regionali in master

```
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.
```

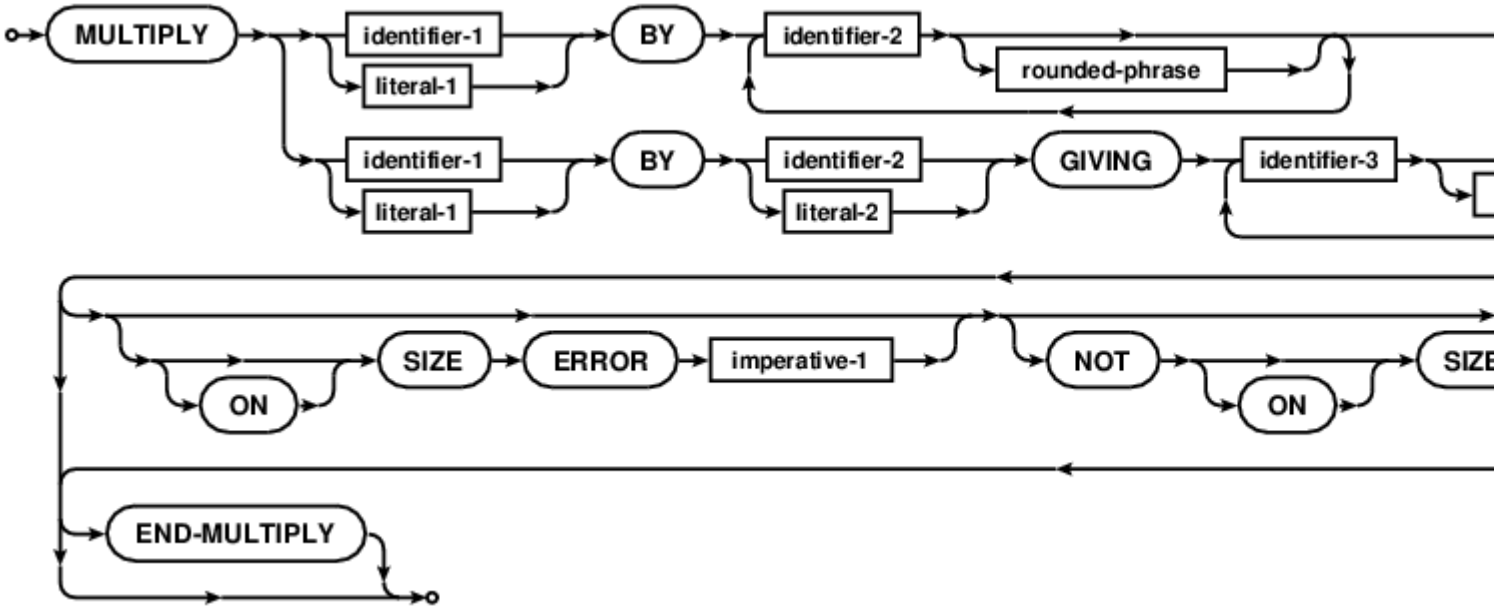
```
       end program gnucobol-merge-sample.
```

Leggi Dichiarazione MERGE online: https://riptutorial.com/it/cobol/topic/7183/dichiarazione-merge

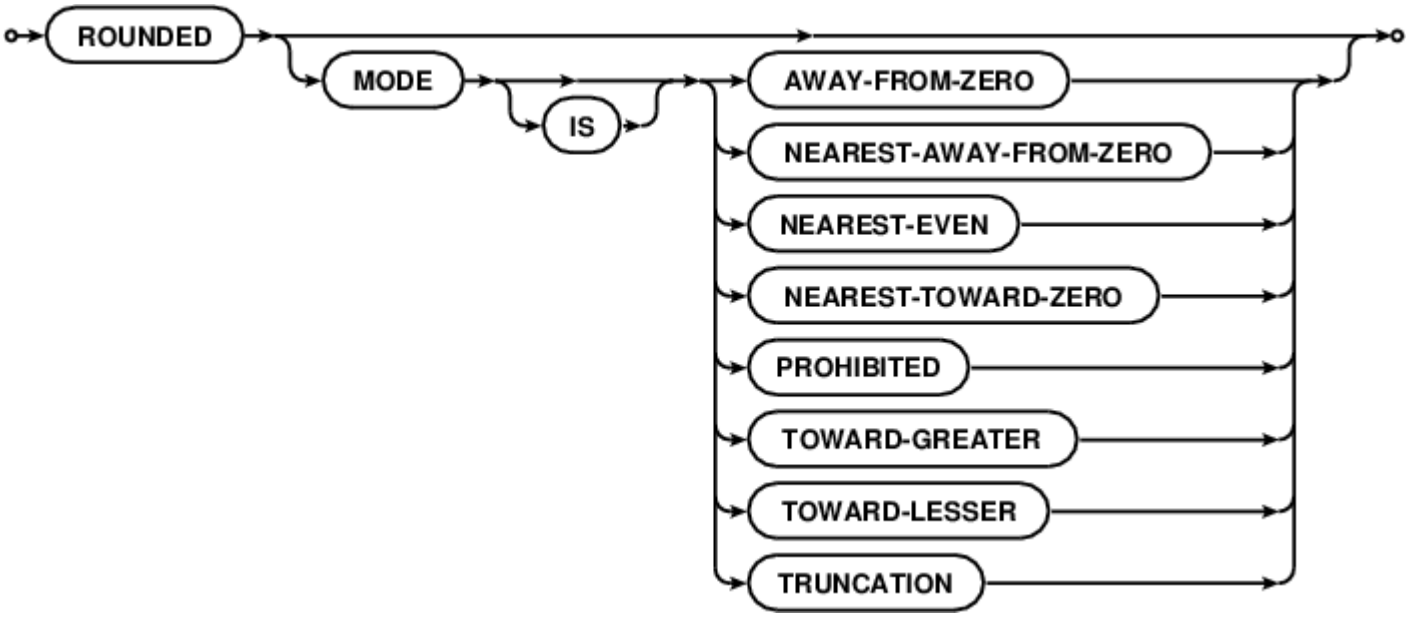
Capitolo 22: Dichiarazione MULTIPLY

Osservazioni

L'istruzione `MULTIPLY` moltiplica i dati numerici impostando il risultato su uno o più identificatori di tipo numerico.



Dove è `rounded-phrase`



Examples

Alcuni formati MULTIPLY

```
MULTIPLY 5 BY a
```

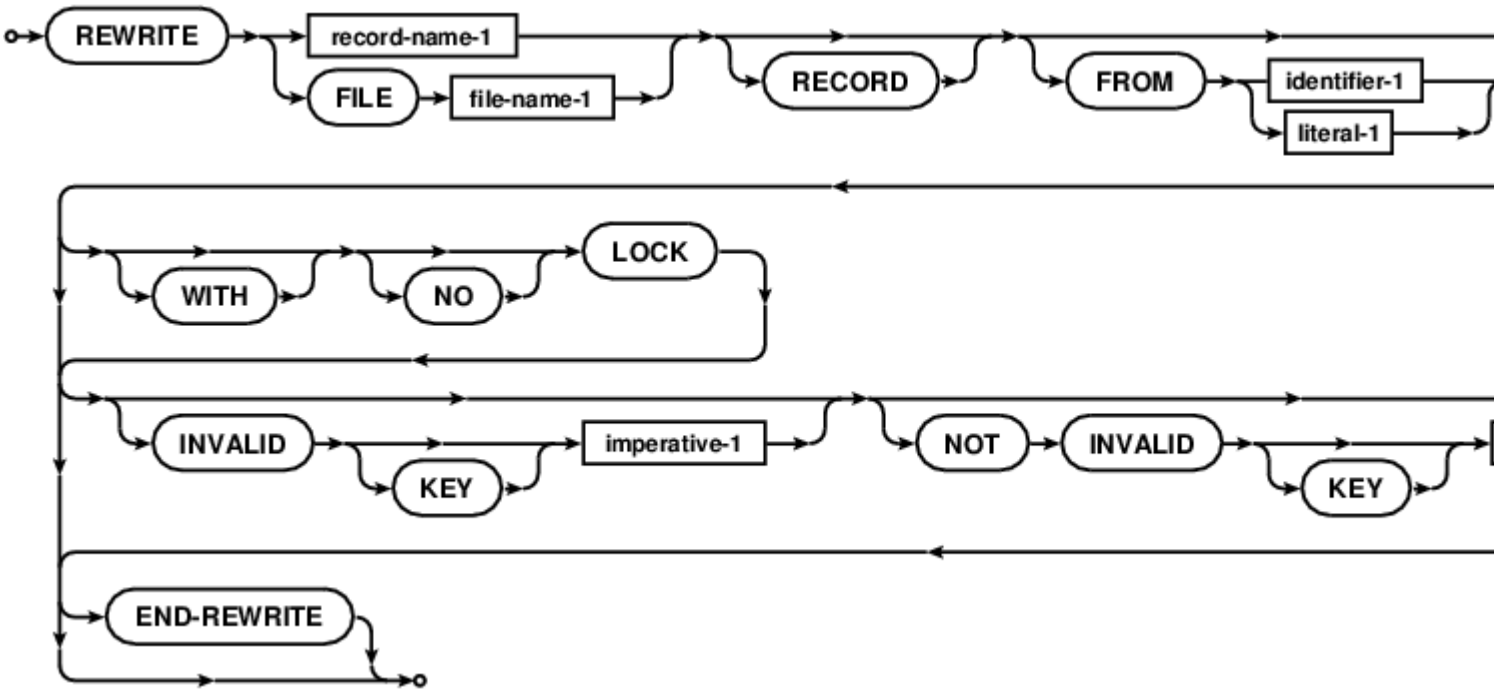
```
MULTIPLY a BY b
    ON SIZE ERROR
        PERFORM error-handling
    NOT ON SIZE ERROR
        PERFORM who-does-that
END-MULTIPLY

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

Leggi Dichiarazione MULTIPLY online: https://riptutorial.com/it/cobol/topic/7264/dichiarazione-multiply

Capitolo 23: Dichiarazione REWRITE

Osservazioni

L'istruzione REWRITE sostituisce logicamente i record esistenti sull'archiviazione di massa.



Examples

REWRITE dei record in un file di accesso 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.
```

Leggi Dichiarazione REWRITE online: https://riptutorial.com/it/cobol/topic/7460/dichiarazione-rewrite

Capitolo 24: Dichiarazione SORT

Osservazioni

L'istruzione COBOL `SORT` può essere utilizzata per ordinare file e tabelle nella memoria di lavoro.

File SORT

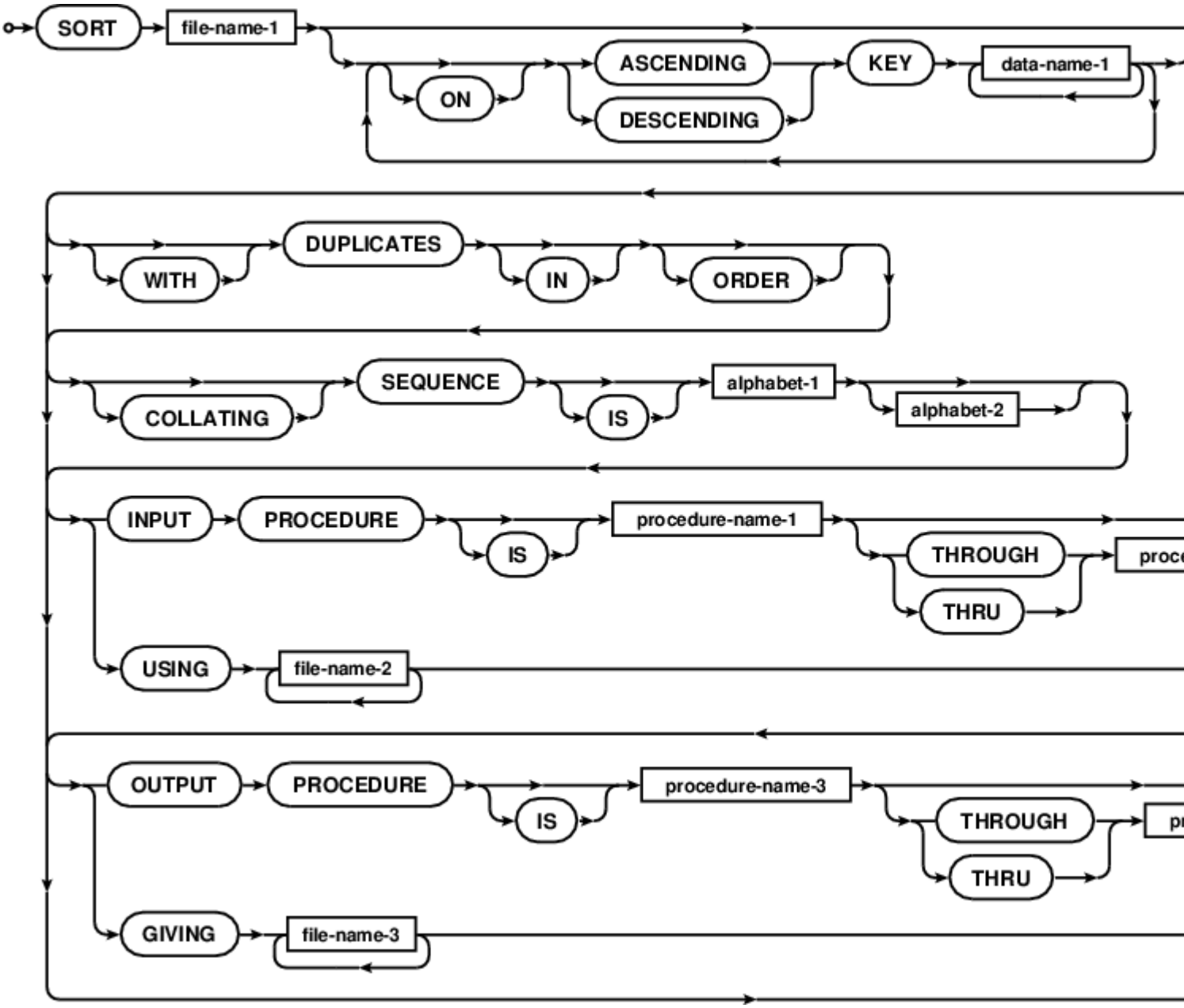
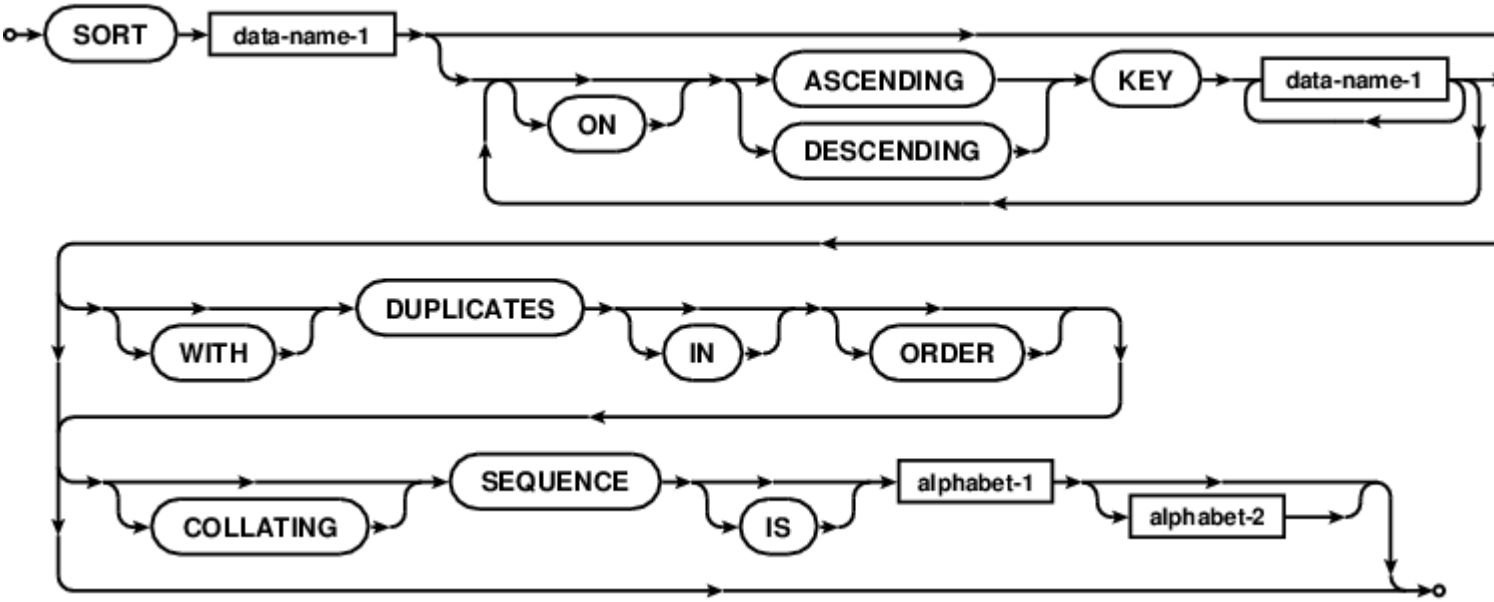


Tabella SORT



Examples

Ordinamento standard in standard out

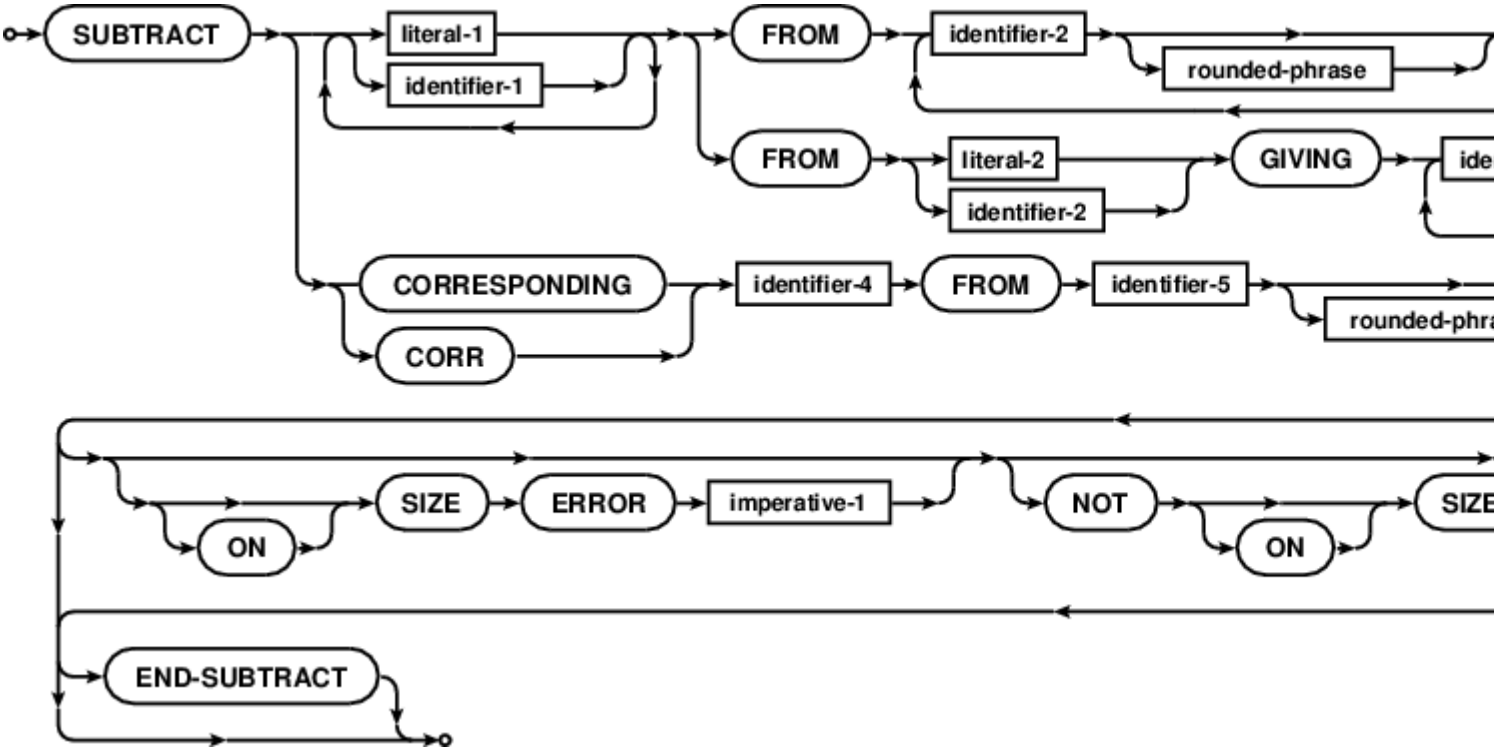
```
GCobol* GnuCOBOL SORT verb example using standard in and standard out
       identification division.
       program-id. sorting.

       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.
```

Leggi Dichiarazione SORT online: https://riptutorial.com/it/cobol/topic/7463/dichiarazione-sort

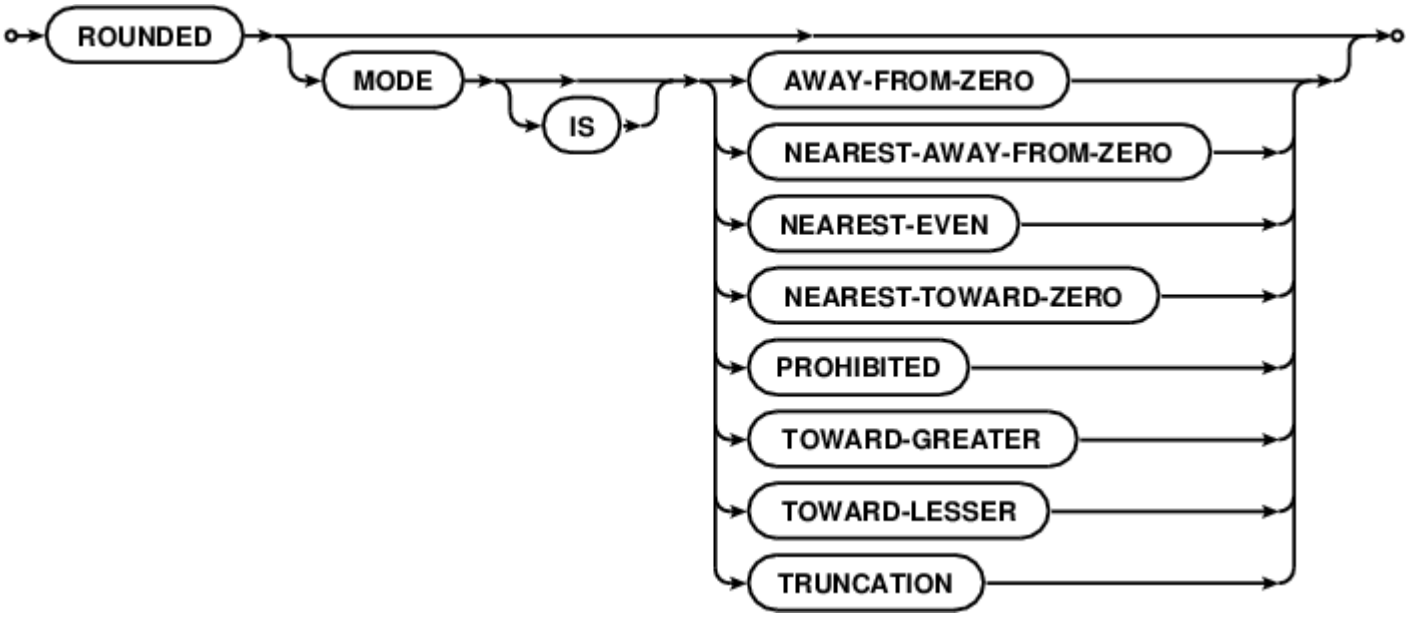
Capitolo 25: Dichiarazione SOTTRA

Osservazioni

L'istruzione `SUBTRACT` viene utilizzata per sottrarre uno, o la somma di due o più elementi di dati numerici da uno o più elementi e impostare i valori di uno o più identificatori sul risultato.



arrotondato Frasi



Examples

Esempio di SOTTRAZIONE

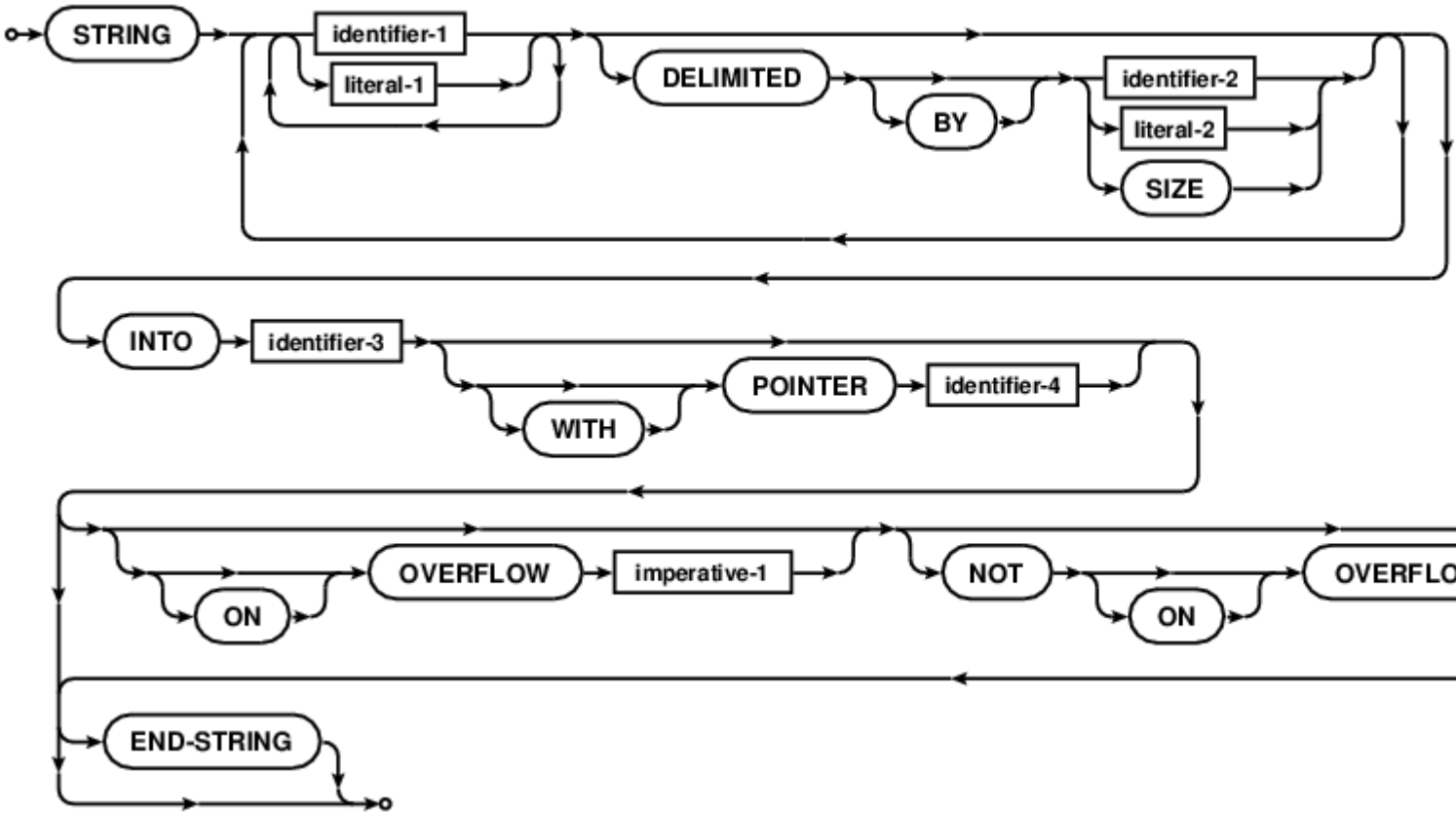
```
SUBTRACT item-a item-b item-c FROM account-z ROUNDED MODE IS NEAREST-EVEN
    ON SIZE ERROR
        DISPLAY "CALL THE BOSS, Account `Z` is OUT OF MONEY" END-DISPLAY
        PERFORM promisary-processing
    NOT ON SIZE ERROR
        PERFORM normal-processing
END-SUBTRACT
```

Leggi Dichiarazione SOTTRA online: https://riptutorial.com/it/cobol/topic/7465/dichiarazione-sottra

Capitolo 26: Dichiarazione STRING

Osservazioni

L'istruzione `STRING` concatena i contenuti parziali o completi di più campi in un singolo risultato.



Examples

Esempio STRING per stringhe C.

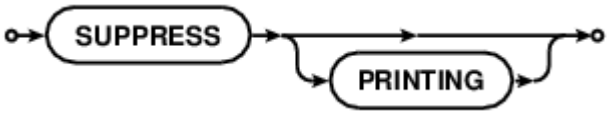
```
*> Strip off trailing zero bytes
STRING c-string DELIMITED BY LOW-VALUE INTO working-store
```

Leggi Dichiarazione STRING online: https://riptutorial.com/it/cobol/topic/7468/dichiarazione-string

Capitolo 27: Dichiarazione SUPPRESS

Osservazioni

L'istruzione `SUPPRESS` inibisce la stampa di un gruppo di report. Funzione Scrittura report COBOL.



Examples

Esempio SUPPRESS

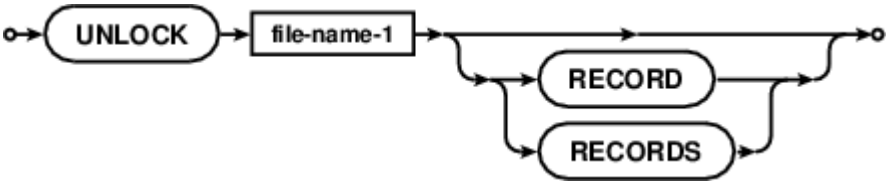
```
SUPPRESS PRINTING
```

Leggi Dichiarazione SUPPRESS online: https://riptutorial.com/it/cobol/topic/7470/dichiarazione-suppress

Capitolo 28: Dichiarazione UNLOCK

Osservazioni

L'istruzione `UNLOCK` rilascia esplicitamente i blocchi di record associati a un connettore di file.



Examples

UNLOCK record da un connettore di file

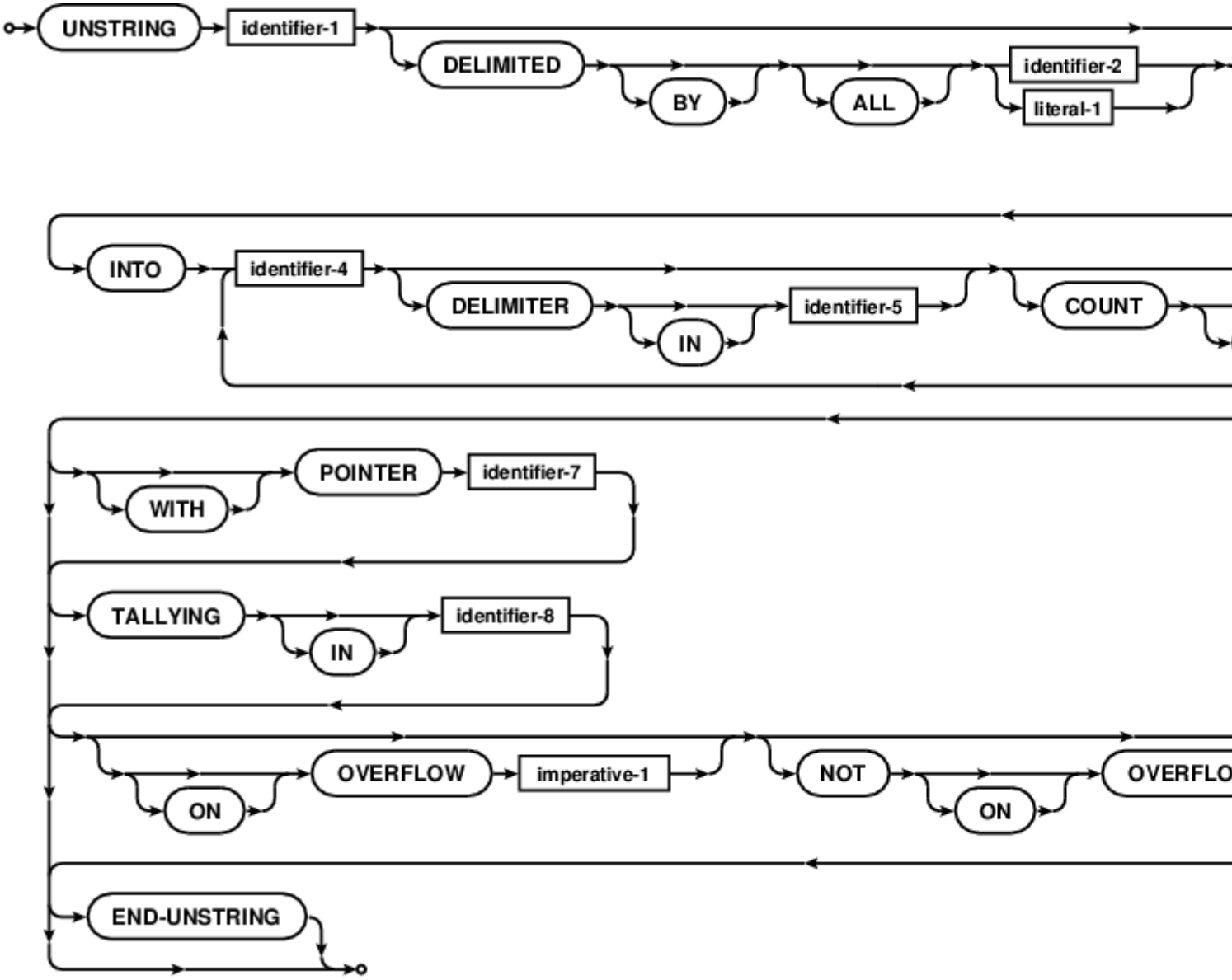
```
UNLOCK filename-1 RECORDS
```

Leggi Dichiarazione UNLOCK online: https://riptutorial.com/it/cobol/topic/7471/dichiarazione-unlock

Capitolo 29: Dichiarazione UNSTRING

Osservazioni

L'istruzione `UNSTRING` separa un campo di invio e posiziona i risultati in più campi di ricezione.



Examples

Esempio 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
```

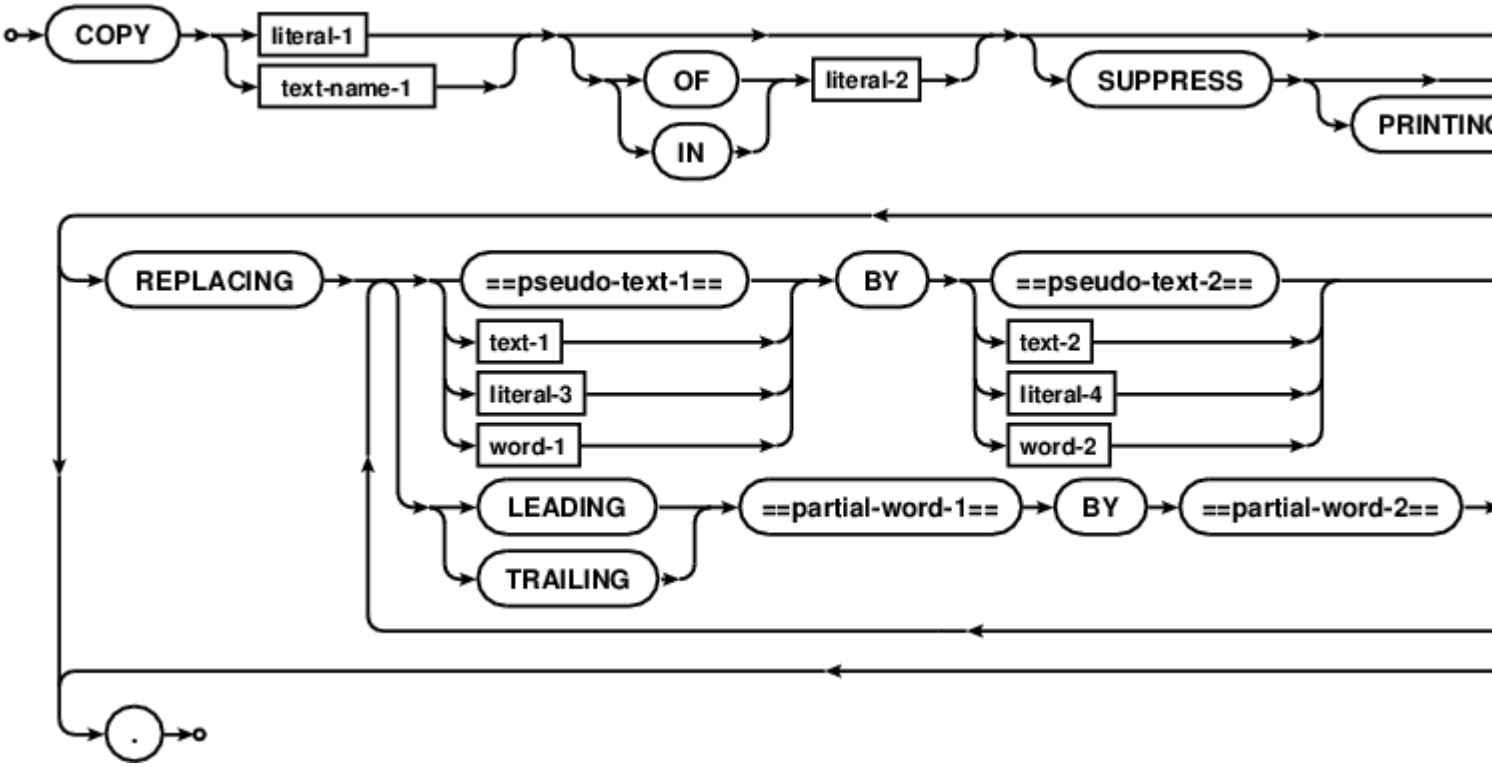
Leggi Dichiarazione UNSTRING online: https://riptutorial.com/it/cobol/topic/7581/dichiarazione-unstring

Capitolo 30: Direttiva COPY

Osservazioni

La versione COBOL della direttiva C `#include` preprocessore. *O, più storicamente accurato, il COBOL è arrivato per primo, sviluppato circa 10 anni prima.*

A causa di alcune delle decisioni di progettazione in COBOL (nessun argomento per `PERFORM` come ragione principale), molte sequenze di accesso alla struttura dei dati devono rompere il principio di DRY . I nomi dei componenti della struttura devono essere ripetuti nella DIVISIONE AMBIENTE, nella DIVISIONE DATI e possibilmente in più volte nella DIVISIONE PROCEDURA. Questo di solito viene gestito aggiungendo quaderni. Le dichiarazioni di registrazione e il codice di accesso sono nascosti in file separati e l'istruzione COPY è l'unica fonte ripetuta. Una modifica al quaderno mantiene sincronizzati tutti gli usi dell'ortografia dei nomi e del layout dei dati, invece di richiedere più modifiche a più file quando si verifica una modifica.



Examples

Copia il layout del record.

programma-one.

```
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.
```

programma-due.

```
   DATA DIVISION.

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

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

Leggi Direttiva COPY online: https://riptutorial.com/it/cobol/topic/6982/direttiva-copy

Capitolo 31: Divisione dati

introduzione

DATA DIVISION è una delle quattro parti che compongono un programma COBOL. Contiene dichiarazioni che descrivono i dati utilizzati dal programma. Si compone di quattro sezioni: SEZIONE FILE, SEZIONE LAVORAZIONE-STOCCAGGIO, SEZIONE LOCALE-STOCCAGGIO e SEZIONE COLLEGAMENTO.

Examples

Sezioni nella divisione dati

Le SEZIONI in COBOL possono essere richieste o facoltative, a seconda della DIVISIONE in cui si trovano.

```
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).
```

Nell'esempio sopra, 01 sono numeri di livello.

Numero di livello

Il numero del livello viene utilizzato per specificare il livello dei dati in un record. Sono usati per distinguere tra elementi elementari e oggetti di gruppo. Gli elementi elementari possono essere raggruppati per creare elementi di gruppo.

- 01: registra la voce di descrizione. Il numero del livello di gruppo è sempre 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: oggetti elementari

- 66: Rinomina articoli
- 77: elementi che non possono essere suddivisi.
- 88: Il livello 88 è un numero di livello speciale utilizzato per migliorare la leggibilità dei programmi COBOL e per migliorare i test IF. Un livello 88 sembra un livello sotto un'altra variabile, ma non lo è. Non ha una IMMAGINE, ma ha un valore. Un livello 88 è sempre associato ad un'altra variabile ed è un nome di condizione per quella variabile.

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

Entrambe le seguenti condizioni verificano se YES-NO è uguale a "Y":

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

Un nome di condizione di livello 88 può essere utilizzato per una variabile alfanumerica o numerica.

Foto

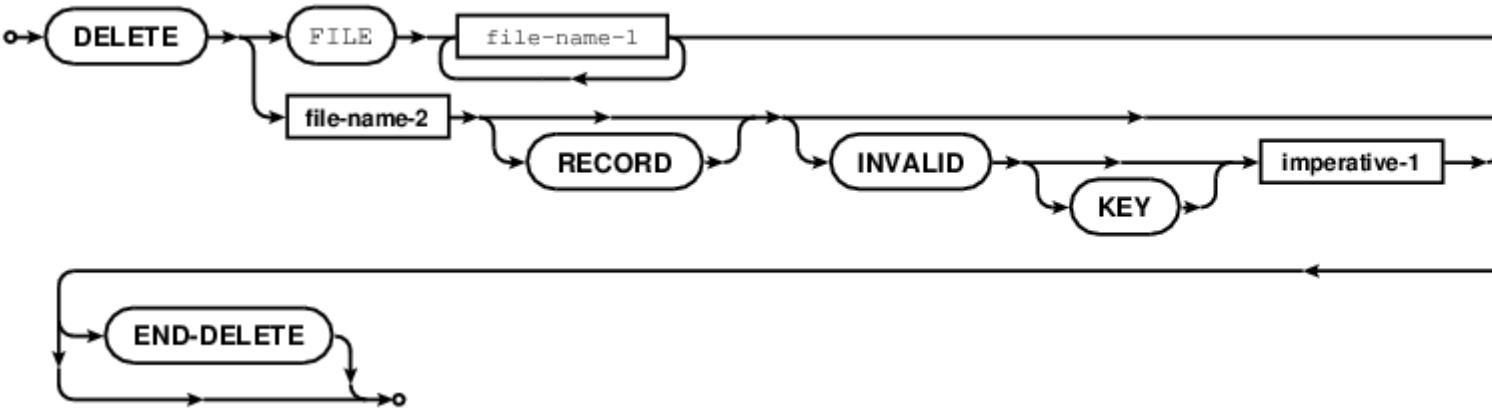
La CLIÙ IMMAGINE definisce due cose su una variabile: la dimensione della variabile (il numero di byte utilizzati nella memoria per il valore) e il tipo di dati che possono essere memorizzati nella variabile.

Leggi Divisione dati online: https://riptutorial.com/it/cobol/topic/10859/divisione-dati

Capitolo 32: ELIMINA la dichiarazione

Osservazioni

L'istruzione `DELETE` cancella i record dalla memoria di massa. Alcuni compilatori consentono di utilizzare l'istruzione `DELETE` con una clausola `FILE` , per eliminare i nomi `FD` (insieme alle strutture di indicizzazione associate che potrebbero essere richieste dal motore di gestione del database in uso).



Examples

Elimina un record, digita il campo chiave primaria

```
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

...
```

Leggi ELIMINA la dichiarazione online: https://riptutorial.com/it/cobol/topic/7063/elimina-la-dichiarazione

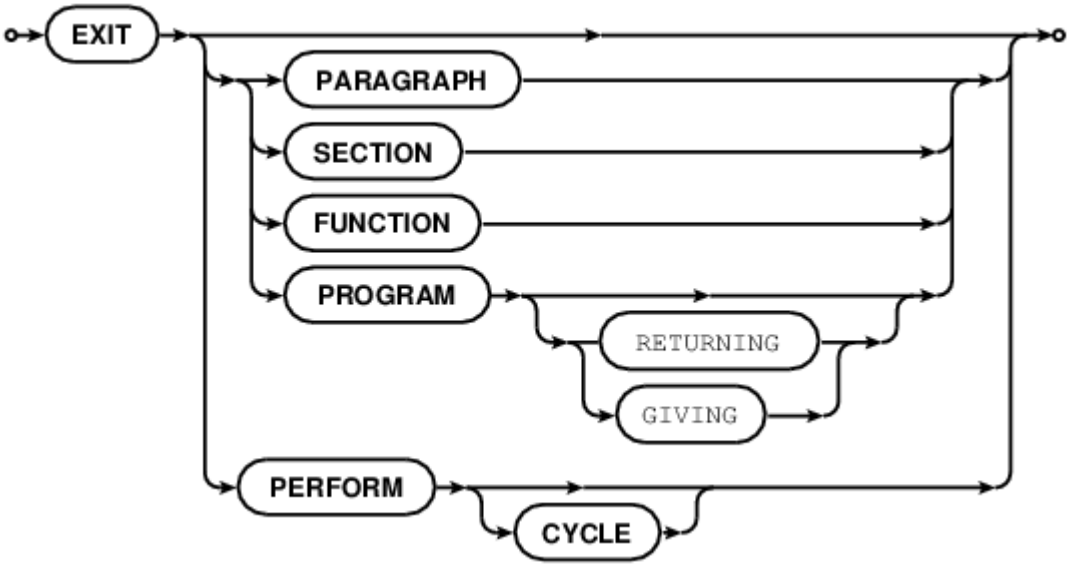
Capitolo 33: EXIT statement

Osservazioni

L'istruzione COBOL `EXIT` è un verbo di controllo del flusso terminante.

`EXIT` è di pochi gusti:

- nuda `EXIT` è un punto finale comune per una serie di procedure.
- `EXIT PARAGRAPH` , `EXIT SECTION` consente di uscire da una procedura strutturata senza eseguire alcuna delle dichiarazioni successive.
- `EXIT FUNCTION` , `EXIT METHOD` , `EXIT PROGRAM` segna la fine logica di un modulo di codice.
- `EXIT PERFORM` interrompe da un ciclo di esecuzione in linea.
- `EXIT PERFORM CYCLE` fa in modo che un ciclo di esecuzione inline inizi la successiva iterazione.



Examples

EXIT statement

```
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
```

Leggi EXIT statement online: https://riptutorial.com/it/cobol/topic/7084/exit-statement

Capitolo 34: Funzioni intrinseche

introduzione

Le funzioni intrinseche sono incluse nello standard COBOL come un insieme di funzioni che restituiscono valori da un algoritmo specifico, dato zero o più argomenti. Queste funzioni intrinseche sono fornite come funzionalità del compilatore e del sistema di runtime. Gli articoli di restituzione sono campi COBOL temporanei e possono essere dati di carattere, campi di bit o valori numerici.

Gli esempi includono funzioni trigonometriche, routine di data e ora, conversioni di tipi di dati, deviazione standard e altri algoritmi di supporto.

Osservazioni

COBOL 2014 elenca le seguenti funzioni intrinseche standard:

```
======================================== ==========
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
======================================= ==========
```

Aggiunge GnuCOBOL

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

La parola chiave `FUNCTION` è richiesta a meno che non includa l'opzione source (o compile time)

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

Dove `ALL INTRINSIC` può essere un elenco di funzioni da utilizzare senza il prefisso `FUNCTION` nelle istruzioni `PROCEDURE DIVISION` .

La funzione `LENGTH` ha una cronologia ordinata. Alcuni compilatori includono una parola riservata `LENGTH` . Per GnuCOBOL, questa parola riservata è riconosciuta solo quando viene usata nella frase `LENGTH OF` , il token `OF` è richiesto per disambiguare la funzione dall'estensione di parola riservata più vecchia.

Examples

Esempio di TRIM FUNZIONE

```
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) ":"
```

Mostrando

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

```
:    a string literal:
```

LETTERE MAIUSCOLE

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

Produzione

```
HELLO WORLD!
```

Funzione LOWER-CASE

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

Produzione

```
hello world!
```

Leggi Funzioni intrinseche online: https://riptutorial.com/it/cobol/topic/7580/funzioni-intrinseche

Capitolo 35: GENERARE la dichiarazione

Osservazioni

L'istruzione COBOL `GENERATE` è un'istruzione opzionale supportata se il compilatore include la funzione Report Writer.

Examples

GENERARE una riga di dettaglio

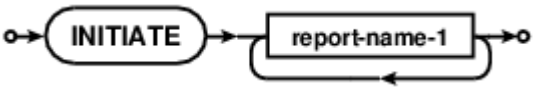
```
GENERATE detail-line
```

Leggi GENERARE la dichiarazione online: https://riptutorial.com/it/cobol/topic/7161/generare-la-dichiarazione

Capitolo 36: INIZIA dichiarazione

Osservazioni

L'istruzione `INITIATE` inizializza i campi di controllo di `Report Writer` interni. La maggior parte dell'impostazione del writer di report avviene nella `DATA DIVISION` con istruzioni di `PROCEDURE DIVISION` molto brevi. Una volta inizializzato, `GENERATE` fa tutto il duro lavoro di controllo interruzione e paging dei report.



Examples

INIZIARE le variabili di controllo dei rapporti

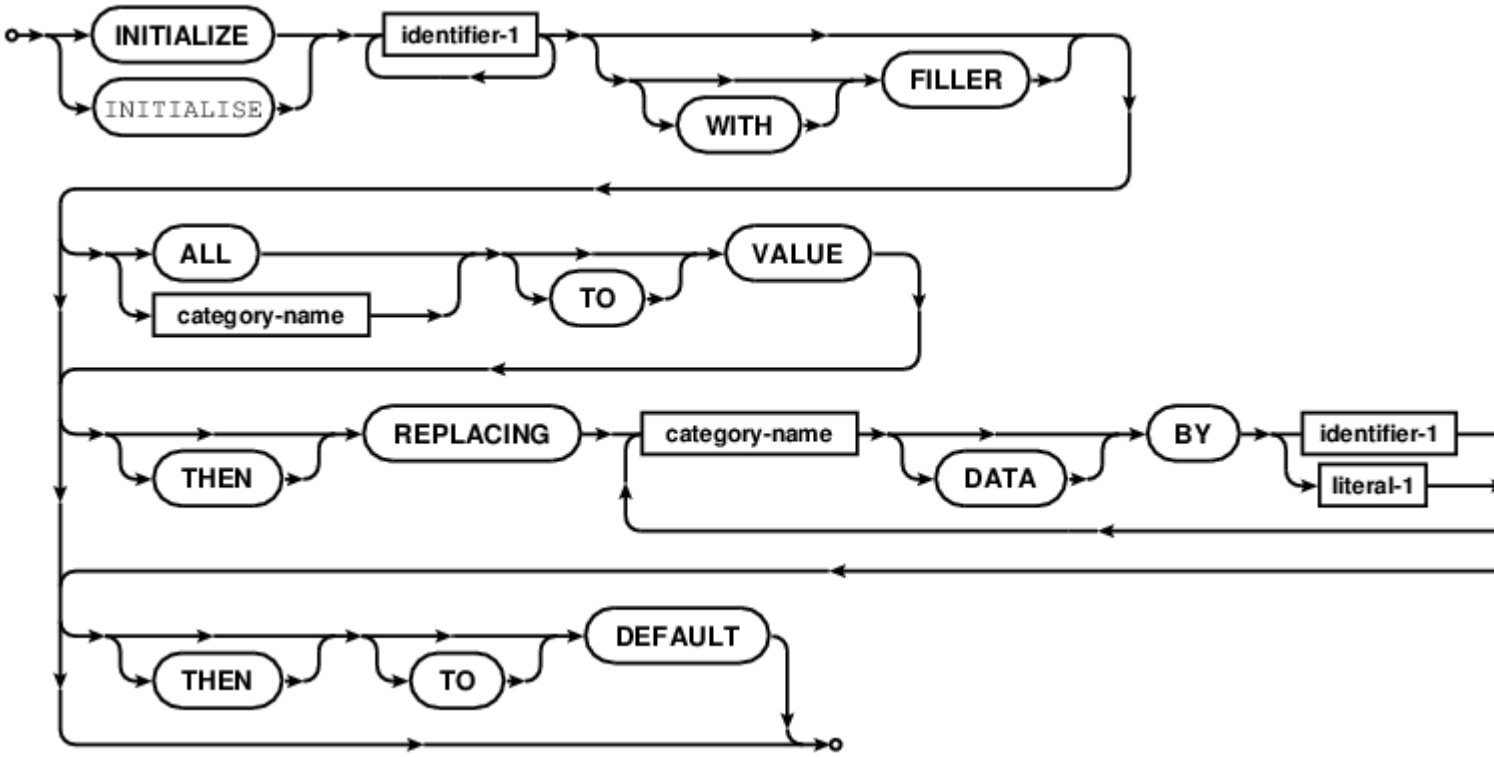
```
INITIATE report-1 report-2
```

Leggi INIZIA dichiarazione online: https://riptutorial.com/it/cobol/topic/7180/inizia-dichiarazione

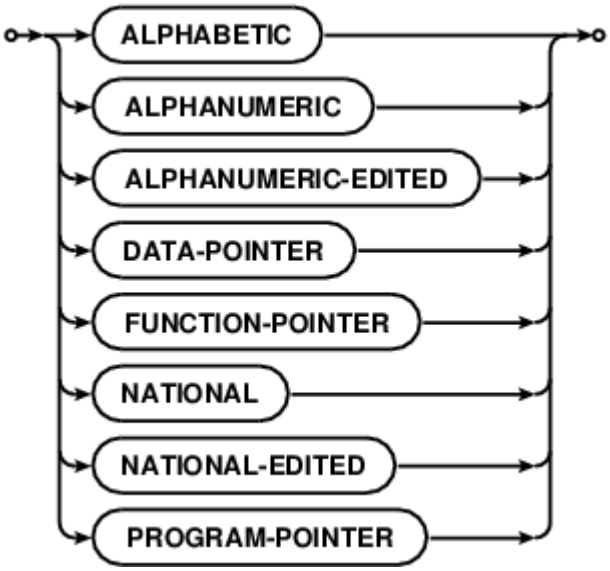
Capitolo 37: INIZIALIZZA dichiarazione

Osservazioni

L'istruzione `INITIALIZE` imposta i dati selezionati su valori specificati.



Dove `category-name` è:



Examples

Varie clausole INITIALIZE

```
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.
```

Leggi INIZIALIZZA dichiarazione online: https://riptutorial.com/it/cobol/topic/7179/inizializza-dichiarazione

Capitolo 38: Installazione di GnuCOBOL con GNU / Linux

Examples

Installazione GNU / Linux

Per la maggior parte delle distribuzioni GNU / Linux, una versione di `GnuCOBOL` è disponibile nei repository. `GnuCOBOL` era originariamente `OpenCOBOL` , rinominato quando il progetto è diventato un progetto GNU ufficiale. Molti repository utilizzano ancora `open-cobol` come nome del pacchetto (ad agosto 2016).

Per Fedora e altri gestori di pacchetti basati su RPM

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

Per pacchetti basati su Debian, Ubuntu e APT

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

Di solito è la versione 1.1 della suite del compilatore e si occuperà del tempo di compilazione e delle dipendenze di runtime richieste quando si utilizza GnuCOBOL.

Dalla fonte, (ospitata su SourceForge su https://sourceforge.net/projects/open-cobol/) è necessario.

- AC compilatore suite; `build-essential` (o simile)
- Intestazioni di sviluppo BerkeleyDB e BerkelyDB; `libdb` , `libdb-dev` (o nomi simili)
- Libreria numerica GNU Multi-Precision; `libgmp` , `libgmp-dev`
- Una versione di `curses` ; `ncurses` , `ncurses-dev`
- Il kit sorgente, `gnucobol-1.1.tar.gz` (o meglio, `gnucobol-2.0.tar.gz`)
- (Per cambiare le fonti del compilatore, sono necessari anche gli strumenti `GNU Autoconf`).

Da una directory di lavoro, a tua scelta:

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

Per vedere le possibili opzioni di configurazione, usa:

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

Poi

```
prompt$ ./configure
```

```
prompt$ make
```

Supponendo che le dipendenze siano a posto e la compilazione abbia esito positivo, verifica la preinstallazione con

```
prompt$ make check
```

o

```
prompt$ make checkall
```

Questo esegue controlli interni del compilatore (`make check`) e opzionalmente esegue test contro la suite di verifica NIST COBOL85 (`make checkall`). La versione 1.1 di OpenCOBOL copre circa 9100 test NIST, le versioni recenti coprono oltre 9700 passaggi di test. *La suite di test NIST COBOL85 non viene più gestita, ma è una serie di test molto completa e rispettabile. COBOL è altamente compatibile con le versioni precedenti, per finalità di progettazione, ma le nuove funzionalità COBOL 2002 e COBOL 2014 non fanno parte della suite di verifica NIST.*

Le verifiche interne coprono circa 500 test e compilazioni di codice di esempio.

Se tutto va bene, l'ultimo passo è

```
prompt$ sudo make install
```

Oppure, per i sistemi senza `sudo` , diventa l'utente root per `make install` o usa un prefisso `./configure` che non richiede permessi per superutente. Il prefisso predefinito per le build di origine è `/usr/local` .

Se si sono verificati più di una build sulla macchina e le librerie locali sono state reinstallate, è necessario eseguirne la verifica

```
prompt$ sudo ldconfig
```

Per garantire che il caricatore del linker `ld` cache sia correttamente aggiornato per corrispondere alla nuova installazione del compilatore.

`cobc` sarà pronto per l'uso.

`cobc --help` per un aiuto rapido, `info open-cobol` (o `info gnucobol`) per un aiuto più approfondito e visita http://open-cobol.sourceforge.net/ per i collegamenti alla Guida per il programmatore e un documento con le FAQ di oltre 1200 pagine.

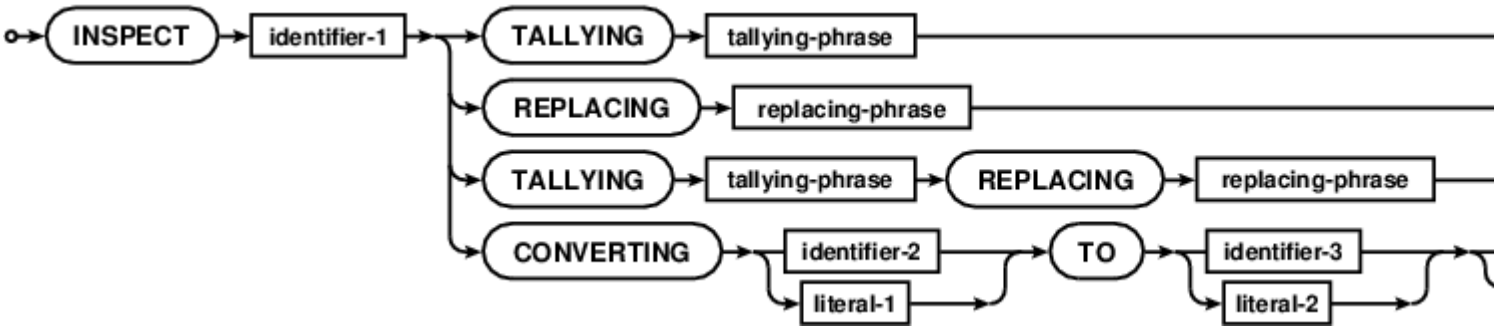
I problemi di installazione, i problemi o le domande generali possono essere postati nello spazio del progetto GnuCOBOL, nella `Help getting started` pagine di discussione su SourceForge.

Leggi Installazione di GnuCOBOL con GNU / Linux online:
https://riptutorial.com/it/cobol/topic/5446/installazione-di-gnucobol-con-gnu---linux

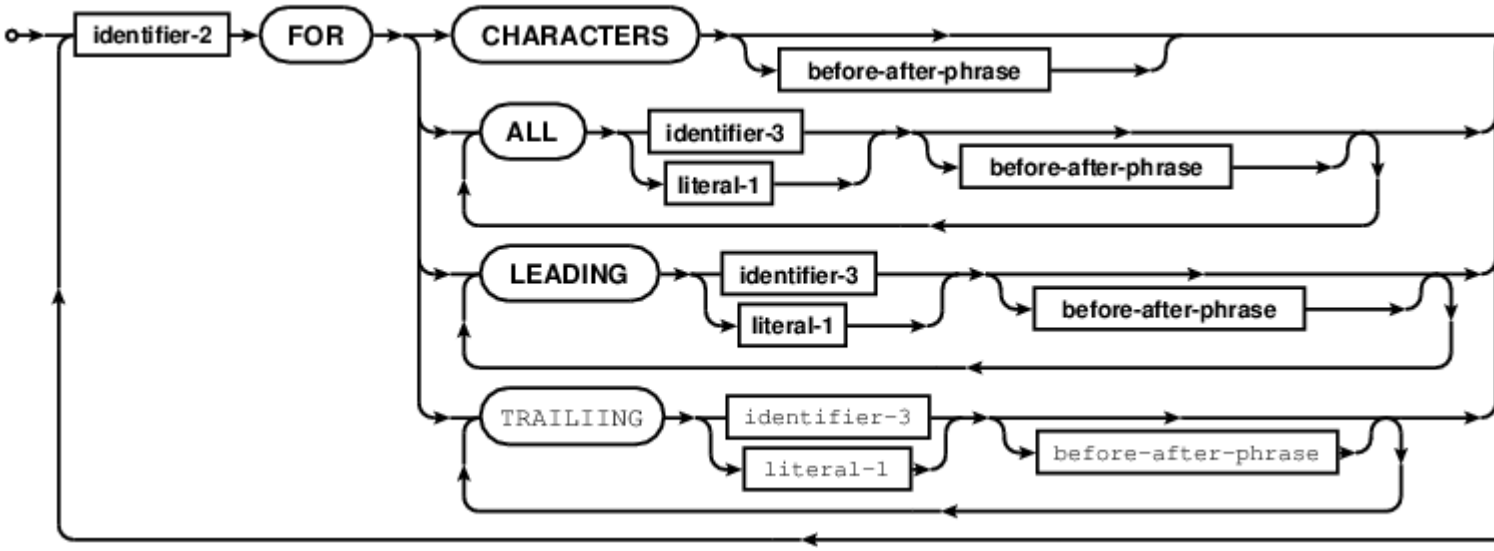
Capitolo 39: ISPEZIONARE la dichiarazione

Osservazioni

L'istruzione `INSPECT` è un verbo di scansione e sostituzione in COBOL.



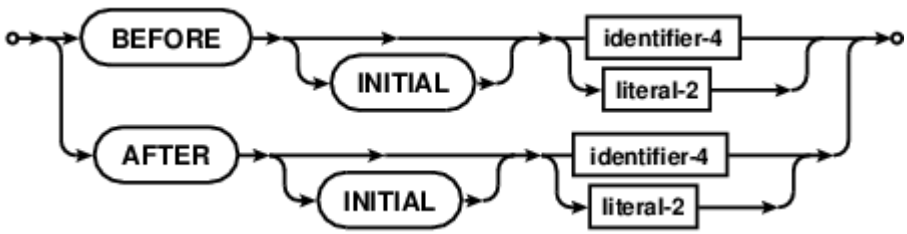
Dove è la `tallying-phrase` :



`replacing-phrase` è:

```
missing image
```

`before-after-phrase` è:



Examples

ISPEZIONARE la riformattazione di una data line

```
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:

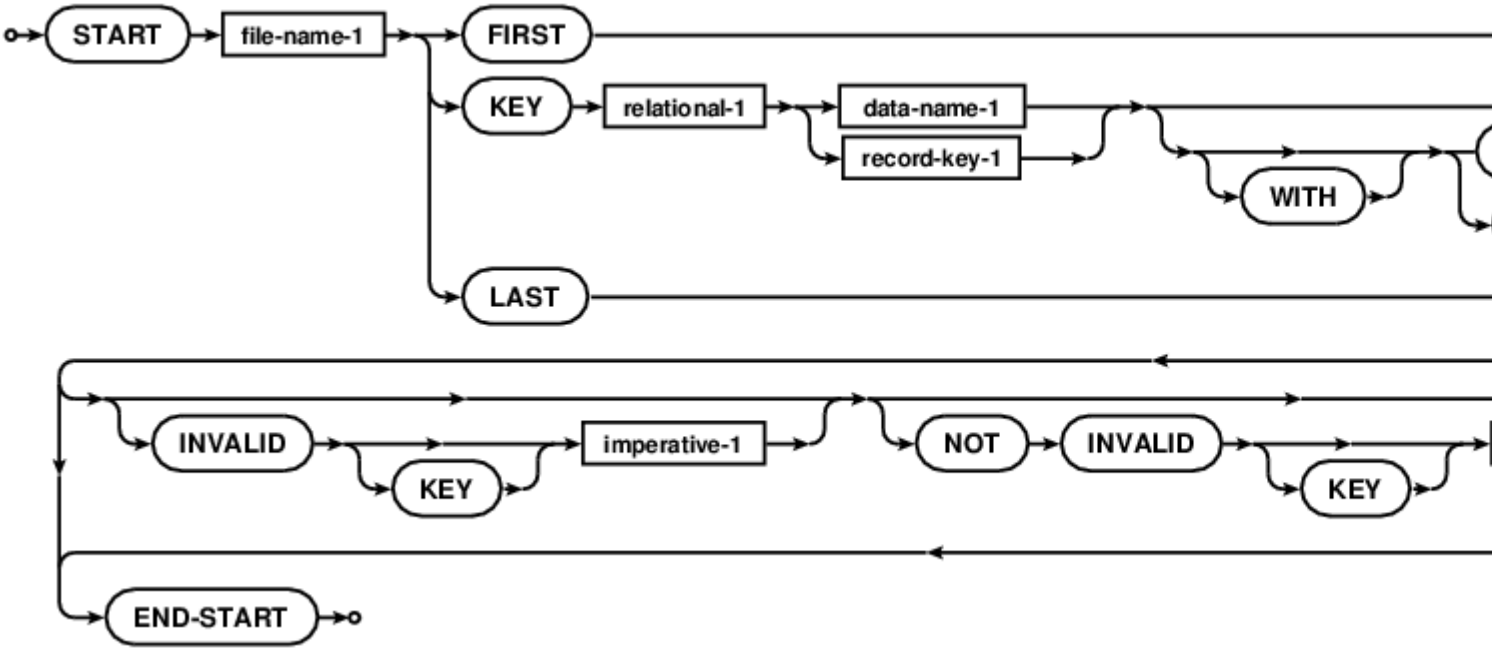
```
Formatted function WHEN-COMPILED 2010/03/25 23/05/0900-04/00
 after INSPECT REPLACING         2010/03/25 23:05:0900-04:00
```

Leggi ISPEZIONARE la dichiarazione online: https://riptutorial.com/it/cobol/topic/7182/ispezionare-la-dichiarazione

Capitolo 40: Istruzione START

Osservazioni

L'istruzione `START` fornisce un modo per posizionare una lettura in un file per il successivo recupero sequenziale (per chiave).



La chiave relazionale può includere (ma non è limitata a):

- LA CHIAVE È PIÙ GRANDE
- CHIAVE È>
- LA CHIAVE È MENO DI
- CHIAVE È <
- LA CHIAVE È PARI AL
- CHIAVE È =
- LA CHIAVE NON È PIÙ GRANDE
- LA CHIAVE NON È>
- LA CHIAVE NON È MENO DI QUANTO
- CHIAVE NON È <
- LA CHIAVE NON È UGUALE
- CHIAVE NON È =

- CHIAVE È <>
- LA CHIAVE È MAGGIORE O PARI AL
- CHIAVE È> =
- LA CHIAVE È MENO O PARI AL
- CHIAVE È <=

Examples

START esempio

```
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
```

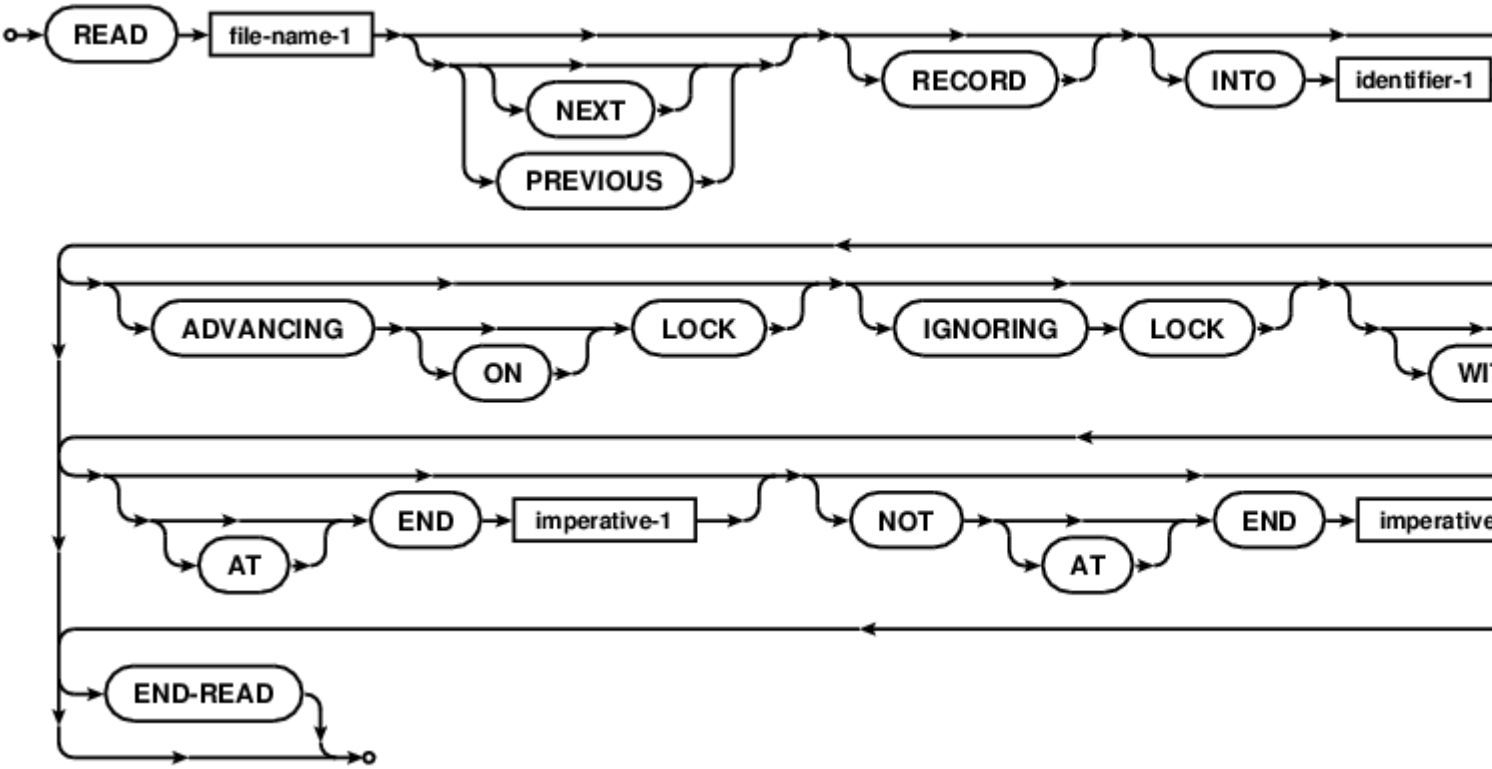
Leggi Istruzione START online: https://riptutorial.com/it/cobol/topic/7464/istruzione-start

Capitolo 41: LEGGI la dichiarazione

Osservazioni

L'istruzione `READ` è una graffetta della programmazione dell'elaborazione delle transazioni COBOL. Legge i dati dall'archiviazione esterna nel working store. Con o senza blocchi o condivisione, in sequenza, tramite accesso casuale o chiave. Si possono anche specificare clausole dichiarative per `AT END` , ma alcuni programmatori preferiscono test di stato `FILE STATUS` esplicito.

Poiché ogni risorsa file può contenere qualsiasi tipo di record in un dato spazio, COBOL è un linguaggio "leggi un file", "scrivi un record", `READ` accetta un nome file (FD) e spetta al programmatore posizionare il record in una struttura appropriata se i dati eterogenei vengono salvati nel file.



Examples

Semplice da leggere da FD

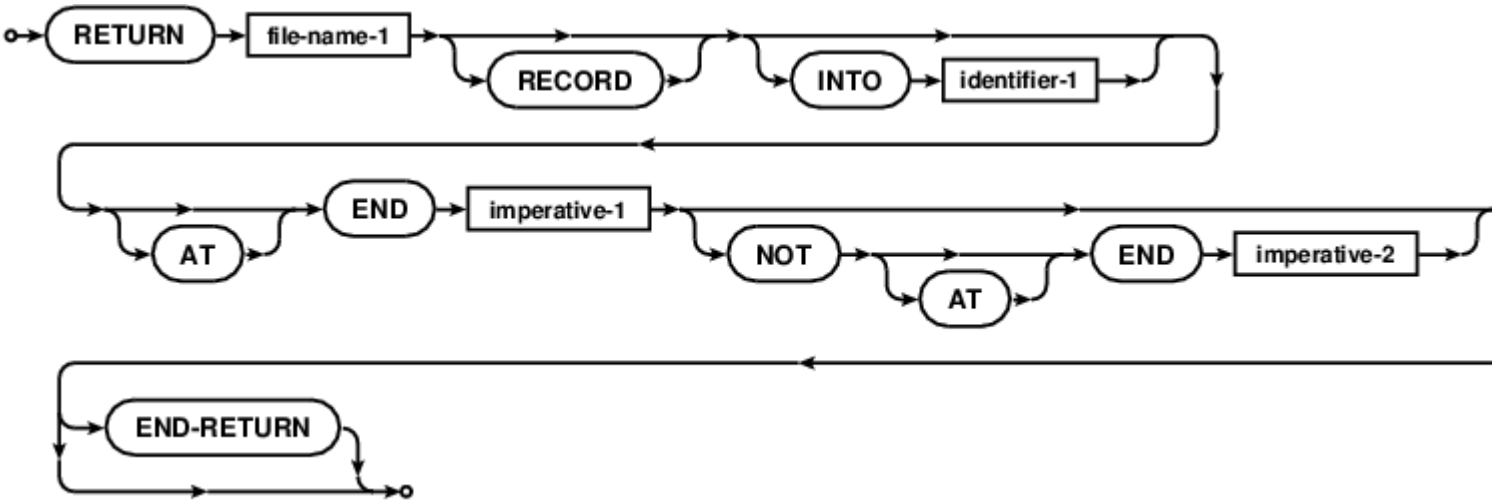
```
READ data-file
```

Leggi LEGGI la dichiarazione online: https://riptutorial.com/it/cobol/topic/7336/leggi-la-dichiarazione

Capitolo 42: RESTITUIRE la dichiarazione

Osservazioni

L'istruzione `RETURN` controlla quando i dati vengono inviati allo scrittore dell'algoritmo di ordinamento COBOL interno, come parte di una `OUTPUT PROCEDURE` . I dati di ordinamento dei post possono essere trasformati sotto il controllo del programmatore prima di essere restituiti e scritti nel file di output dall'algoritmo di ordinamento.



Examples

RESTITUISCI un record per SORT OUTPUT PROCEDURE

Questo è un campione di seedwork. `SORT OUTPUT PROCEDURE` può manipolare i record ordinati prima che vengano restituiti alla parte di scrittura dell'algoritmo di ordinamento COBOL interno. In questo caso, non viene eseguita alcuna trasformazione, il `work-rec` viene spostato direttamente in `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
```

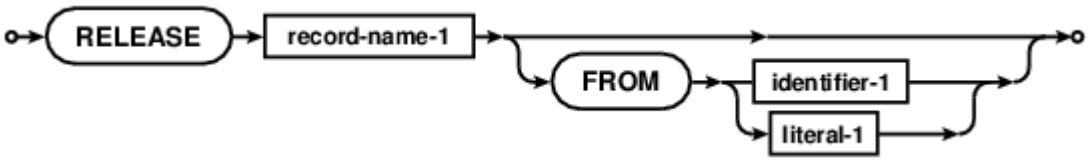
```
            move work-rec to out-rec
            write out-rec end-write
            RETURN sort-work
                at end move high-value to loop-flag
            end-return
        end-perform
        close sort-out
        .

        exit program.
        end program sorting.
```

Leggi RESTITUIRE la dichiarazione online: https://riptutorial.com/it/cobol/topic/7338/restituire-la-dichiarazione

Capitolo 43: RILASCIO dichiarazione

Osservazioni

L'istruzione `RELEASE` viene utilizzata per fornire record all'algoritmo COBOL `SORT` in condizioni controllate da programmatori.



Examples

RILASCIARE un record su una PROCEDURA DI INGRESSO ORDINATA

Questo è un esempio forzato. Ordina i record in base ad un `ALPHABET` che ha caratteri maiuscoli e minuscoli insieme, con `A` e `a` scambiato rispetto alle altre lettere. Questo è stato fatto apposta per dimostrare le possibilità. Il lettore dell'algoritmo SORT recupera i record utilizzando `RELEASE` nella `INPUT PROCEDURE` . `OUTPUT PROCEDURE` utilizza `RETURN` per il writer dell'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.
```

Leggi RILASCIO dichiarazione online: https://riptutorial.com/it/cobol/topic/7337/rilascio-

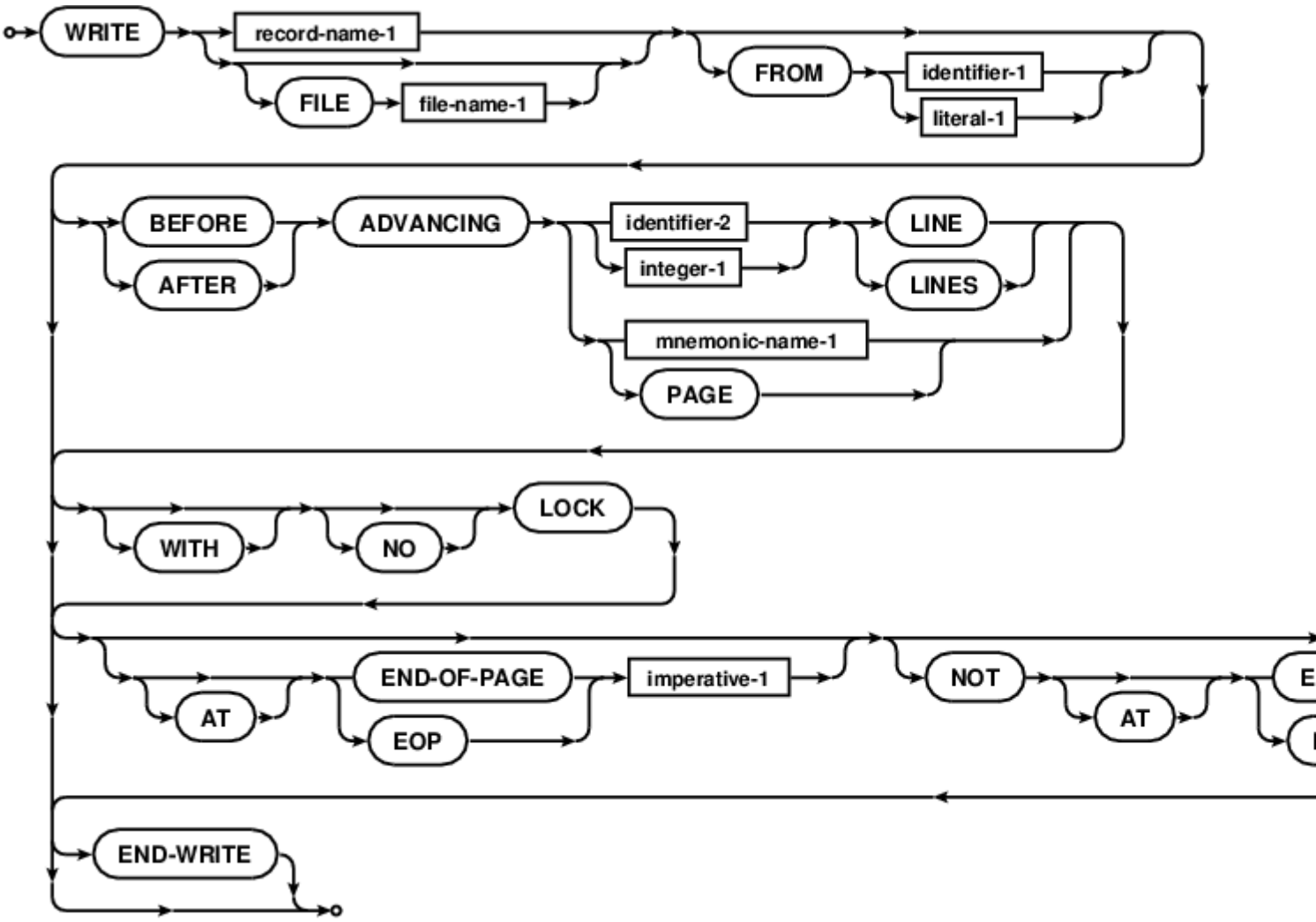
dichiarazione

Capitolo 44: SCRIVERE la dichiarazione

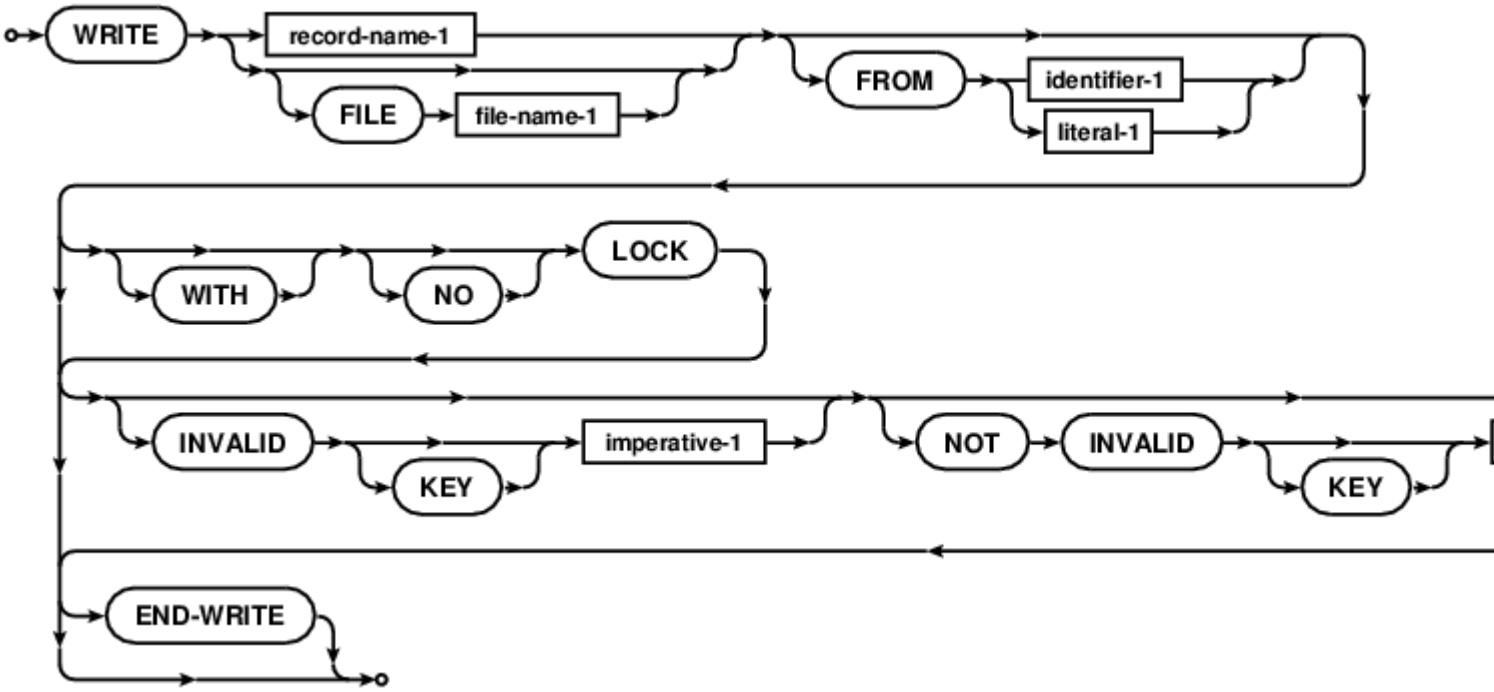
Osservazioni

L'istruzione `WRITE` rilascia record logici a una risorsa di archiviazione di `output` o `input-output` e per il posizionamento logico di righe all'interno di una pagina logica.

SCRIVI sequenziale



SCRIVERE casualmente



Examples

SCRIVI esempi

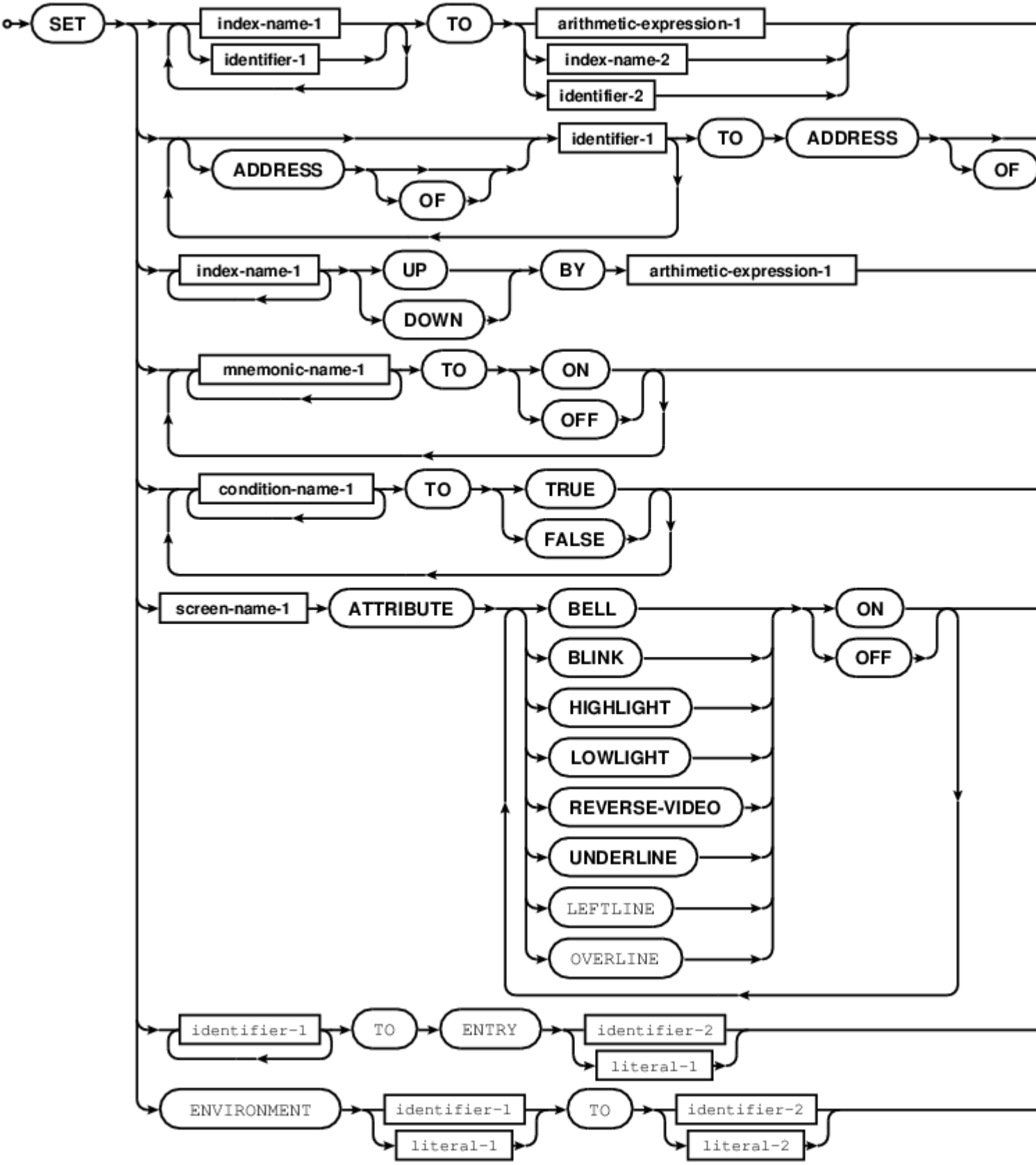
```
WRITE record-buff

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
```

Leggi SCRIVERE la dichiarazione online: https://riptutorial.com/it/cobol/topic/7583/scrivere-la-dichiarazione

Capitolo 45: SET statement

Osservazioni

L'istruzione COBOL `SET` imposta i valori e i dati dell'ambiente operativo. Si può affermare che `SET` stato abusato dal comitato, poiché ha oltre una dozzina di formati di sintassi documentati.



Examples

SET esempio puntatore

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

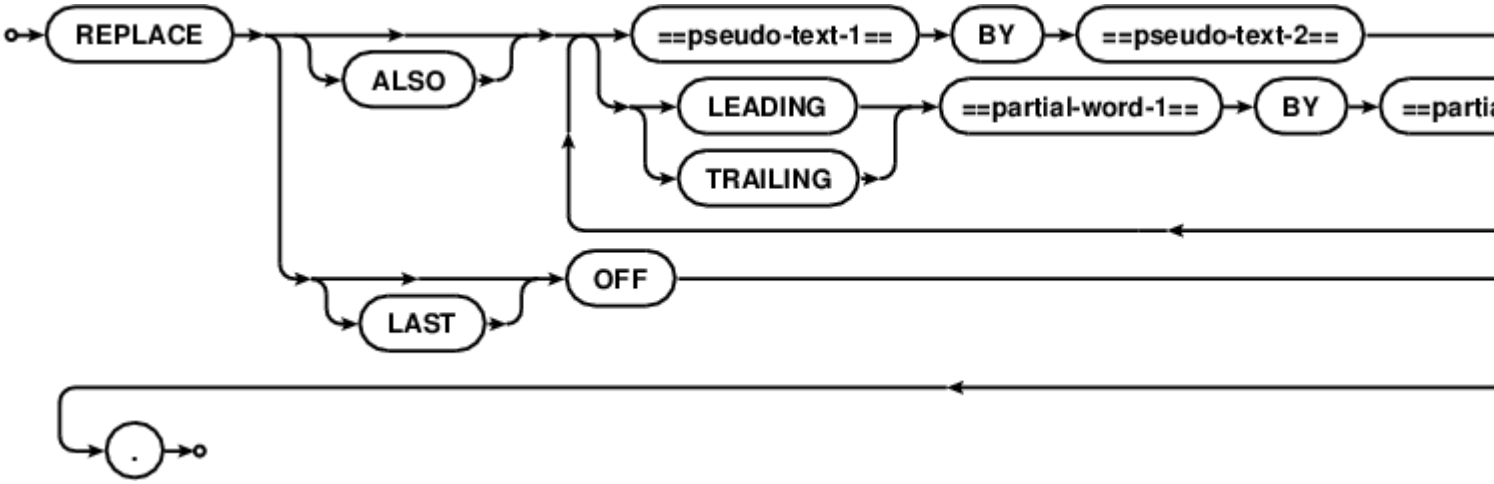
```
SET handle UP BY LENGTH(returned-pointer)
SET ADDRESS OF buffer-space TO handle
MOVE buffer-space TO work-store
DISPLAY "Second element is " work-store
```

Leggi SET statement online: https://riptutorial.com/it/cobol/topic/7461/set-statement

Capitolo 46: SOSTITUIRE la direttiva

Osservazioni

La direttiva `REPLACE` fa parte del preprocessore standard COBOL. Le sostituzioni sono fatte prima che inizi la compilazione.



Examples

SOSTITUISCI il campione di manipolazione del testo

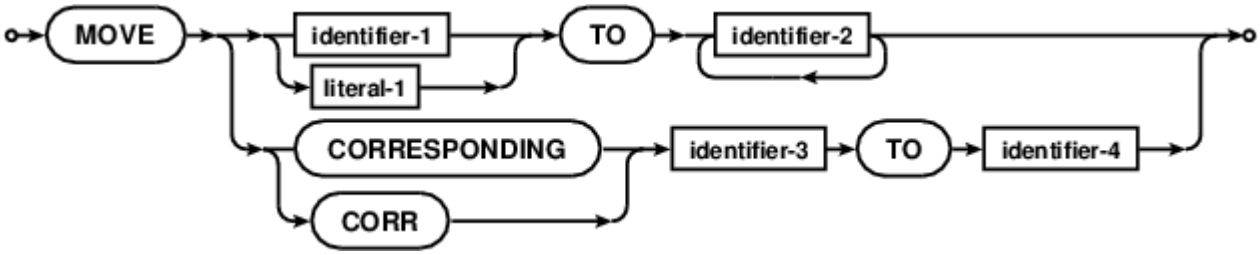
```
REPLACE ==magic-number== BY ==65535==.
```

Leggi SOSTITUIRE la direttiva online: https://riptutorial.com/it/cobol/topic/7459/sostituire-la-direttiva

Capitolo 47: Sposta la dichiarazione

Osservazioni

`MOVE` è il cavallo di battaglia di COBOL. I dati vengono spostati dal letterale o dall'identificatore a uno o più identificatori. COBOL ha una distinzione tra *elementare* e *gruppo* MOVE. I dati elementari sono di tipo convertito da origine a destinazione. I dati del gruppo vengono spostati come una matrice di byte, indipendentemente dai tipi di campo con una struttura. I campi numerici vengono spostati da destra a sinistra, troncamento di cifre di ordine superiore con riempimento a zero (normalmente). I dati alfanumerici dei caratteri vengono spostati da sinistra a destra, il troncamento dei caratteri di estremità destra con il riempimento dello spazio. Ci sono alcune regole su come `MOVE` sue attività, con i moduli dati BINARY e PICTURE DISPLAY e le gerarchie di gruppo che sono state prese in considerazione.



Examples

Alcuni dettagli MOVE, ce ne sono molti

```
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
```

Leggi Sposta la dichiarazione online: https://riptutorial.com/it/cobol/topic/7263/sposta-la-dichiarazione

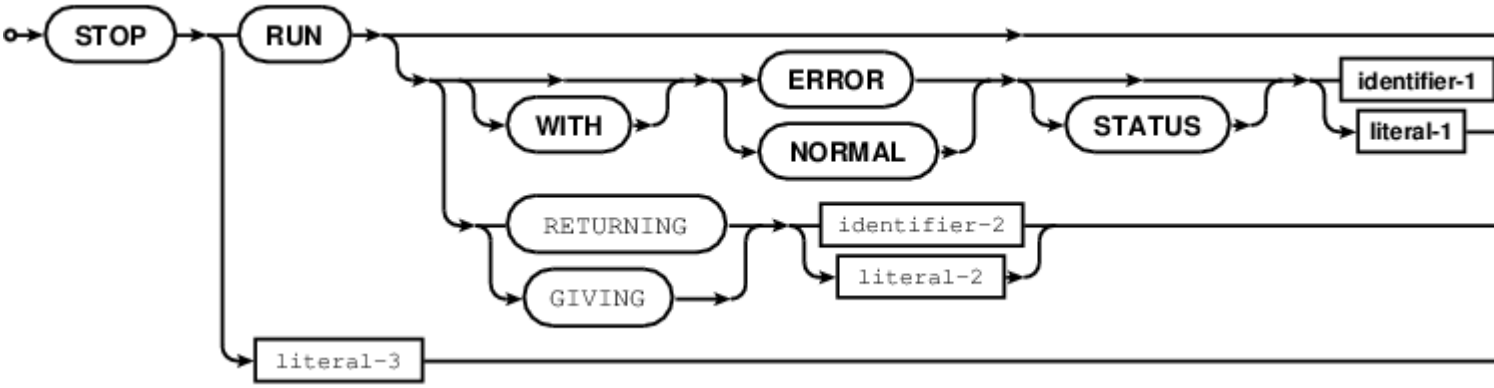
Capitolo 48: STOP statement

Osservazioni

L'istruzione `STOP` termina l'unità di esecuzione corrente.

Un'estensione ormai ritenuta obsoleta di `STOP RUN` è `STOP literal` , che interrompe un programma fino a quando viene fornita una risposta dalla console, a quel punto l'esecuzione riprenderà. Questo potrebbe essere utile per cose come "vai a prendere la grande scatola di carta e carica la stampante speciale".

`STOP` è un programma difficile, `GOBACK` è un modo leggermente migliore per tornare al sistema operativo o al modulo chiamante, specialmente nelle subroutine che potrebbero non avere attività commerciali che terminano una corsa.



Examples

ARRESTARE

```
STOP RUN
```

Leggi STOP statement online: https://riptutorial.com/it/cobol/topic/7466/stop-statement

Capitolo 49: Stringa

Examples

STRINGVAL ... Move -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.
```

Non un esempio, ma ....

sembrava l'unico modo per aggiungere un commento. Una cosa facile da dimenticare è che se stringa alcune variabili come nell'esempio sopra, e la lunghezza risultante è più breve di quella che era originariamente nella variabile ricevente (o-string sopra), i caratteri "finali" vengono lasciati sul posto.

Ad esempio, se o- string contiene "la stringa contiene questi dati" e stringa insieme "fred & Bert", allora la stringa conterrà "fred & Bertontains questi dati" (se ho contato bene).

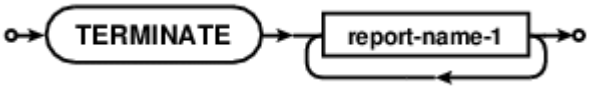
Sommario Summa, prendi l'abitudine di spostare SEMPRE gli spazi nella variabile ricevente prima di iniziare l'incordatura.

Leggi Stringa online: https://riptutorial.com/it/cobol/topic/7039/stringa

Capitolo 50: TERMINATE statement

Osservazioni

L'istruzione `TERMINATE` è una funzione di Report Writer COBOL. Termina l'elaborazione sui nomi dei report indicati.



Examples

Termina esempio

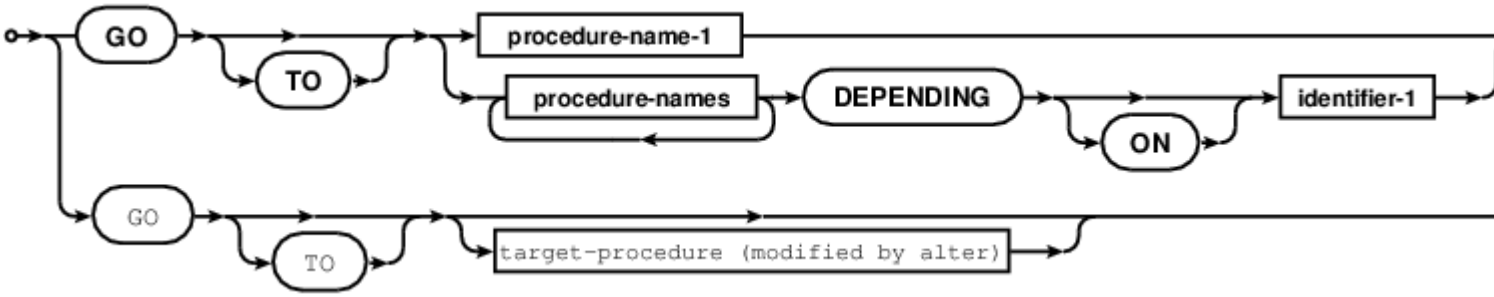
```
TERMINATE report-1 report-2 report-summary
```

Leggi TERMINATE statement online: https://riptutorial.com/it/cobol/topic/7467/terminate-statement

Capitolo 51: VAI ALLA dichiarazione

Osservazioni

Il tanto amato `GO TO` COBOL include paragrafi e sezioni con nome, insieme ad altre etichette, e ognuno di essi può essere il bersaglio di una dichiarazione `GO` .



Examples

GO dichiarazione

```
GO TO label

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

GO TO label OF section

GO.
```

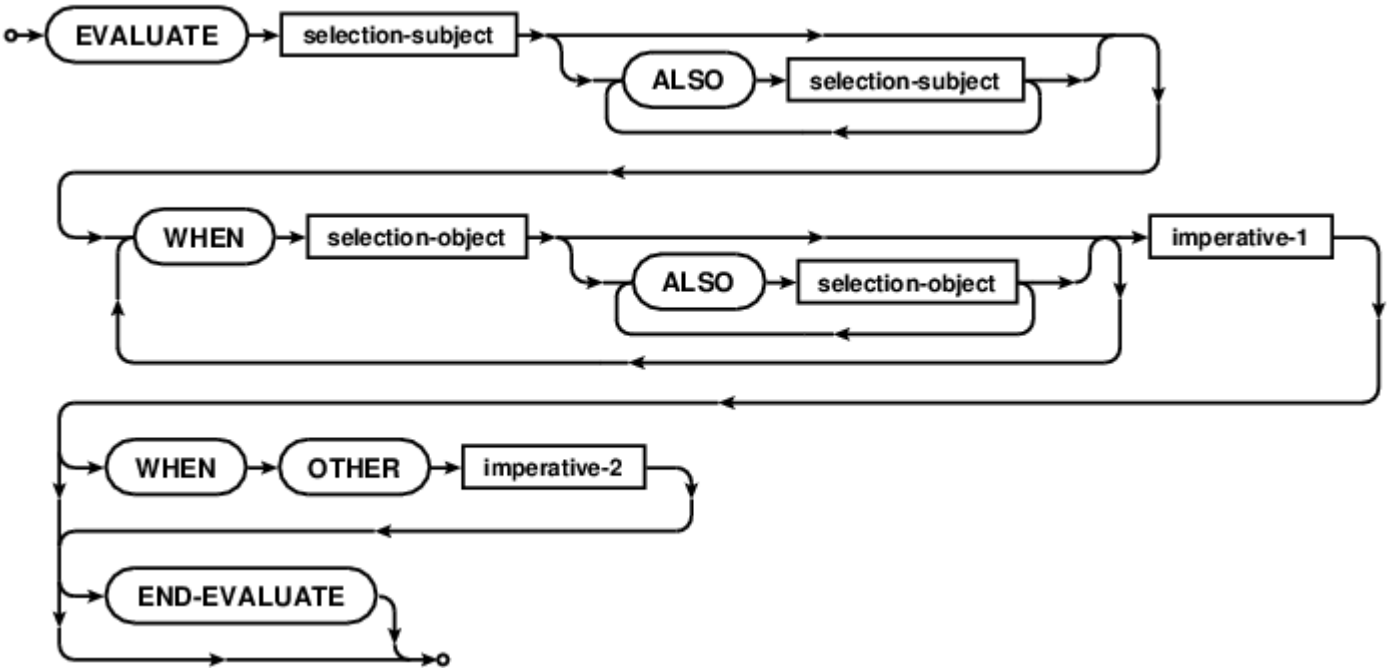
L'ultimo esempio di riga indica che è in gioco un'istruzione `ALTER` e un'altra parte del codice specificherà quale `label` effettiva è l'obiettivo del salto.

Leggi VAI ALLA dichiarazione online: https://riptutorial.com/it/cobol/topic/7163/vai-alla-dichiarazione

Capitolo 52: VALUTARE la dichiarazione

Osservazioni

L'istruzione `EVALUATE` è una struttura a più rami, più join, test condizionale e selezione.



Examples

A tre condizioni VALUTARE

```
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
```

Leggi VALUTARE la dichiarazione online: https://riptutorial.com/it/cobol/topic/7083/valutare-la-dichiarazione

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con cobol	4444, Abhishek Jain, Bharat Anand, Brian Tiffin, Community, Joe Zitzelberger, ncmathsadist
2	ADD dichiarazione	Brian Tiffin
3	ALLOCATE dichiarazione	Brian Tiffin
4	ANNULLA dichiarazione	Brian Tiffin
5	Come funziona il calcolo in cobol?	Bruce Martin, Bulut Colak
6	CONTINUA la dichiarazione	Brian Tiffin
7	Dichiarazione ACCEPT	Brian Tiffin
8	Dichiarazione ALTER	Brian Tiffin
9	Dichiarazione APERTA	Brian Tiffin
10	Dichiarazione COMPUTE	Brian Tiffin
11	Dichiarazione d'uso	Brian Tiffin
12	Dichiarazione di CHIAMATA	4444, Bill Woodger, Brian Tiffin, infoRene, Jeffrey Ranney, Joe Zitzelberger, Simon Sobisch
13	Dichiarazione di COMMIT	Brian Tiffin
14	Dichiarazione di PERFORM	Brian Tiffin
15	Dichiarazione di ricerca	Brian Tiffin

16	Dichiarazione DISPLAY	Brian Tiffin
17	Dichiarazione DIVIDE	Brian Tiffin
18	Dichiarazione GOBACK	Brian Tiffin
19	Dichiarazione GRATUITA	Brian Tiffin
20	Dichiarazione IF	Brian Tiffin
21	Dichiarazione MERGE	Brian Tiffin
22	Dichiarazione MULTIPLY	Brian Tiffin
23	Dichiarazione REWRITE	Brian Tiffin
24	Dichiarazione SORT	Brian Tiffin
25	Dichiarazione SOTTRA	Brian Tiffin
26	Dichiarazione STRING	Brian Tiffin
27	Dichiarazione SUPPRESS	Brian Tiffin
28	Dichiarazione UNLOCK	Brian Tiffin
29	Dichiarazione UNSTRING	Brian Tiffin
30	Direttiva COPY	Brian Tiffin
31	Divisione dati	Bulut Colak
32	ELIMINA la dichiarazione	Brian Tiffin
33	EXIT statement	Brian Tiffin
34	Funzioni intrinseche	Brian Tiffin, MC Emperor

35	GENERARE la dichiarazione	Brian Tiffin
36	INIZIA dichiarazione	Brian Tiffin
37	INIZIALIZZA dichiarazione	Brian Tiffin
38	Installazione di GnuCOBOL con GNU / Linux	Brian Tiffin
39	ISPEZIONARE la dichiarazione	Brian Tiffin
40	Istruzione START	Brian Tiffin
41	LEGGI la dichiarazione	Brian Tiffin
42	RESTITUIRE la dichiarazione	Brian Tiffin
43	RILASCIO dichiarazione	Brian Tiffin
44	SCRIVERE la dichiarazione	Brian Tiffin
45	SET statement	Brian Tiffin
46	SOSTITUIRE la direttiva	Brian Tiffin
47	Sposta la dichiarazione	Brian Tiffin
48	STOP statement	Brian Tiffin
49	Stringa	Jeffrey Ranney, Michael Simpson
50	TERMINATE statement	Brian Tiffin
51	VAI ALLA dichiarazione	Brian Tiffin
52	VALUTARE la dichiarazione	Brian Tiffin