FREE eBook

LEARNING

cobol

Free unaffiliated eBook created from
Stack Overflow contributors.

#cobol

Table of Contents

About

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

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

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

Chapter 1: Getting started with cobol

Remarks

COBOL is the **CO**mmon **B**usiness **O**riented programming **L**anguage.

Even though it has become a pronounceable name, COBOL is still treated as an acronym by the standards committee, and COBOL is the preferred spelling by the ISO and INCITS standards bodies.

Standard Specification

The current specification is

ISO/IEC 1989:2014 Information technology – Programming languages, their environments and system software interfaces – Programming language COBOL

That document was published in May of 2014 and can be purchased from various branches of standard bodies, officially homed at

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

Principal field of use

Business oriented. That usually means transaction processing. Banking, government agencies, and the insurance industry are major areas of COBOL application deployments. IBM mainframe systems usually have a COBOL compiler installed. There are upwards of 300 COBOL dialects in existence, with perhaps 10 or so versions taking the lion's share of deployments. Most of these compilers are proprietary systems, but free software COBOL is also available.

Category

COBOL is a procedural, imperative, compiled programming language. As of the COBOL 2002 spec, Object Oriented features were added to the standard.

By design intent, COBOL is a very verbose programming language. Although algebraic form is allowed:

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

the initial intent was to use full words for computational descriptions and data manipulation:

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

This design decision has both champions and detractors. Some feel it is too verbose, while others

argue that the syntax allows for greater readability in a business environment.

Decimal Math

COBOL is designed around decimal arithmetic, unlike most languages that use a binary internal representation. The COBOL spec calls for very precise fixed point decimal calculations, an aspect of the language that has been well regarded in financial sectors. *COBOL also allows for USAGE BINARY, but leans towards decimal (base-10) representations.*

History

COBOL dates back to the late 1950s, with initial implementations published in 1960.

U.S. Navy Rear Admiral Grace Hopper is often associated with COBOL, and championed on behalf of the language during the early stages of development. She was not the only person involved in the design and development of COBOL, by any means, but is often referred to as the Mother of COBOL.

Due to early backing by governments and large corporations, COBOL has been in wide use for many decades. It remains a point of pride for some, and a thorn for others, who see it as outdated. The truth likely lies somewhere in between these extreme views. When applied to transaction processing, COBOL is at home. When applied to modern web screens and networking applications it may not feel as comfortable.

Structure

COBOL programs are written in four separate divisions.

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- PROCEDURE DIVISION

Data Descriptions

Being designed to handle decimal data, COBOL allows for PICTURE based data descriptions, in grouped hierarchies.

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

That defines `balance` as a signed eight digit value with two digits assumed after the decimal point. `rate` is three digits before and three digits after an assumed decimal point. `show-balance` is a numeric-edit field that will have a leading dollar sign, seven digits (zero suppressed) with at least one digit shown preceding two digits after a decimal point.

`balance` can be used in calculations, `show-balance` is only for display purposes and cannot be used in computational instructions.

Procedural statements

COBOL is a reserved keyword heavy language. MOVE, COMPUTE, MULTIPLY, PERFORM style long form words make up most of the standard specification. Over 300 keywords and 47 operational statements in the COBOL 2014 spec. Many compiler implementations add even more to the reserved word list.

Examples

Hello, world

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

The days of punch card layout and uppercase only inputs are far behind. Yet most COBOL implementations still handle the same code layout. Even current implementations follow the same (often even in uppercase,) compiled and in production.

A well-formatted modern implementation might look like:

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

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

With some implementations of COBOL, this can be shortened to:

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

This format usually requires compile time switches to put a COBOL compiler into a relaxed syntax mode, as some of the normally mandatory `DIVISION` statements are missing.

COBOL assumes FIXED format sources by default, even in the current specification.

Pre-2002 COBOL

Column	Area
1-6	Sequence Number Area
7	Indicator Area
8-12	Area A
12-72	Area B
73-80	Program Name Area

IBM mainframe text editors are still configured for this form in some cases.

Post 2002 and into COBOL 2014, Area A and B were merged and extended to column 255, and the Program Name Area was dropped.

Column	Area
1-6	Sequence Number Area
7	Indicator Area
8-	Program text Area

Column 8 thru an implementation defined column *Margin R*, is usually still limited to column 72, but allowed by spec to run up to column 255.

COBOL 2002 introduced `FORMAT FREE` source text. There is no *Sequence Number Area*, no *Indicator Area*, and source lines can be any length (up to an implementation defined *Margin R* limit, usually less than 2048 characters per line, commonly 255).

But the compiler starts out in FORMAT FIXED mode by default. There is usually a compilation switch or *Compiler Directive Facility* statement before free format source is recognized.

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

Where `bbbbbb` represents 6 blanks, or any other characters. (These are ignored as part of the initial default fixed format mode Sequence Number Area.)

Install gnu-cobol on Mac OS X

gnu-cobol is available via the homebrew system.

Open a terminal window from `/Applications/Utilities/Terminal` or use the keypress `Command+Space` and type `"Terminal"`.

If you do not have the homebrew system installed, add it by typing, or copying and pasting into your terminal:

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

Once the command has finished, type:

```
brew install gnu-cobol
```

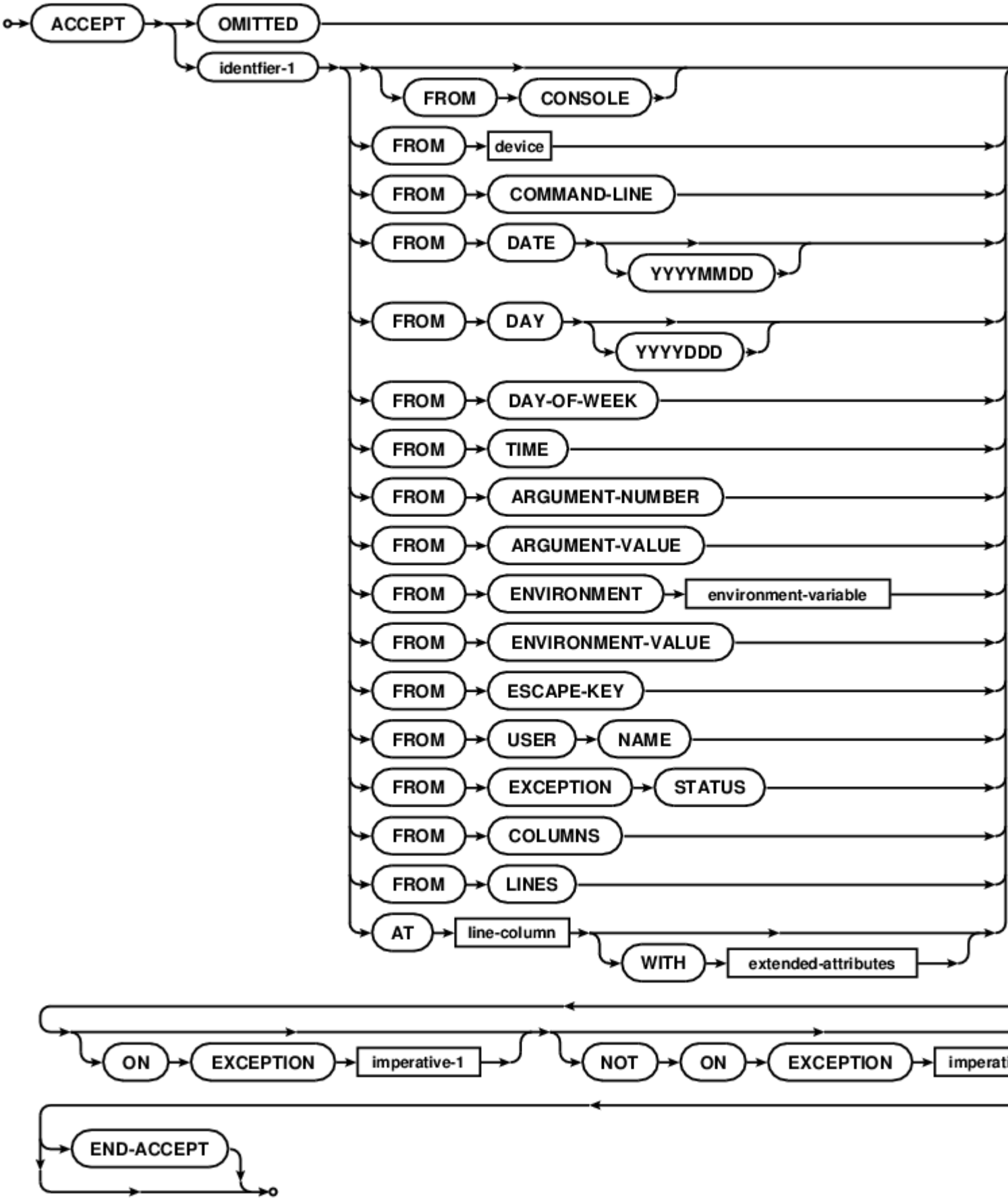
That is it, you can now compile Cobol programs on your Mac.

Read Getting started with cobol online: https://riptutorial.com/cobol/topic/4728/getting-started-with-cobol

Chapter 2: ACCEPT statement

Remarks

The COBOL ACCEPT statement is used to retrieve data from the system.



Examples

ACCEPT statement

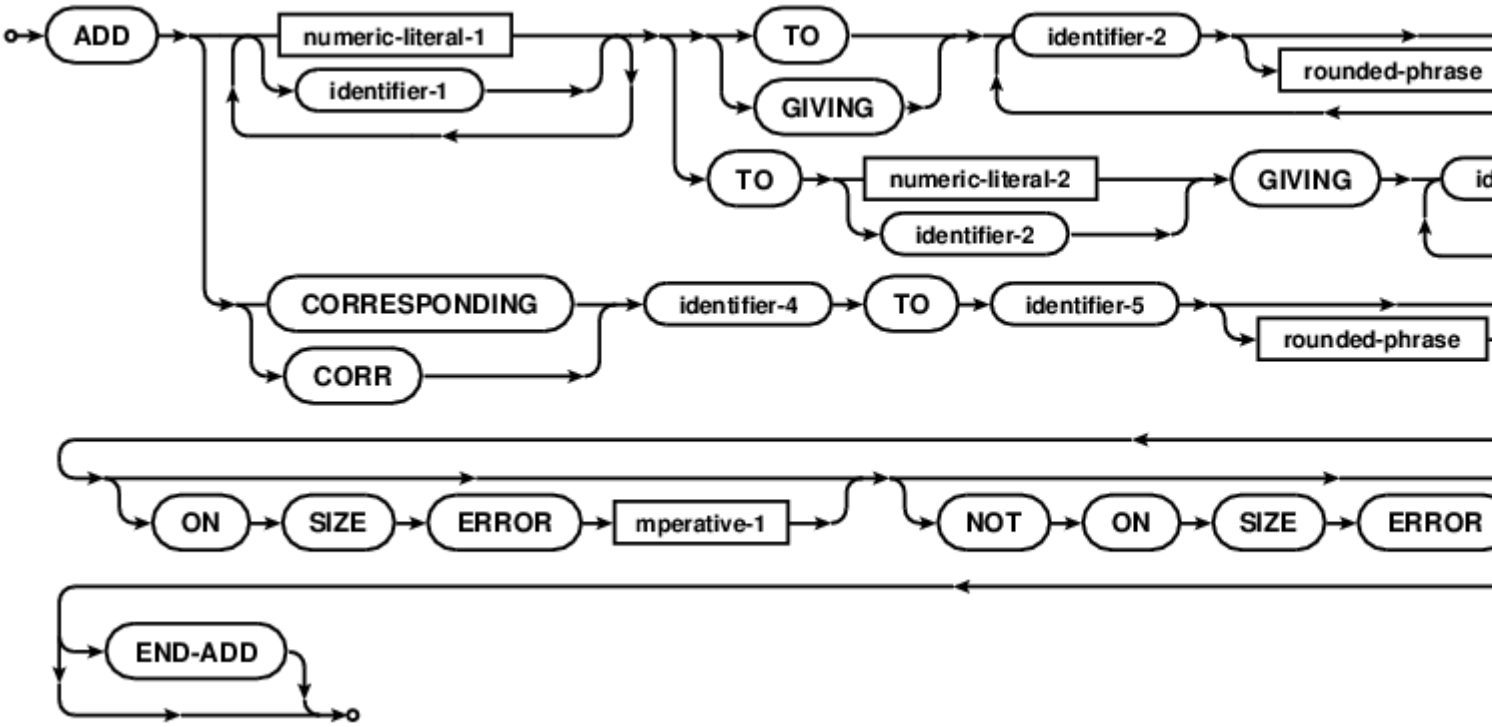
```
ACCEPT variable.
ACCEPT variable FROM CONSOLE.

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.
```

See http://open-cobol.sourceforge.net/faq/index.html#accept for more details.

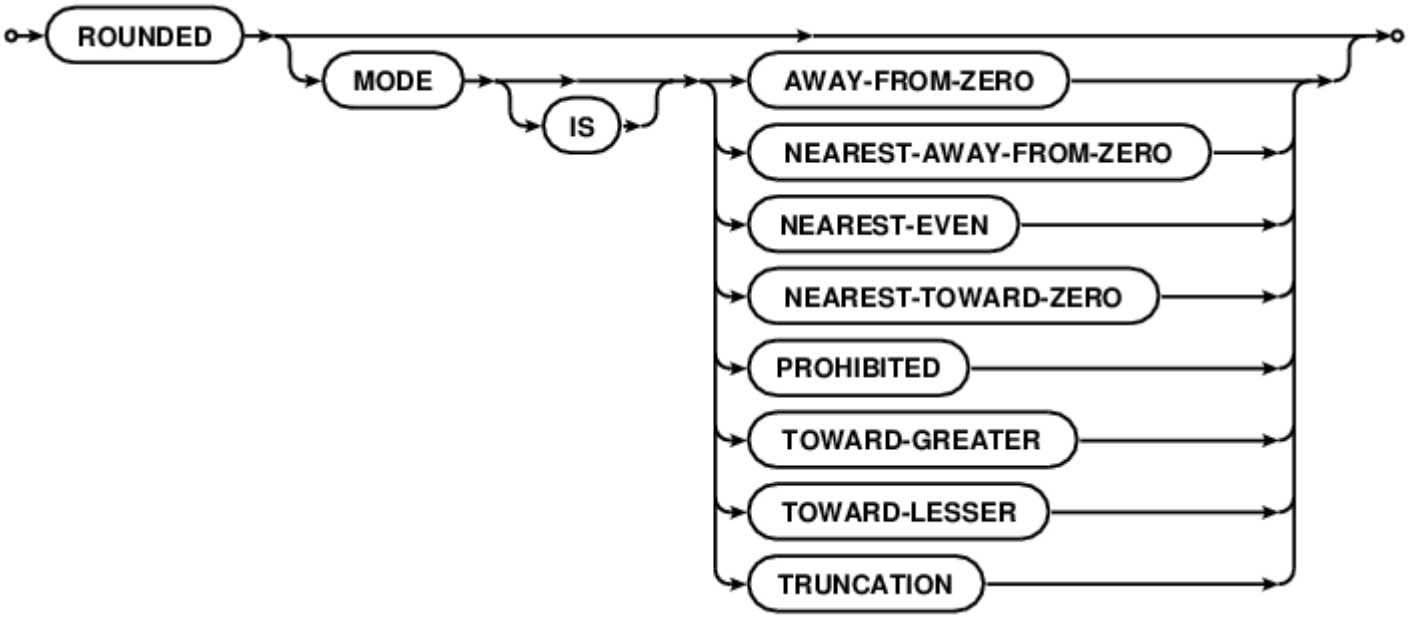
Read ACCEPT statement online: https://riptutorial.com/cobol/topic/5512/accept-statement

Chapter 3: ADD statement

Remarks



Where rounded-phase is



Examples

ADD statement

```
ADD 1 TO cobol
```

This modifies the variable `cobol`. Overflow silently ignored.

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

This doesn't modify `cobol`, the result of the ADD being stored in `GnuCOBOL`. Again, overflow of the storage allocation silently ignored (the field will stay at its old value on size errors and there will be no exception raised).

```
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
```

Multiple inputs are allowed, with storage size testing explicit. COBOL has an intrinsic `FUNCTION E`, so it not a wise choice for a single letter identifier.

`SIZE ERROR` in COBOL is dependent on type and/or `PICTURE`. A `PIC 9` field will only safely store values from 0 to 9, an intermediate result of 10 would trigger the `ON SIZE ERROR` phrase in that case.

Read ADD statement online: https://riptutorial.com/cobol/topic/5533/add-statement

Chapter 4: ALLOCATE statement

Remarks

Allocate working storage for a BASED item, or allocate a give size of heap storage.

See also: FREE statement

Examples

ALLOCATE statement

```
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
```

See http://open-cobol.sourceforge.net/faq/index.html#allocate for more details.

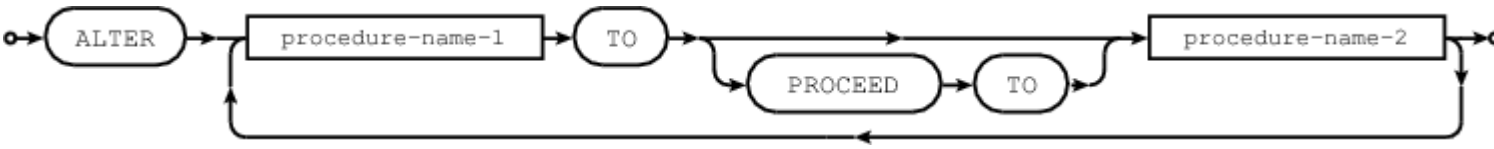
Read ALLOCATE statement online: https://riptutorial.com/cobol/topic/5556/allocate-statement

Chapter 5: ALTER statement

Remarks

The much beloved ALTER statement. Changes the target of a GO TO paragraph.

No longer part of the COBOL standard, still supported by many compilers for reasons of backward compatibility. *(The syntax diagram is dimmed to show that this is no longer standard COBOL).*



Examples

A contrived example using 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.
```

With a run sample of

```
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$
```

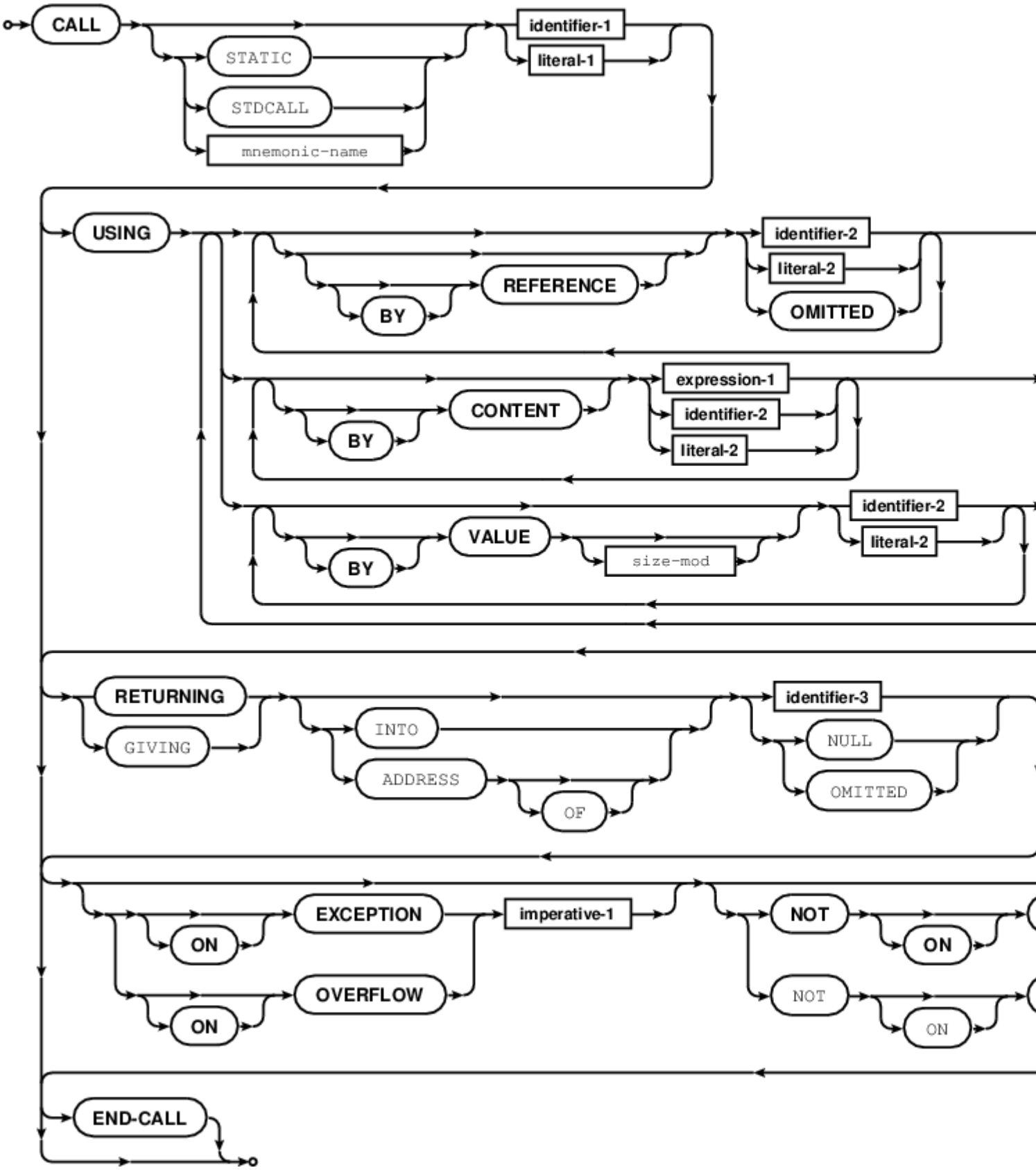
See http://open-cobol.sourceforge.net/faq/index.html#alter for more details.

Read ALTER statement online: https://riptutorial.com/cobol/topic/5584/alter-statement

Chapter 6: CALL statement

Remarks

The COBOL CALL statement provides access to compiled library routines.



Examples

CALL statement

COBOL can use static linkage for the following statement. GnuCOBOL uses dynamic linkage by default for all external symbols known at compile time, even when the symbol is a literal:

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

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

This statement forces compile time link edit resolution. *(Non standard, syntax extension)*:

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

Fields in COBOL can be passed `BY REFERENCE` (the default, until overridden - overrides are `sticky` in a left to right order), `BY CONTENT` (a copy is passed BY REFERENCE), or in some cases directly `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 is designed to be a `BY REFERENCE` language, so using `BY VALUE` can present issues. For instance, literal numerics have no explicit type and the COBOL spec has no explicit type promotion rules. Therefore developers have to worry about call frame setup with `BY VALUE` of literals.

See http://open-cobol.sourceforge.net/faq/index.html#call for more details.

SLEEPY TIME

CALL is also a way to extend COBOL functionality, and also to allow the reusability of code. It can also give access to "system" functionality.

This example illustrates ways to provide "sleep" functionality to IBM Mainframe COBOLs. Bear in mind that the requirement to do so is rare to the extent that usually when someone thinks they need to "sleep" for some reason, it is the wrong thing to do.

ILBOWAT0 is from the old COBOL-specific runtime era on Mainframes. BXP1SLP and BXP4SLP are Unix System Services (USS) routines which can be used by any language. Effectively they are Unix "sleep" requests.

The current IBM Mainframe Runtime (Language Environment (LE)) provides for inter-language communication, and the CEE3DLY LE services is shown in another example, Using z/OS Language Environment thread delay service.

ILBOWAT0 has been around for a very long time (perhaps more than 40 years), and you may still come across it. It's use should be replaced by CEE3DLY or BXP1SLP, whichever is the more appropriate for the particular requirement.

Sometimes you need to cause a program to sleep, or cause a Job to sleep for a while (after an FTP or NDM step), which are usually run as separate jobs, and you would need to sleep/loop looking for the resulting datasets.

Here is a cute little COBOL program to do said task, calling the COBOL sleep programs available in OS/VS and perhaps other legacy and current mainframe operating environments.

```
       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     .
```

microfocus way

For Microfocus, it uses the "SleepEx" API. As an example;

```
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.
```

Using z/OS Language Environment thread delay service

You can call the CEE3DLY service in 24- 31- or 64- bit mode to delay a task to the nearest second. It is CICS save and will only delay the thread.

An example:

```
    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.
```

You can see more detail here:

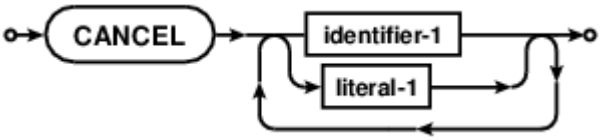
IBM Language Environment Callable Services - Sleep

Read CALL statement online: https://riptutorial.com/cobol/topic/5601/call-statement

Chapter 7: CANCEL statement

Remarks

The CANCEL statement ensures that a referenced program will be in an initial state the next time it is called, and to unload any resources for the module.



Examples

CANCEL statement

```
CALL "submodule"
CALL "submodule"

CANCEL "submodule"
CALL "submodule"
```

Any static data in the working set of `submodule` will be in an initial state on the last `CALL` statement above. The second `CALL` will have any initial values set as left overs from the first `CALL`.

COBOL compilers can support physical cancel (object unloaded from memory) and/or virtual cancel (ensure an initial state, but leave the object available to the host operating environment). This is an implementation detail.

See http://open-cobol.sourceforge.net/faq/index.html#cancel for more details.

Read CANCEL statement online: https://riptutorial.com/cobol/topic/5600/cancel-statement

Chapter 8: COMMIT statement

Remarks

Flushes ALL current locks, synching file I/O buffers.

This is a non standard extension, available with some COBOL implementations that support `ROLLBACK` features.

Examples

COMMIT statement

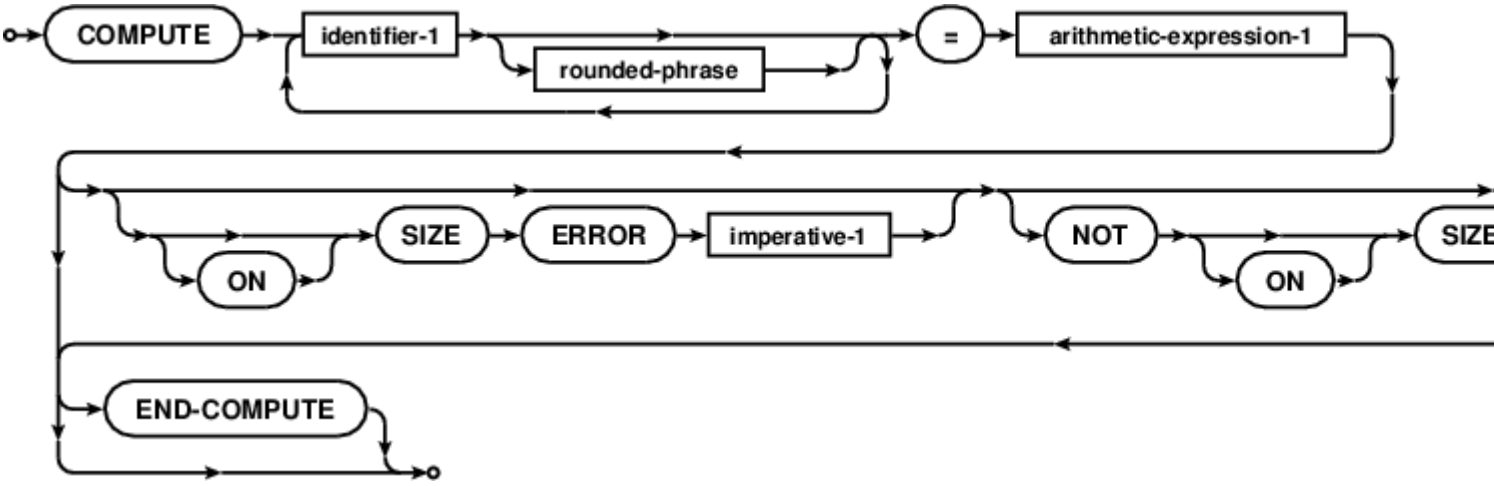
```
WRITE record
COMMIT
```

Read COMMIT statement online: https://riptutorial.com/cobol/topic/6357/commit-statement

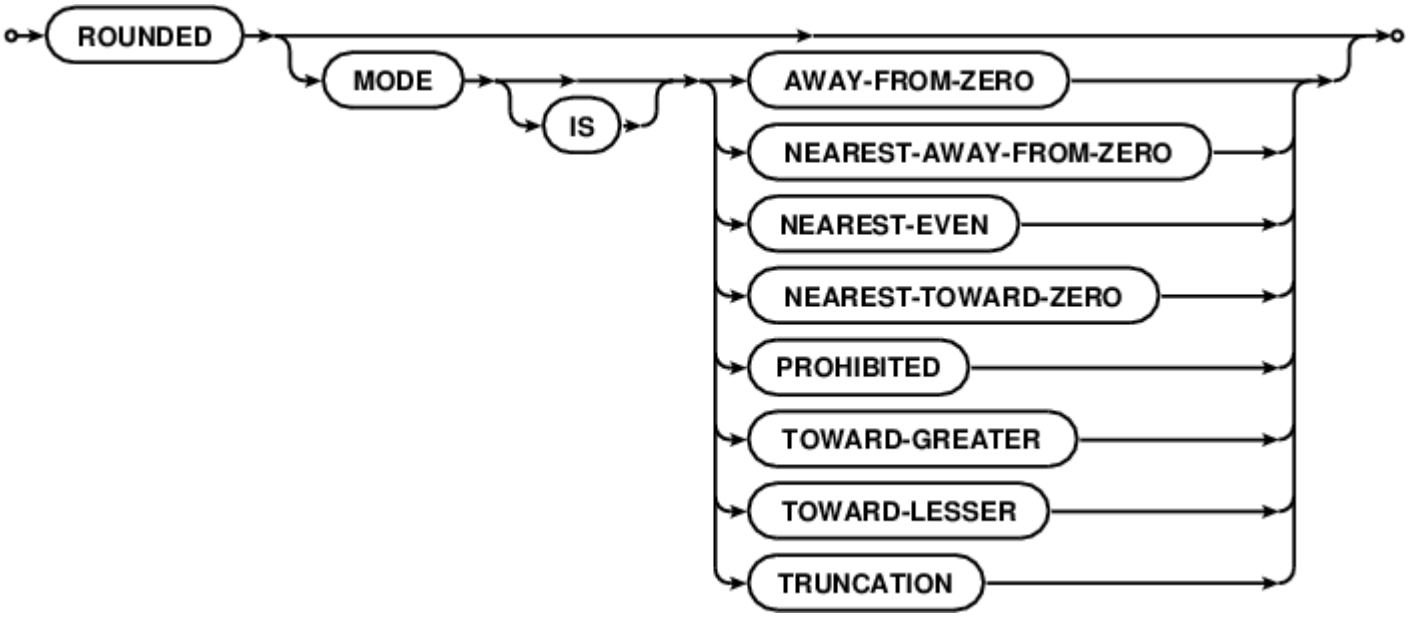
Chapter 9: COMPUTE statement

Remarks

The COMPUTE statement allows for algebraic calculation expressions.



Rounded phrase is



Examples

Advice: Use spaces around all components

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

That is a reference to the variable `var-1`, and not `var - 1`.

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

Recommended, *opinion*.

Read COMPUTE statement online: https://riptutorial.com/cobol/topic/6726/compute-statement

Chapter 10: CONTINUE statement

Remarks

The CONTINUE statement causes the flow of control to continue at the next statement. Not quite a no-op, as it can influence control flow when inside compound statement sequences, in particular IF/THEN/ELSE.

A handy? example is during early development and building with and without debugging aids.

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

That code, while expensive, will allow for formatted memory dumps when the module `CBL_OC_DUMP` is linked into the executable, but will harmlessly fail when it is not. *That trick is only applicable during early stages of development. The expense of a dynamic lookup failure is not something to leave in active code, and those lines should be removed from the source as soon as any initial concerns are satisfied in alpha testing. On first day coding, it can be a handy aid. By second day coding ON EXCEPTION CONTINUE occurrences should be wiped clean.

Examples

Placeholder

This is contrived; but some COBOL programmers may prefer the positive clarity, versus using `NOT` in conditional expressions (especially with the logic error prone `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
```

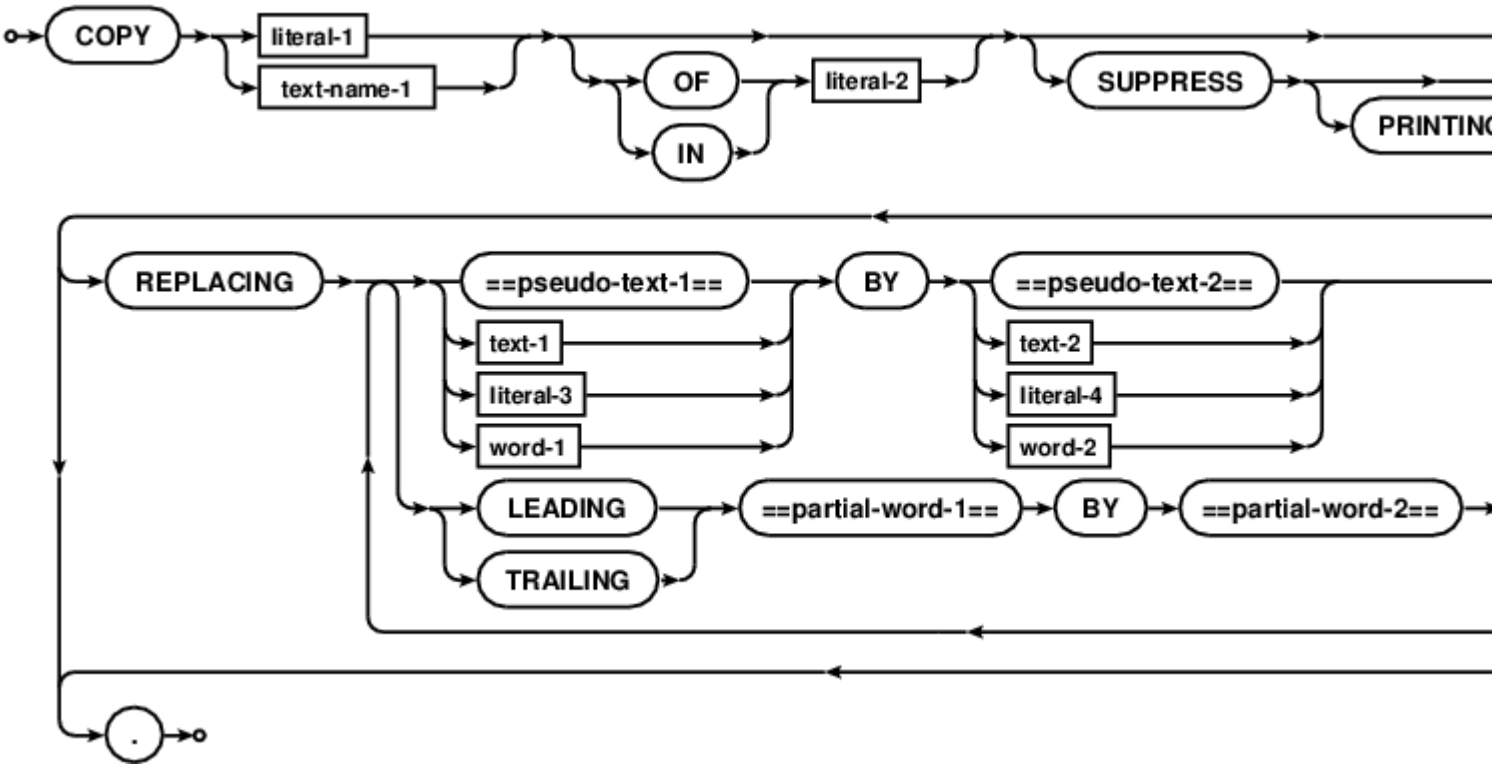
Read CONTINUE statement online: https://riptutorial.com/cobol/topic/6981/continue-statement

Chapter 11: COPY directive

Remarks

The COBOL version of the C `#include` preprocessor directive. *Or, more historically accurate, COBOL came first, developed some 10 years earlier.*

Due to some of the design decisions in COBOL (no arguments for `PERFORM` as the primary reason), many data structure access sequences need to break the DRY principle. Names of structure components need to be repeated in the ENVIRONMENT DIVISION, the DATA DIVISION and possibly many times in the PROCEDURE DIVISION. This is usually handled by adding copybooks. Record declarations and access code are tucked away in separate files and the COPY statement is the only repeated source. A change to the copybook keeps all uses of name spelling and data layout in synch, instead of requiring multiple edits to multiple files when a change occurs.



Examples

COPY record-layout.

program-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.
```

program-two.

```
   DATA DIVISION.

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

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

Read COPY directive online: https://riptutorial.com/cobol/topic/6982/copy-directive

Chapter 12: Data division

Introduction

DATA DIVISION is one of the four parts that make up a COBOL program. It contains statements describing the data used by the program. It consists of four sections: FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION and LINKAGE SECTION.

Examples

Sections in Data Division

SECTIONs in COBOL can be required or optional, depending on which DIVISION they are in.

```
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).
```

In the above example, 01's are level numbers.

Level Number

Level number is used to specify the level of data in a record. They are used to differentiate between elementary items and group items. Elementary items can be grouped together to create group items.

- 01: Record description entry. Group level number is always 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 to 49: Elementary items
- 66: Rename Clause items
- 77: Items which cannot be sub-divided.

- 88: Level 88 is a special level number used to improve the readability of COBOL programs and to improve IF tests. A level 88 looks like a level under another variable, but it's not. It does not have a PICTURE, but it does have a value. A level 88 is always associated with another variable and is a condition name for that variable.

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

Both of the following conditions test whether YES-NO is equal to "Y":

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

A level 88 condition name can be used for an alphanumeric or numeric variable.

Picture Clause

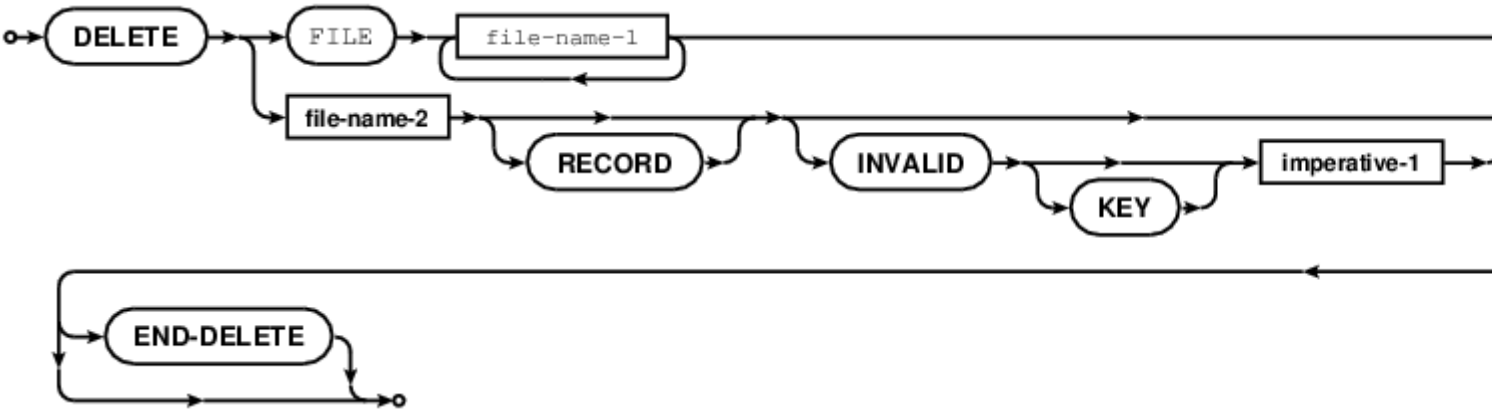
The PICTURE CLAUSE defines two things about a variable: the size of the variable (the number of bytes used in memory for the value) and the type of data that can be stored in the variable.

Read Data division online: https://riptutorial.com/cobol/topic/10859/data-division

Chapter 13: DELETE statement

Remarks

The `DELETE` statement deletes records from mass storage. Some compilers allow the `DELETE` statement to be used with a `FILE` clause, to delete `FD` names (along with any associated indexing structures that may be required by the database management engine in use).



Examples

Delete a record, key in primary key field

```
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
```

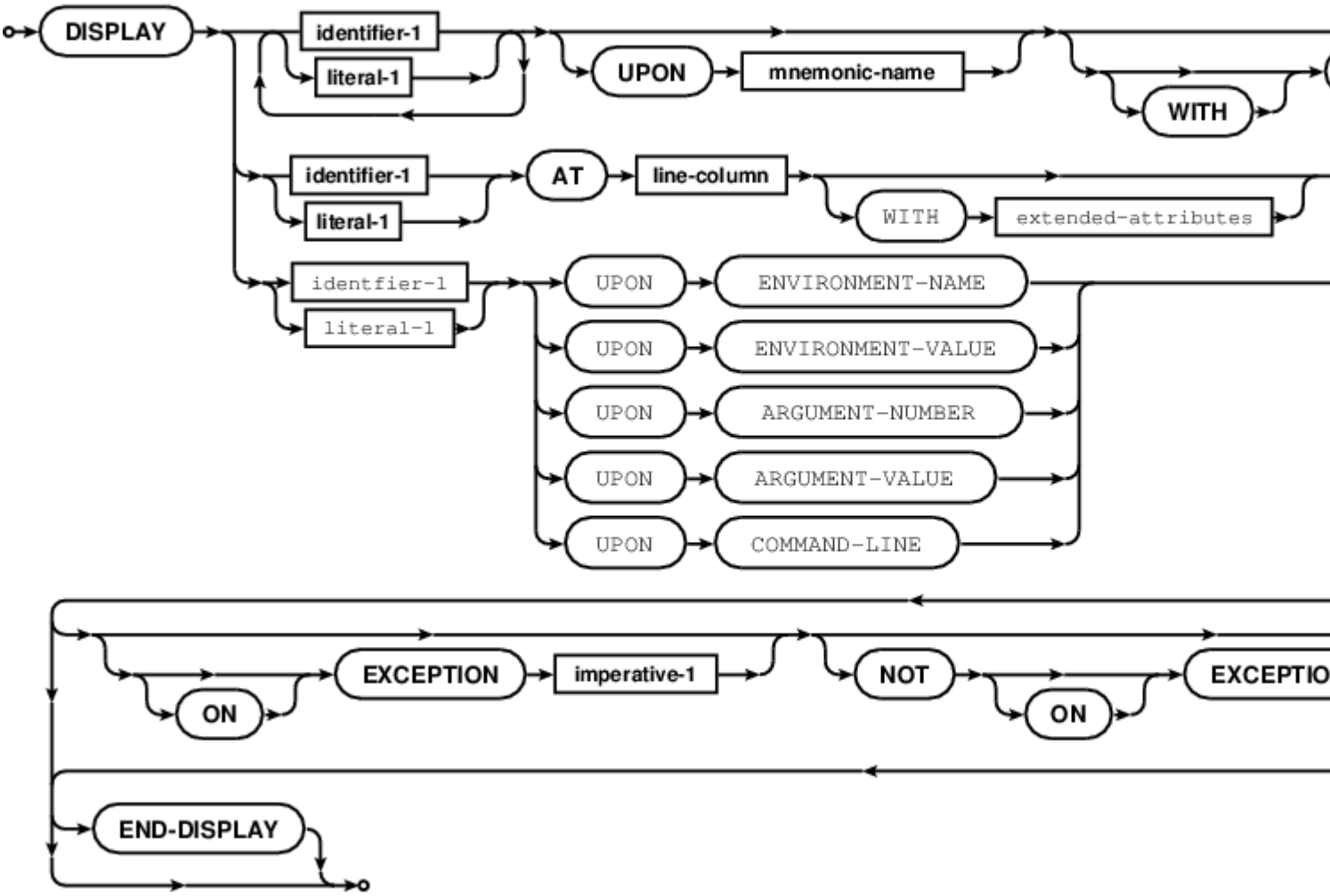
```
       display "Record " keyfield " removed" end-display
end-delete

perform check-delete-status

...
```

Read DELETE statement online: https://riptutorial.com/cobol/topic/7063/delete-statement

Chapter 14: DISPLAY statement

Remarks

The `DISPLAY` statement causes data to be transferred to hardware or software of the operating environment. `DISPLAY` comes in two forms, `UPON device` or for display of `SCREEN` data. Environment variables can also be set with `DISPLAY UPON` in some implementations of COBOL, along with other extensions for data transfer of graphics or other device specific needs.



Examples

DISPLAY UPON

```
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
```

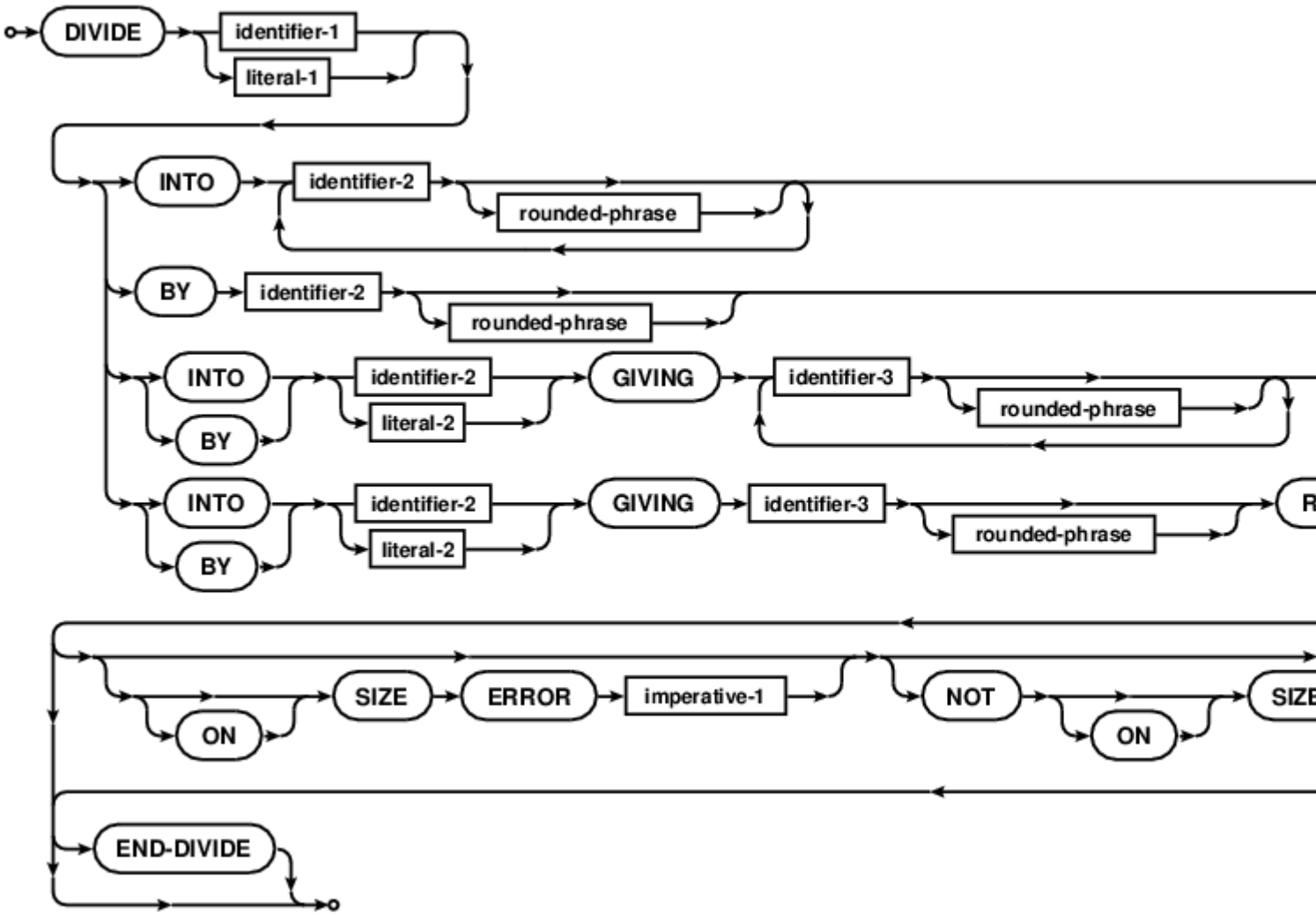
UPON CONSOLE is a default, rarely written. Messages with DISPLAY are one way of debugging COBOL code, but many COBOL programs are transactional in nature, and might not ever interact with a human operator once a job is submitted.

Read DISPLAY statement online: https://riptutorial.com/cobol/topic/7082/display-statement

Chapter 15: DIVIDE statement

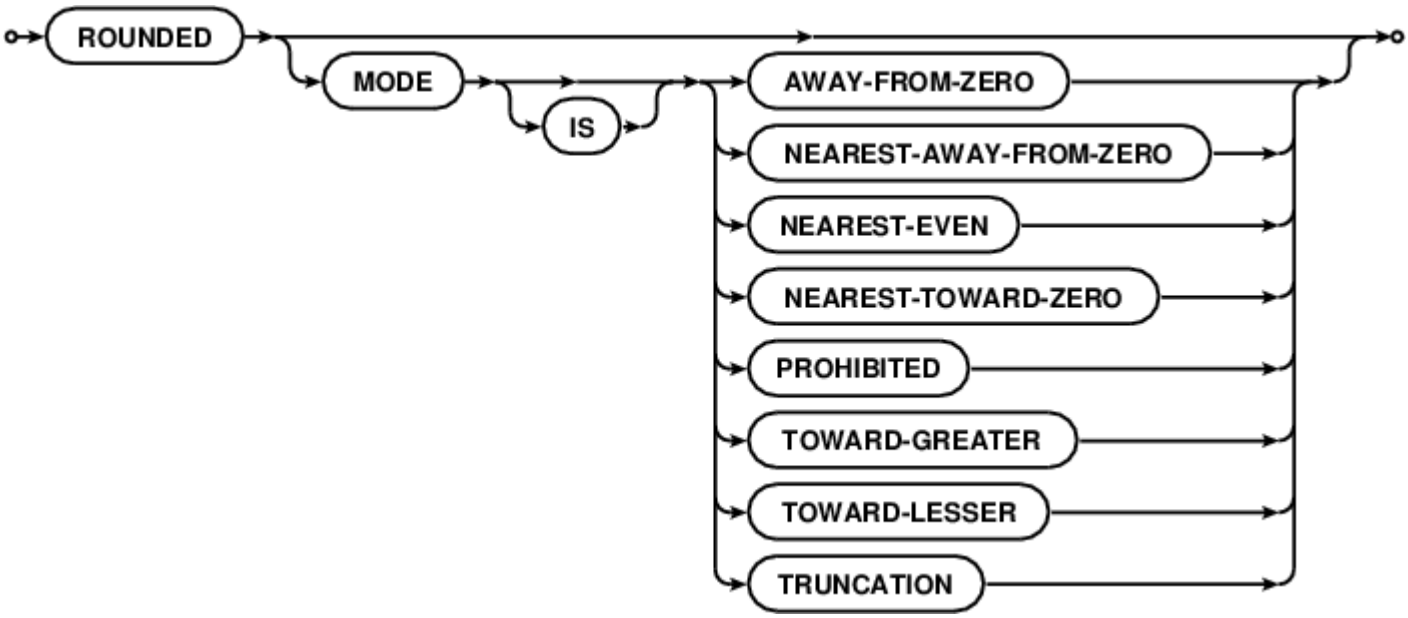
Remarks

The COBOL `DIVIDE` statement divides one numeric item into others setting data items to the quotient and, optionally, the remainder.



`ROUNDED` phrase:

Default is `TRUNCATION` if no rounded phrase specified. Default `ROUNDED` mode is `NEAREST-TOWARD-ZERO` (rounding down) unless other specified. So called *Banker's rounding* is `NEAREST-EVEN`.



Examples

DIVIDE statement formats

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

Data item `b`, `c`, and `d` are changed as `b/a`, `c/a` and `d/a`.

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

Data item `c` is changed as `b/a`.

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

Data item `c` is changed as `a/b`.

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

Data items `q` and `r` are set with results of `b/a`

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

Data items `q` and `r` are set with results of `b/a`

All `DIVIDE` result fields may have `ROUNDED MODE IS` clauses.

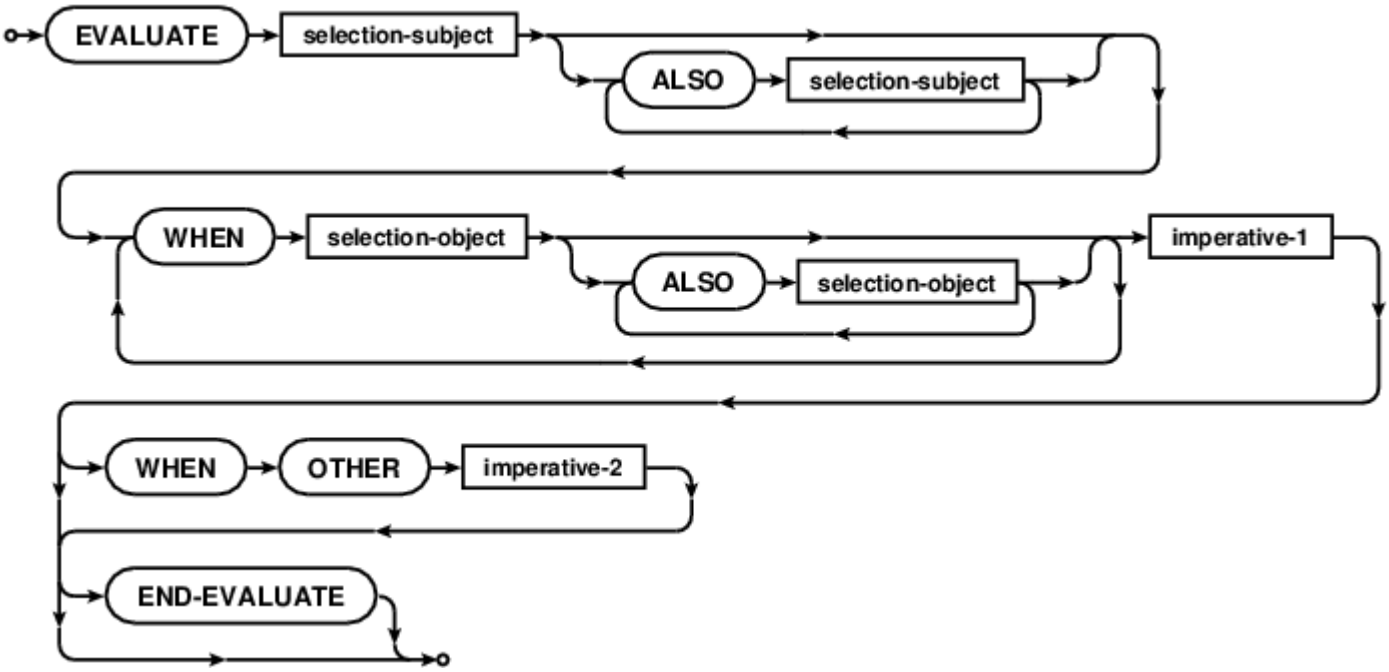
All `DIVIDE` statements may have `ON SIZE ERROR` and `NOT ON SIZE ERROR` declarative statements included to catch invalid results given the type and size of the result fields.

Read DIVIDE statement online: https://riptutorial.com/cobol/topic/7081/divide-statement

Chapter 16: EVALUATE statement

Remarks

The `EVALUATE` statement is a multiple branch, multiple join, conditional test and selection structure.



Examples

A three condition EVALUATE

```
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
```

Read EVALUATE statement online: https://riptutorial.com/cobol/topic/7083/evaluate-statement

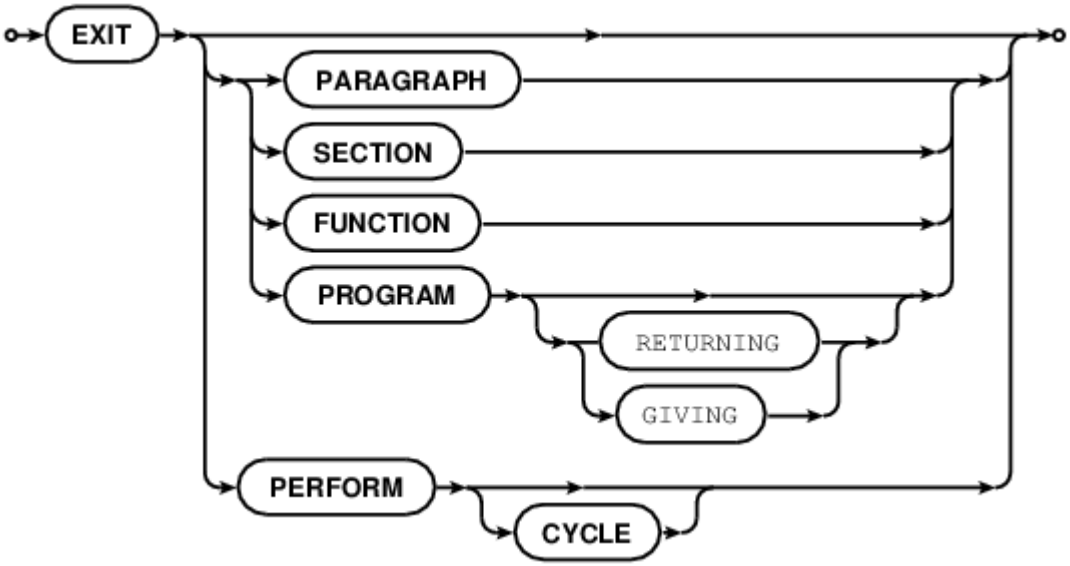
Chapter 17: EXIT statement

Remarks

The COBOL `EXIT` statement is a terminating flow control verb.

`EXIT` comes is a few flavours:

- bare `EXIT` is a common end point for a series of procedures.
- `EXIT PARAGRAPH`, `EXIT SECTION` provides a means of exiting a structured procedure without executing any of the subsequent statements.
- `EXIT FUNCTION`, `EXIT METHOD`, `EXIT PROGRAM` marks the logical end of a module of code.
- `EXIT PERFORM` breaks out of a inline perform loop.
- `EXIT PERFORM CYCLE` causes an inline perform loop to begin the next iteration.



Examples

EXIT statement

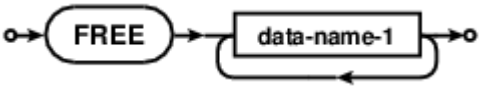
```
PERFORM VARYING counter FROM 1 BY 1 UNTIL counter > 10
    IF debug-override THEN EXIT PERFORM
    IF counter = 5 THEN EXIT PERFORM CYCLE
    PERFORM some-miracle
END-PERFORM
```

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

Chapter 18: FREE statement

Remarks

The `FREE` statement frees allocated memory for one or more identifiers, either by POINTER or from a BASED working storage identifier. Use after FREE is illegal.



Examples

FREE an allocation

```
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
```

Read FREE statement online: https://riptutorial.com/cobol/topic/7162/free-statement

Chapter 19: GENERATE statement

Remarks

The COBOL `GENERATE` statement is an optional statement supported if the compiler includes the Report Writer feature.

Examples

GENERATE a detail line

```
GENERATE detail-line
```

Read GENERATE statement online: https://riptutorial.com/cobol/topic/7161/generate-statement

Chapter 20: GnuCOBOL installation with GNU/Linux

Examples

GNU/Linux install

For most GNU/Linux distributions, a version of `GnuCOBOL` is available in the repositories. `GnuCOBOL` was originally `OpenCOBOL`, rebranded when the project became an official GNU project. Many repositories are still using `open-cobol` as the package name (as of August 2016).

For Fedora, and other RPM based package managers

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

For Debian, Ubuntu and APT based packages

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

This is usually version 1.1 of the compiler suite, and will deal with the compile time and runtime dependencies required when using GnuCOBOL.

From source, (hosted on SourceForge at https://sourceforge.net/projects/open-cobol/) you will need.

- A C compiler suite; `build-essential` (or similar)
- BerkeleyDB and BerkelyDB development headers; `libdb`, `libdb-dev` (or similar names)
- GNU Multi-Precision numeric library; `libgmp`, `libgmp-dev`
- A version of `curses`; `ncurses`, `ncurses-dev`
- The source kit, `gnucobol-1.1.tar.gz` (or better, `gnucobol-2.0.tar.gz`)
- (For changing the compiler sources, `GNU Autoconf` tools are also required).

From a working directory, of your choice:

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

To see the possible configuration options, use:

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

Then

```
prompt$ ./configure
prompt$ make
```

Assuming dependencies are in place and the build succeeds, verify the pre-install with

```
prompt$ make check
```

or

```
prompt$ make checkall
```

That runs internal checks of the compiler (`make check`) and optionally runs tests against the NIST COBOL85 verification suite (`make checkall`). Version 1.1 of OpenCOBOL covers some 9100 NIST tests, recent versions cover more than 9700 test passes. *The NIST COBOL85 testsuite is no longer maintained, but is a very comprehensive and respectable set of tests. COBOL is highly backward compatible, by design intent, but new COBOL 2002 and COBOL 2014 features are not part of the NIST verification suite.*

The internal checks cover some 500 tests and sample code compiles.

If all is well, the last step is

```
prompt$ sudo make install
```

Or, for systems without `sudo`, become the root user for `make install` or use a `./configure` prefix that does not require super user permissions. The default prefix for source builds is `/usr/local`.

If more than one build has occurred on the machine, and local libraries are re-installed, this needs to be followed up with

```
prompt$ sudo ldconfig
```

To ensure that the linker loader `ld` cache is properly refreshed to match the new compiler install.

`cobc` will be ready for use.

`cobc --help` for quick help, `info open-cobol` (or `info gnucobol`) for deeper help, and visit http://open-cobol.sourceforge.net/ for links to the Programmer's Guide and a 1200+ page FAQ document.

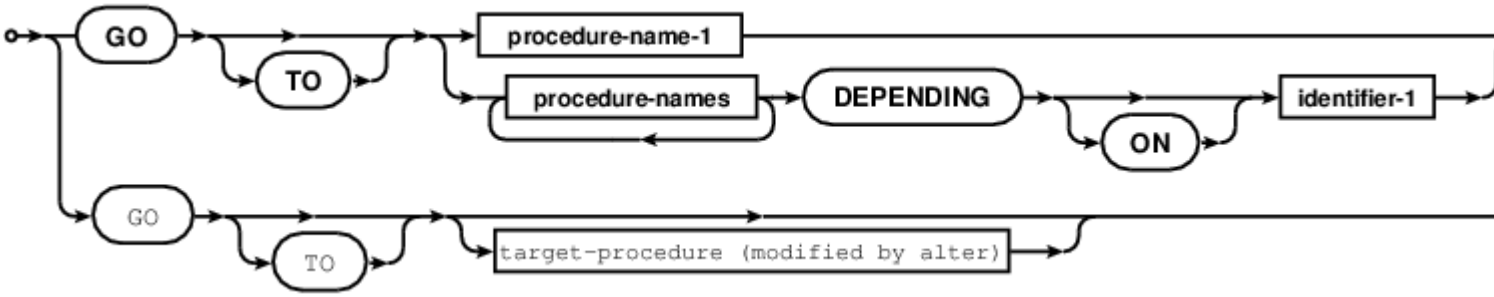
Installation problems, issues or general questions can be posted to the GnuCOBOL project space, in the `Help getting started` Discussion pages on SourceForge.

Read GnuCOBOL installation with GNU/Linux online:
https://riptutorial.com/cobol/topic/5446/gnucobol-installation-with-gnu-linux

Chapter 21: GO TO statement

Remarks

The much beloved `GO TO`. COBOL includes named paragraphs and sections, along with other labels, and any of them can be the target of a `GO` statement.



Examples

GO statement

```
GO TO label

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

GO TO label OF section

GO.
```

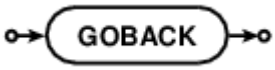
The last line example indicates that an `ALTER` statement is in play, and another part of the code will specify which actual `label` is the target of the jump.

Read GO TO statement online: https://riptutorial.com/cobol/topic/7163/go-to-statement

Chapter 22: GOBACK statement

Remarks

The COBOL `GOBACK` statement is a return. Unlike `EXIT PROGRAM`, or `STOP RUN`, `GOBACK` always returns one level. If the current module is "main", `GOBACK` will return to the operating system. If the current module is a subprogram, `GOBACK` will return to the statement after a call.



Examples

GOBACK

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

...

call "subprog"
goback.
```

The first `GOBACK` above will return from subprog. Assuming the second is inside the main procedure, `GOBACK` will return to the operating system.

Read GOBACK statement online: https://riptutorial.com/cobol/topic/7173/goback-statement

Chapter 23: How does the computational work in cobol?

Introduction

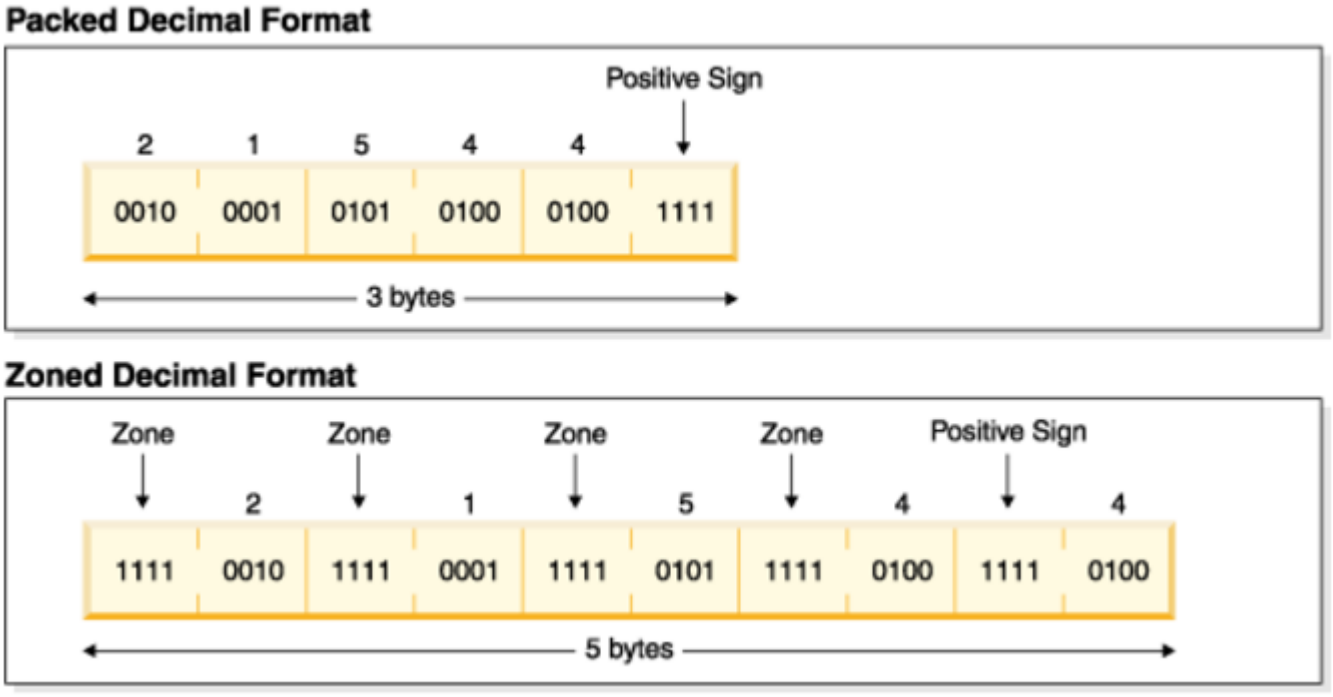
Computational clause is used to describe type of storage used in COBOL. It is used for 3 ways: COMP-1, COMP-2 and COMP-3. The most common form of computational is COMP-3. It frequently is just called "COMP" by programmers.

Examples

COMP-3

Data item is stored in packed decimal format in COMP-3. Packed-decimal format means that each byte of storage (except for the low order byte) can contain two decimal numbers. The low-order byte contains one digit in the leftmost portion and the sign (positive or negative) in the rightmost portion.

"Zoned decimal format" in the image below is the default storage for a number in COBOL.




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

Computational storage is frequently used to reduce the size of a file.

Common implementations

How comp, comp-1 ... comp-5 are implemented is implementation dependent.

```
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
```

Ibm Compilers normally support Comp, Comp-4, Comp-5 in sizes of 2,4,8 bytes. GNU Cobolo support sizes of 1,2,4,8.

Comp-1, Comp-2 fields are defined without a picture clause:

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

For other Comp's are picture is entered:

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

Read How does the computational work in cobol? online:
https://riptutorial.com/cobol/topic/10873/how-does-the-computational-work-in-cobol-

Chapter 24: IF statement

Remarks

The conditional expression and selection statement. Use of explicit scope terminators is recommended. COBOL conditional expressions allow shortforms, where the current identifier (and conditional) is assumed through multiple condition tests, unless explicitly given.

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

is equivalent to

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

Examples

IF with shortform conditionals

```
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
```

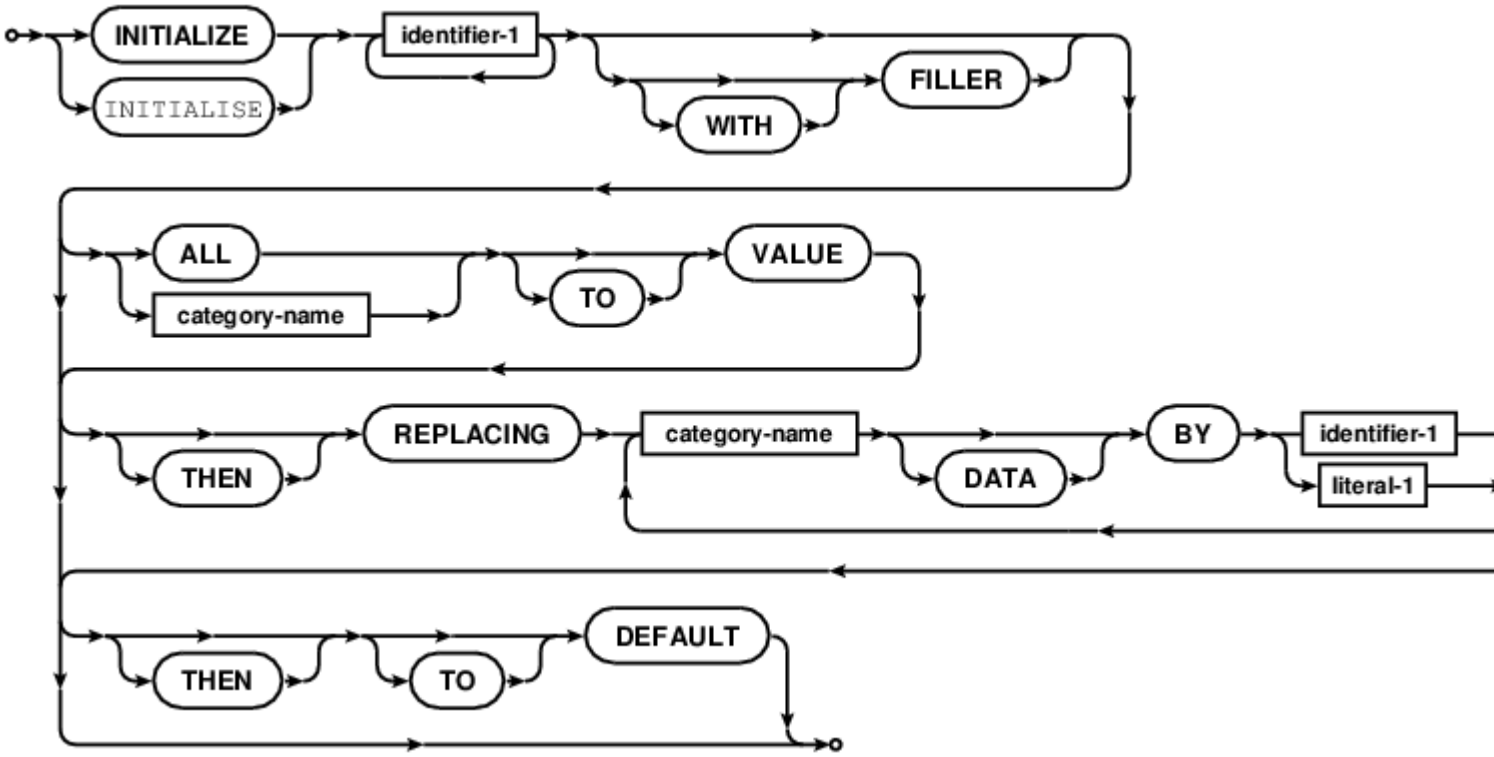
`IF` statements can be terminated with full stop or explicit scope terminator `END-IF`. *Use of periods for scope termination is no longer recommended.* Full stops mean just that in the case of nested IF, all nesting is terminated at the first full stop `.`, and any subsequent code will be outside the IF block.

Read IF statement online: https://riptutorial.com/cobol/topic/7174/if-statement

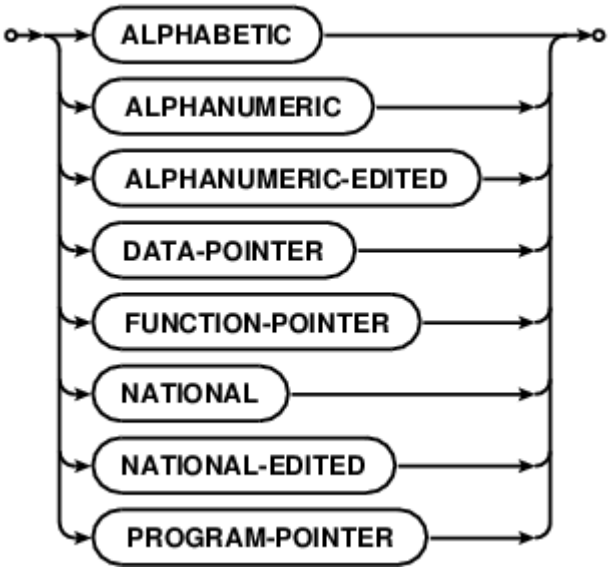
Chapter 25: INITIALIZE statement

Remarks

The `INITIALIZE` statement sets selected data to specified values.



Where `category-name` is:



Examples

Various INITIALIZE clauses

```
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
```

Giving:

```
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.
```

Read INITIALIZE statement online: https://riptutorial.com/cobol/topic/7179/initialize-statement

Chapter 26: INITIATE statement

Remarks

The `INITIATE` statement initializes internal `Report Writer` control fields. Most of a report writer setup occurs in the `DATA DIVISION` with very brief `PROCEDURE DIVISION` statements. Once initialized, `GENERATE` does all the hard work of control break and paging of reports.

Examples

INITIATE reporting control variables

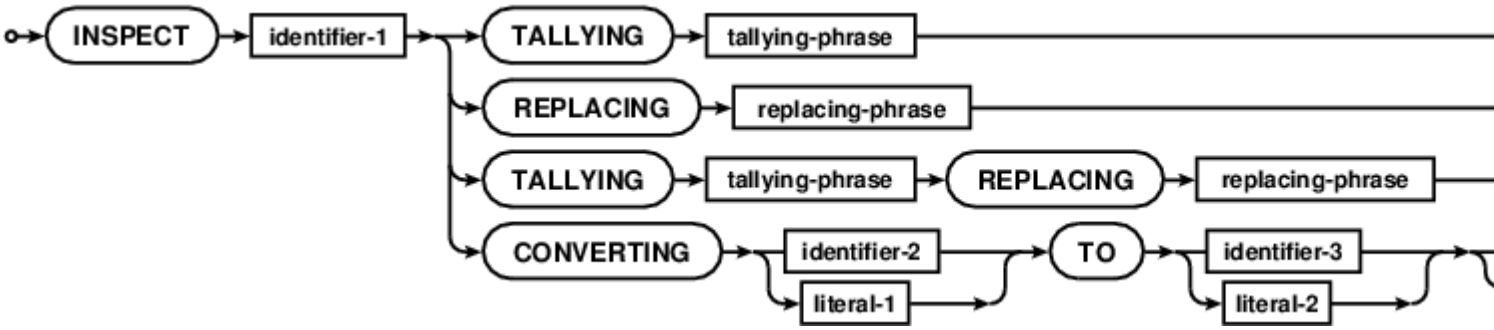
```
INITIATE report-1 report-2
```

Read INITIATE statement online: https://riptutorial.com/cobol/topic/7180/initiate-statement

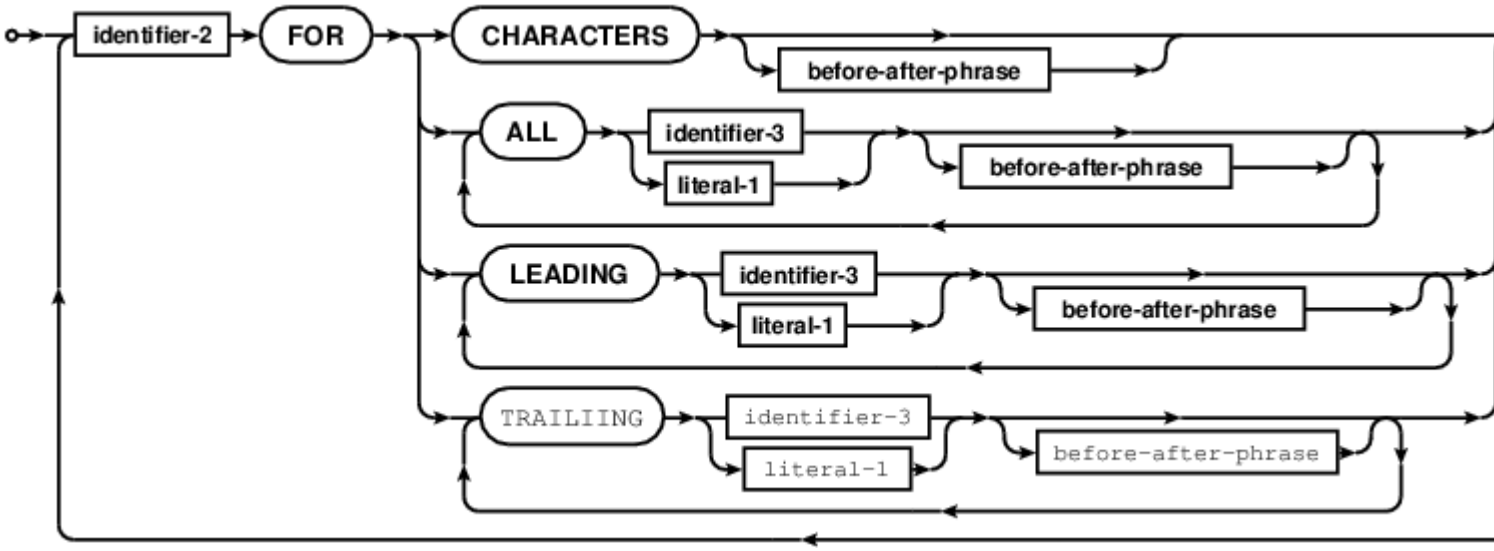
Chapter 27: INSPECT statement

Remarks

The `INSPECT` statement is a scan and replace verb in COBOL.



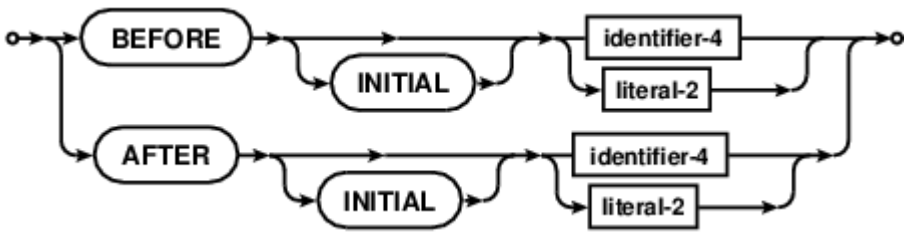
Where `tallying-phrase` is:



`replacing-phrase` is:

```
missing image
```

`before-after-phrase` is:



Examples

INSPECT reformatting a date 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.
```

Giving:

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

Read INSPECT statement online: https://riptutorial.com/cobol/topic/7182/inspect-statement

Chapter 28: Intrinsic Functions

Introduction

Intrinsic Functions are included in the COBOL standard as a set of functions that return values from a specific algorithm, given zero or more arguments. These intrinsic functions are provided as a facility of the compiler and runtime system. The return items are temporary COBOL fields, and can be character data, bit fields, or numeric values.

Examples include trigonometric functions, date time routines, data type conversions, standard deviation, and other support algorithms.

Remarks

COBOL 2014 lists the following standard Intrinsic Functions:

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

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

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

GnuCOBOL adds

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

The keyword `FUNCTION` is required unless source (or compile time option) includes

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

Where `ALL INTRINSIC` can be a list of functions to be used without the `FUNCTION` prefix in `PROCEDURE DIVISION` statements.

The `LENGTH` function has a sorted history. Some compilers include a `LENGTH` reserved word. For GnuCOBOL, this reserved word is only recognized when used in the phrase `LENGTH OF`, the `OF` token is required to disambiguate the function from the older reserved word extension.

Examples

FUNCTION TRIM example

```
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) ":"
```

Showing

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

UPPER-CASE

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

Output

```
HELLO WORLD!
```

LOWER-CASE function

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

Output

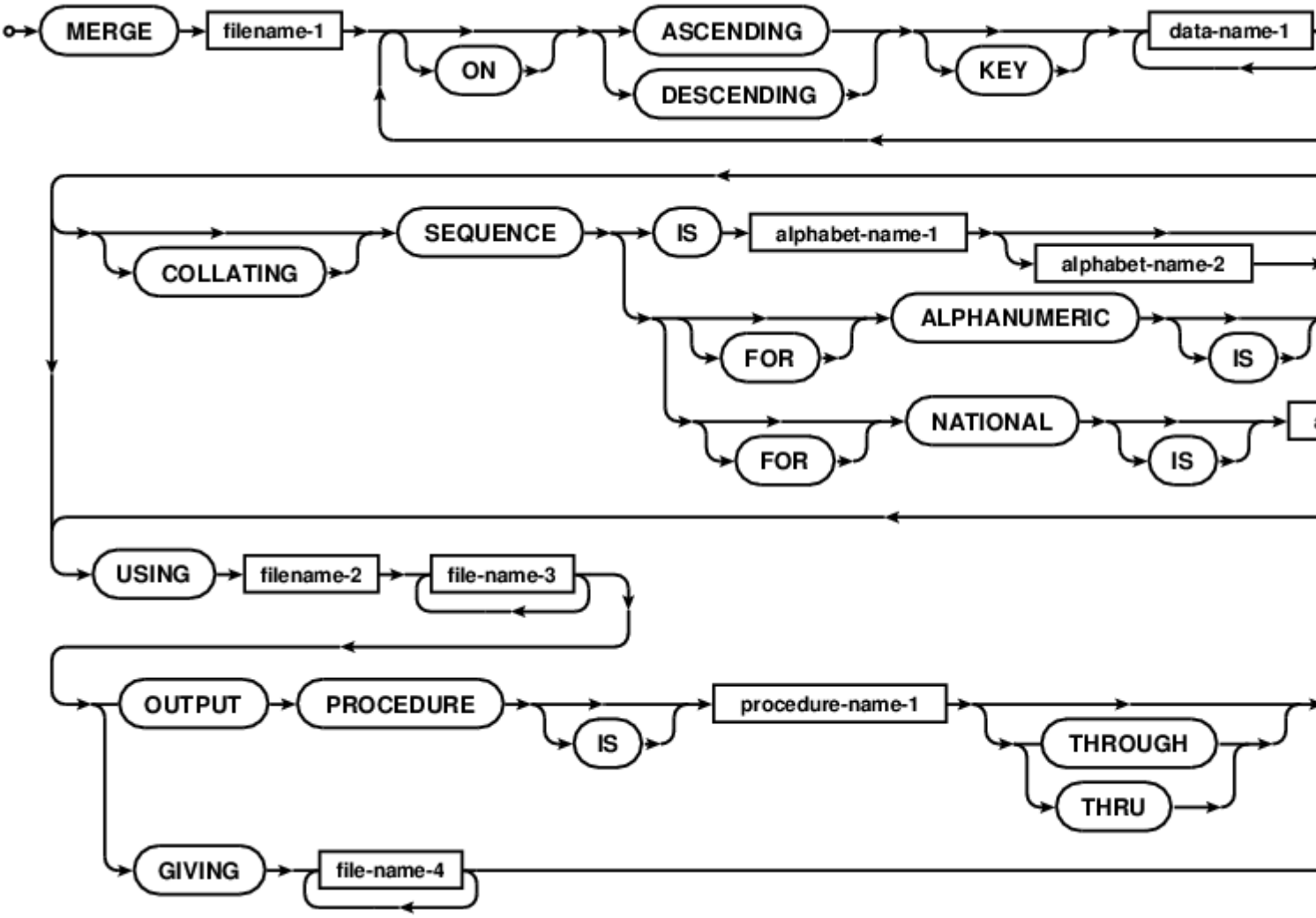
```
hello world!
```

Read Intrinsic Functions online: https://riptutorial.com/cobol/topic/7580/intrinsic-functions

Chapter 29: MERGE statement

Remarks

The MERGE statement will merge one or more like formatted COBOL data files into a single output file. The programmer can assume control over the`OUTPUT PROCEDURE`, which uses the `RELEASE` statement, or use internal COBOL runtime mechanisms with the `GIVING` clause.



Examples

MERGE regional data into 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.
```

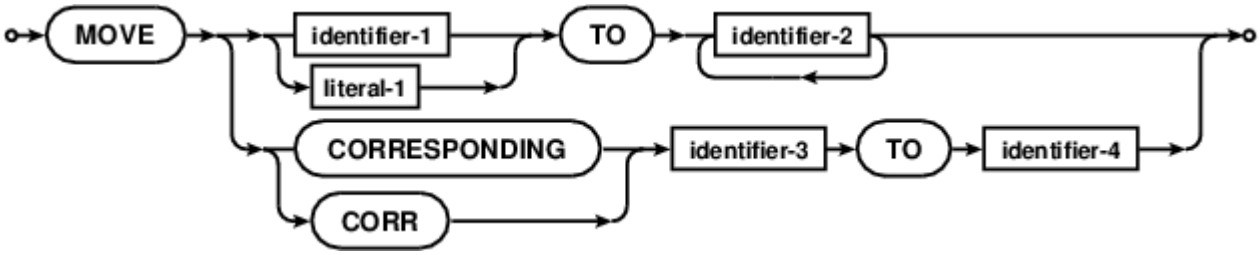
```
       end program gnucobol-merge-sample.
```

Read MERGE statement online: https://riptutorial.com/cobol/topic/7183/merge-statement

Chapter 30: MOVE statement

Remarks

`MOVE` is the workhorse of COBOL. Data is moved from literal or identifier to one or more identifiers. COBOL has a distinction between *elementary* and *group* MOVE. Elementary data is type converted from source to destination. Group data is moved as a byte array, without regard to field types with a structure. Numeric fields are moved from right to left, high order digit truncation with zero fill (normally). Alphanumeric character data is moved left to right, right end character truncation with space fill. There are quite a few rules on how `MOVE` goes about its business, with both BINARY and PICTURE DISPLAY data forms, and group hierarchies all accounted for.



Examples

Some MOVE details, there are many

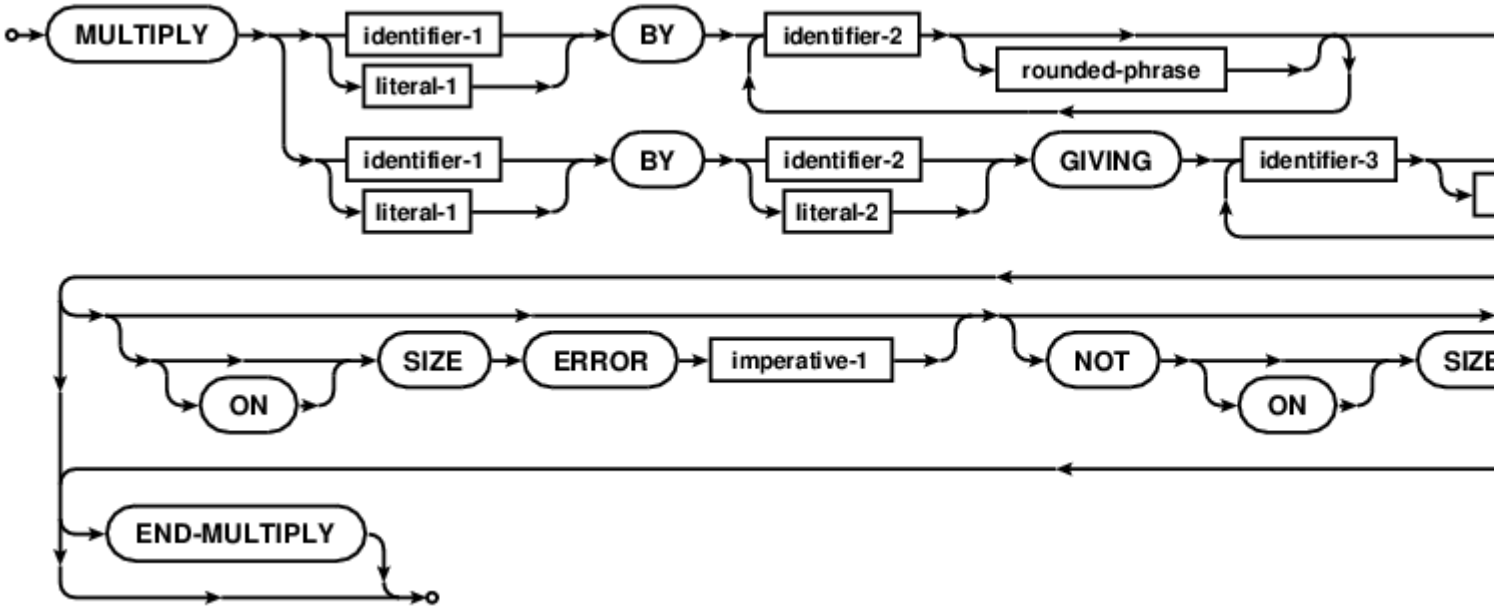
```
01 a PIC 9.
01 b PIC 99.
01 c PIC 999.

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
```

Read MOVE statement online: https://riptutorial.com/cobol/topic/7263/move-statement

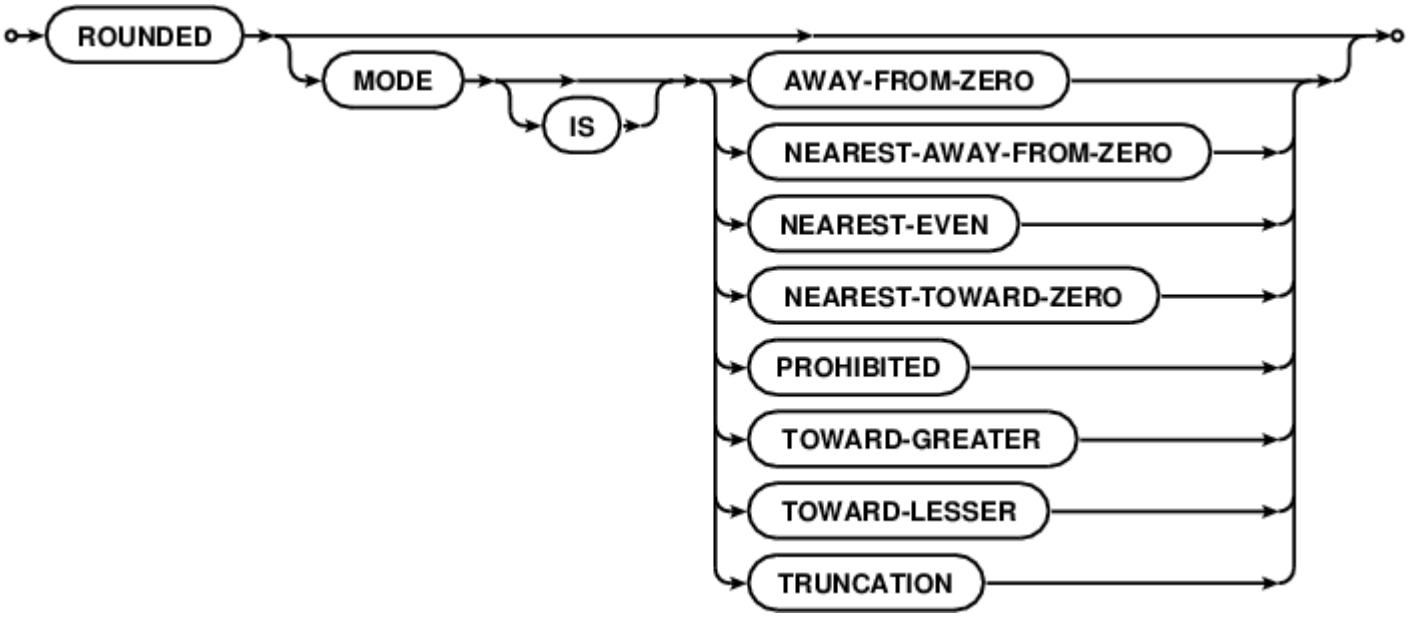
Chapter 31: MULTIPLY statement

Remarks

The `MULTIPLY` statement multiplies numeric data setting the result to one or more identifiers of numeric type.



Where `rounded-phrase` is



Examples

Some MULTIPLY formats

```
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
```

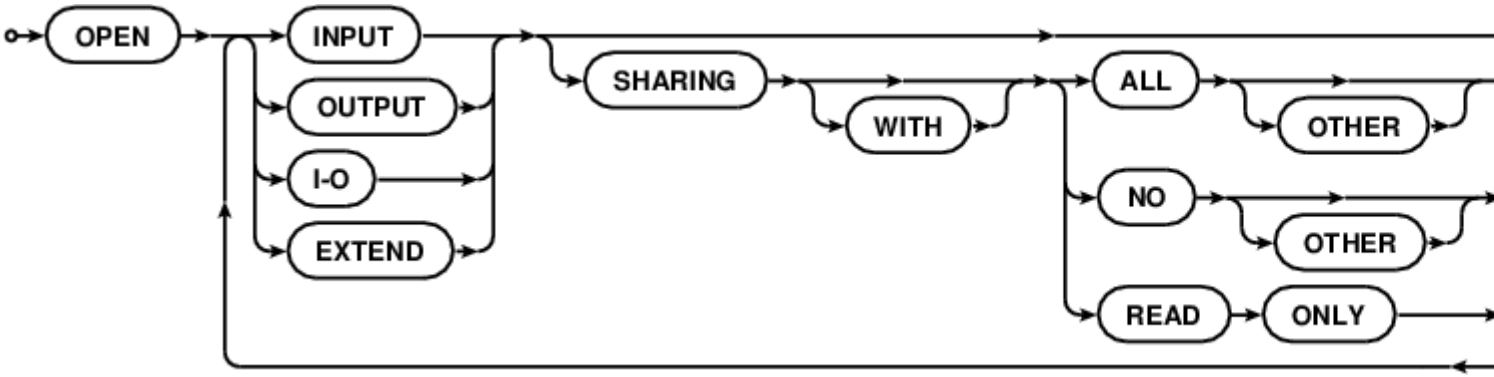
Read MULTIPLY statement online: https://riptutorial.com/cobol/topic/7264/multiply-statement

Chapter 32: OPEN statement

Remarks

The COBOL `OPEN` statement initiates file processing. File resources in COBOL are defined in the `ENVIRONMENT DIVISION`, named in `FD` (File Descriptor) paragraphs. These fd names are used to access physical disk files and various options are specified in a `SELECT` clauses in the `FILE-CONTROL` paragraph of the `INPUT-OUTPUT SECTION`. A programmer is expected to test a `FILE STATUS` identifier for status and error codes.

Modes include `INPUT`, `OUTPUT`, `I-O` and `EXTEND`.



Examples

OPEN sample, with LINAGE mini report

```
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.
```

Read OPEN statement online: https://riptutorial.com/cobol/topic/7288/open-statement

Chapter 33: PERFORM statement

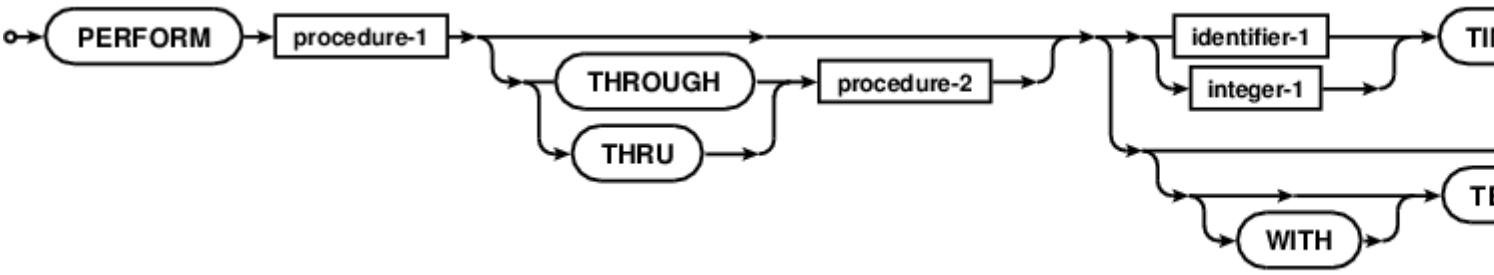
Remarks

The PERFORM statement transfers control to one or more procedures and returns control implicitly when the sequence completes. PERFORM can also be used for inline loops withing the scope of the PERFORM.

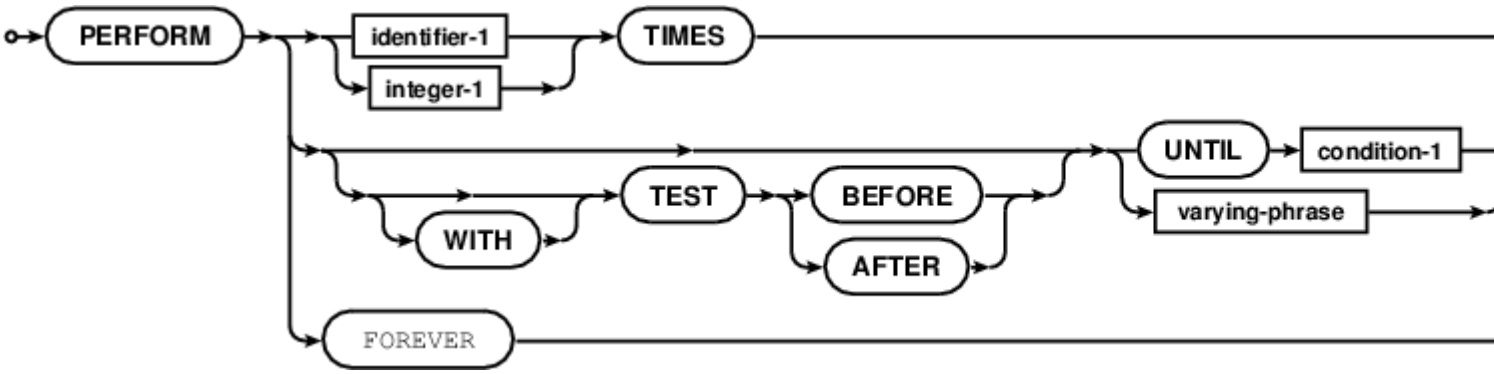
The `VARYING` phrase allows for nesting with one or more `AFTER` clauses, and the conditional test can be `BEFORE` (default) or `AFTER` each loop.

The `THRU` clause of a procedural perform assumes sequential top down control flow from `procedure-1` through the end of `procedure-2`. *THRU* is a contentious issue, and many programmers prefer `PERFORM` by `SECTION` rather than using `THRU` paragraphs. Some shops may mandate `PERFORM THRU` with an explicit exit point paragraph, others may ban the use of `THRU` finding it more difficult to debug.

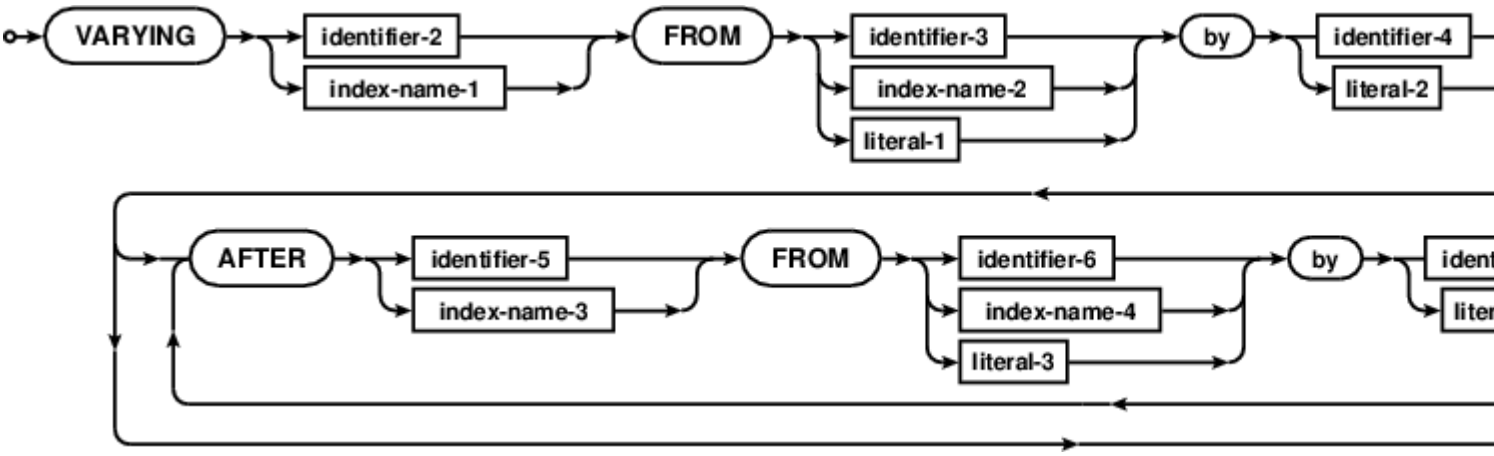
Procedural perform:



Inline perform:



Where `varying-phrase` is:



Examples

Inline PERFORM VARYING

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

Procedural PERFORM

```
PERFORM some-paragraph
```

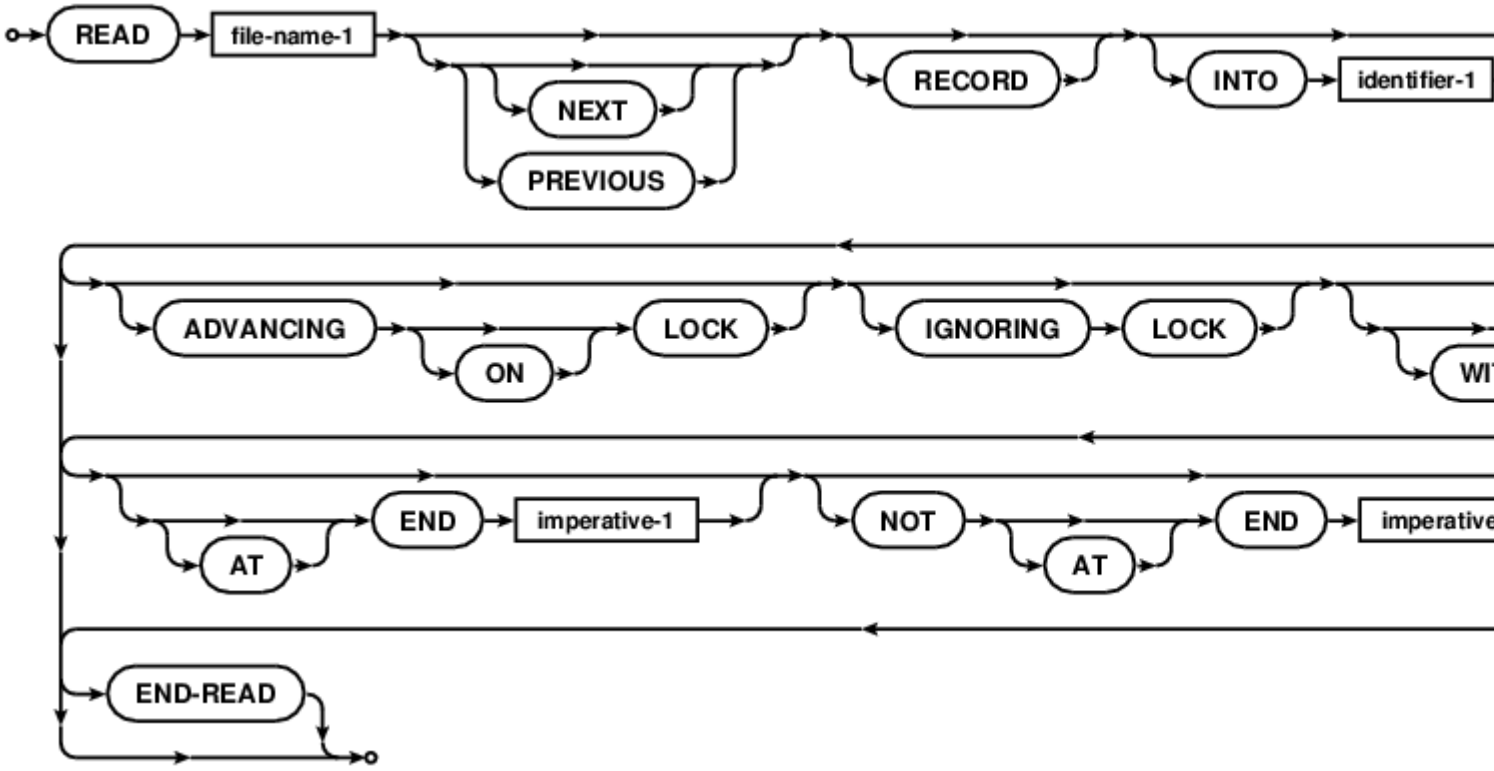
Read PERFORM statement online: https://riptutorial.com/cobol/topic/7334/perform-statement

Chapter 34: READ statement

Remarks

The `READ` statement is a staple of COBOL transaction processing programming. Reads data from external storage into working store. With or without locks or sharing, sequentially, by random access, or by key. Declarative clauses for `AT END` may also be specified, but some programmers prefer explicit `FILE STATUS` testing.

As each file resource may contain any type of record in any given slot, COBOL is a "read a file", "write a record" language, `READ` takes a filename (FD) and it is up to the programmer to place the record in an appropriate structure if heterogeneous data is saved in the file.



Examples

Simple READ from FD

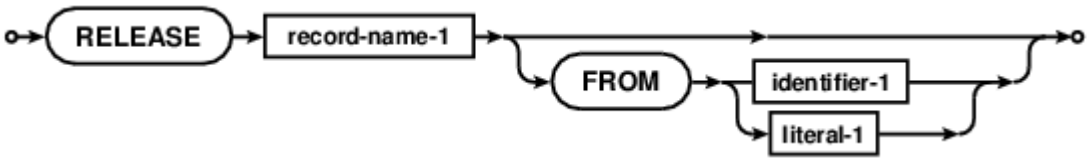
```
READ data-file
```

Read READ statement online: https://riptutorial.com/cobol/topic/7336/read-statement

Chapter 35: RELEASE statement

Remarks

The `RELEASE` statement is used to give records to the COBOL `SORT` algorithm under programmer controlled conditions.



Examples

RELEASE a record to a SORT INPUT PROCEDURE

This is a contrived sample. It sorts records based on an `ALPHABET` that has upper and lower case characters together, with `A` and `a` swapped compared to the other letters. This was done on purpose to demonstrate the possibilities. The SORT algorithm reader retrieves records using `RELEASE` in the `INPUT PROCEDURE`. The `OUTPUT PROCEDURE` uses `RETURN` for the `SORT` algorithm writer.

```
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.
```

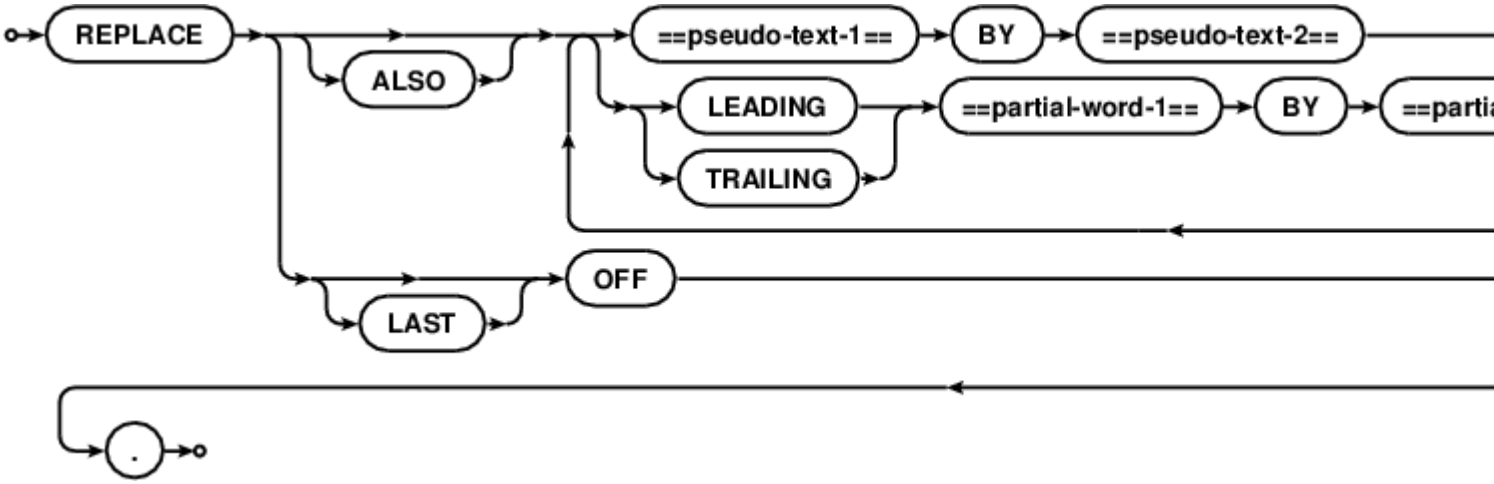
```
       file section.
       fd sort-in.
          01 in-rec        pic x(255).
       fd sort-out.
          01 out-rec       pic x(255).
       sd sort-work.
          01 work-rec      pic x(255).

       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.
```

Read RELEASE statement online: https://riptutorial.com/cobol/topic/7337/release-statement

Chapter 36: REPLACE directive

Remarks

The `REPLACE` directive is part of the COBOL standard preprocessor. Replacements are made before compilation begins.



Examples

REPLACE text manipulation sample

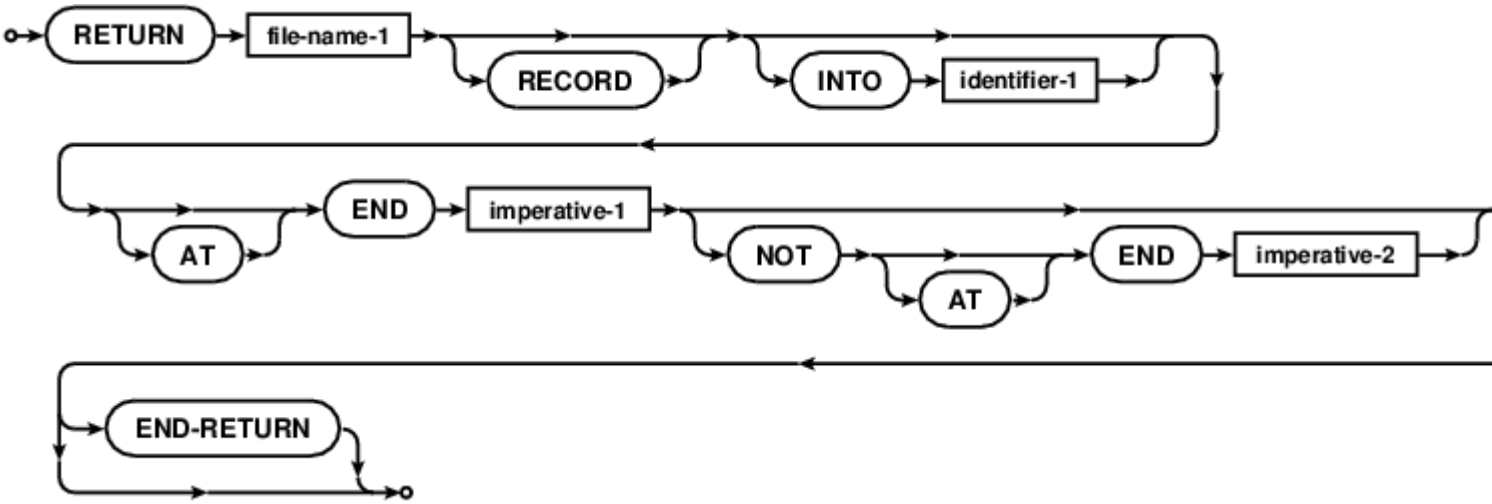
```
REPLACE ==magic-number== BY ==65535==.
```

Read REPLACE directive online: https://riptutorial.com/cobol/topic/7459/replace-directive

Chapter 37: RETURN statement

Remarks

The `RETURN` statement controls when data is sent to the internal COBOL sort algorithm writer, as part of an `OUTPUT PROCEDURE`. Post sort data can be transformed under programmer control before being returned and written to the output file by the sort algorithm.



Examples

RETURN a record to SORT OUTPUT PROCEDURE

This is a seedwork sample. The `SORT OUTPUT PROCEDURE` could manipulate the sorted records before they are returned to the write portion of the internal COBOL sort algorithm. In this case, no transformation is done, `work-rec` is directly moved to `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
```

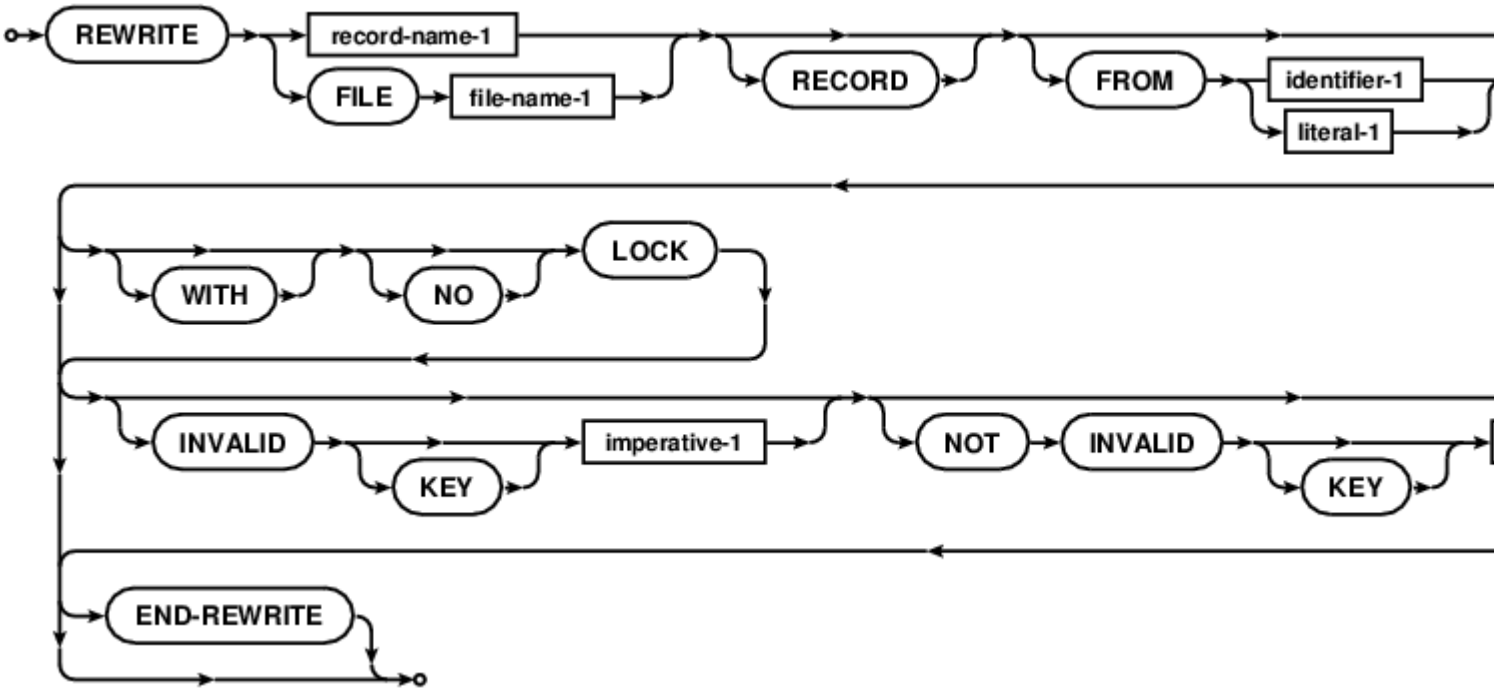
```
                at end move high-value to loop-flag
            end-return
        end-perform
        close sort-out
        .

        exit program.
        end program sorting.
```

Read RETURN statement online: https://riptutorial.com/cobol/topic/7338/return-statement

Chapter 38: REWRITE statement

Remarks

The REWRITE statement logically replaces existing records on mass storage.



Examples

REWRITE of records in a RELATIVE access file

```
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.
```

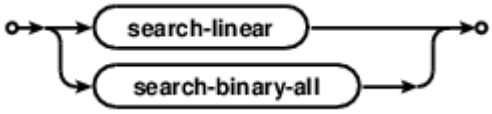
Read REWRITE statement online: https://riptutorial.com/cobol/topic/7460/rewrite-statement

Chapter 39: SEARCH statement

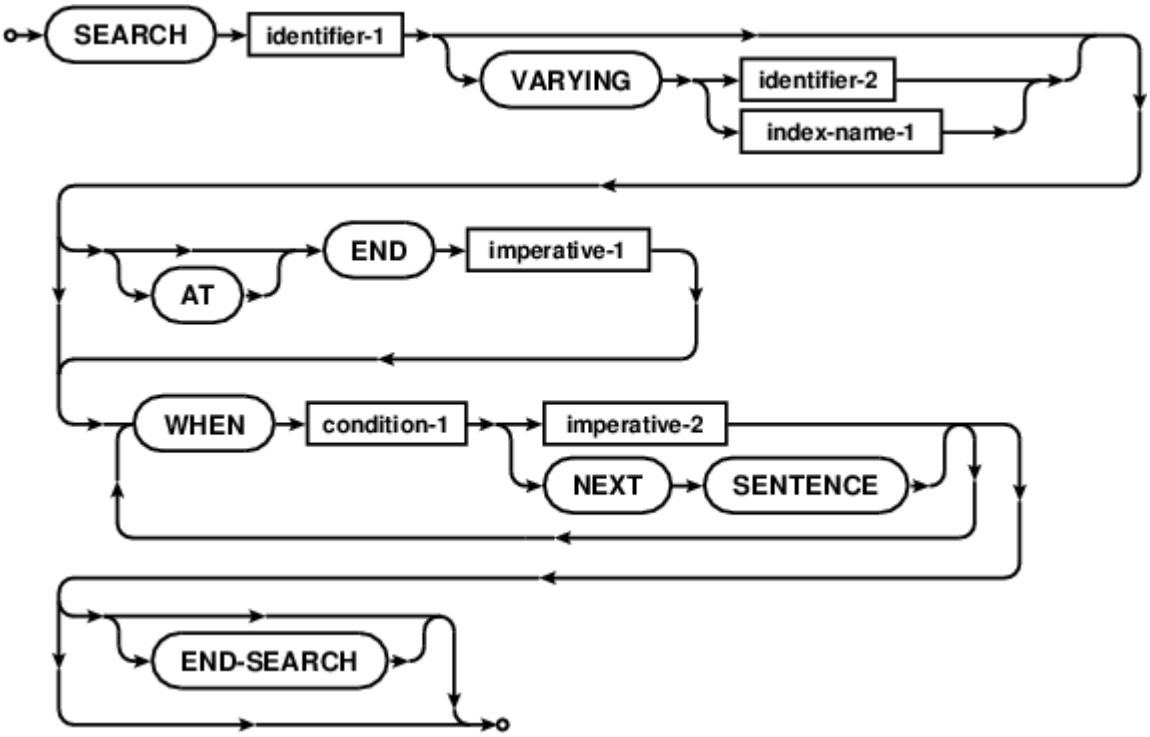
Remarks

The COBOL `SEARCH` statement comes in two forms. Linear top to bottom `SEARCH` and a binary `SEARCH ALL` algorithm. Binary SEARCH ALL assumes a sorted table suitable for a binary search with no elements out of order.

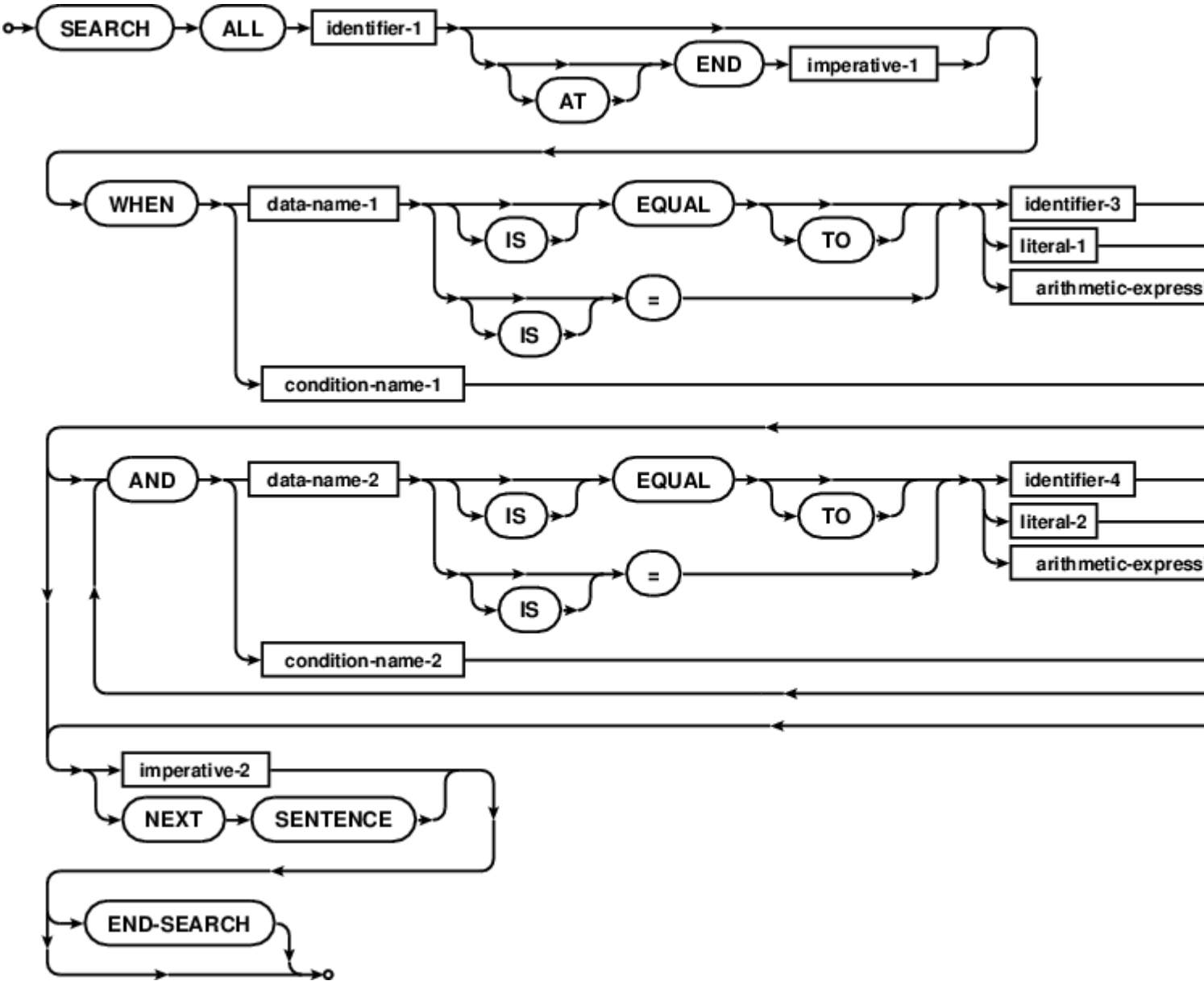
SEARCH statement



Linear SEARCH



Binary SEARCH ALL



Examples

Linear SEARCH

```
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.
```

Binary SEARCH ALL

```
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
```

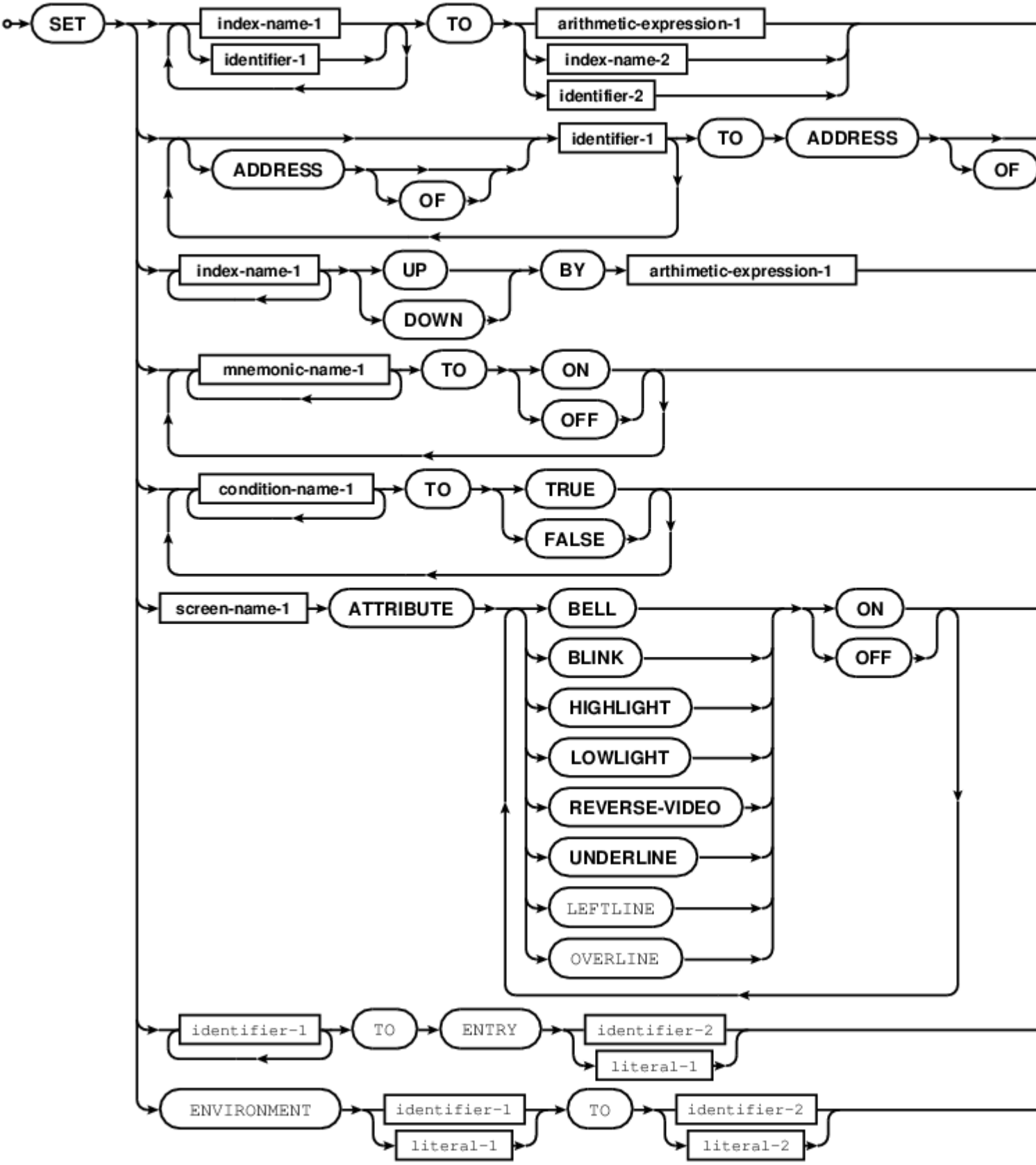
```
       read wordfile
           at end set at-eof to true
       end-read

       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.
```

Read SEARCH statement online: https://riptutorial.com/cobol/topic/7462/search-statement

Chapter 40: SET statement

Remarks

The COBOL `SET` statement sets values, and operating environment data. It can be argued that `SET` was overused by the committee, as it has over a dozen documented syntax formats.



Examples

SET pointer example

```
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
```

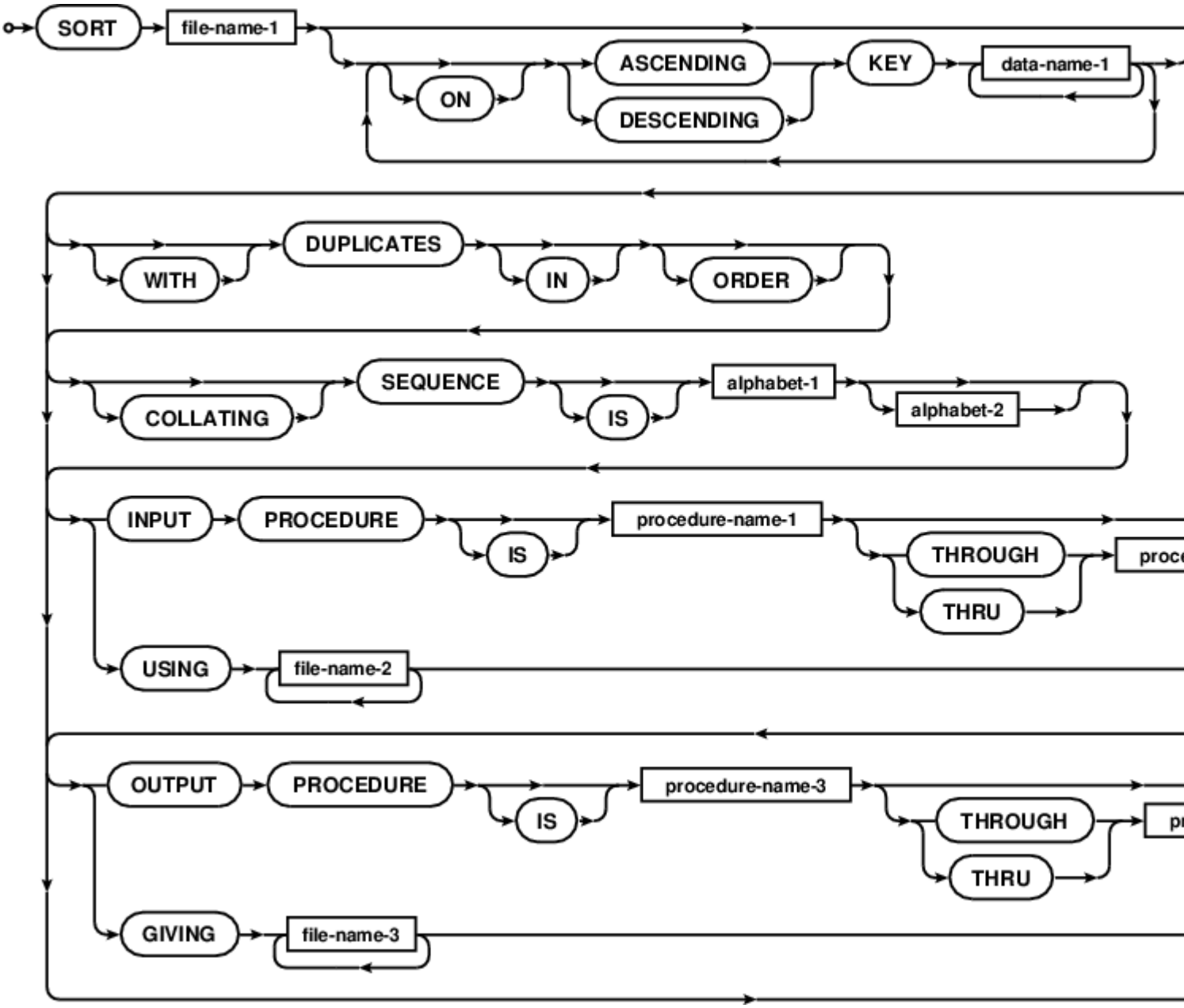
Read SET statement online: https://riptutorial.com/cobol/topic/7461/set-statement

Chapter 41: SORT statement

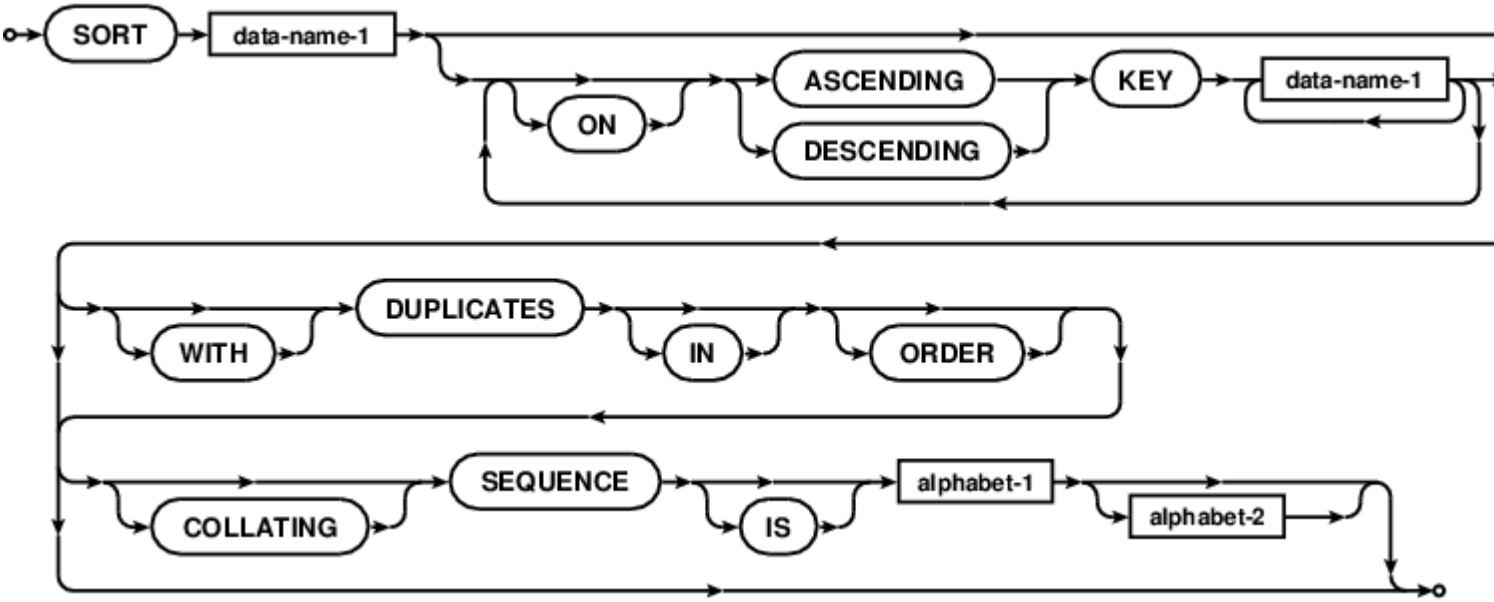
Remarks

The COBOL `SORT` statement can be used to sort files and tables in working storage.

SORT file



SORT table



Examples

Sorting standard in to standard out

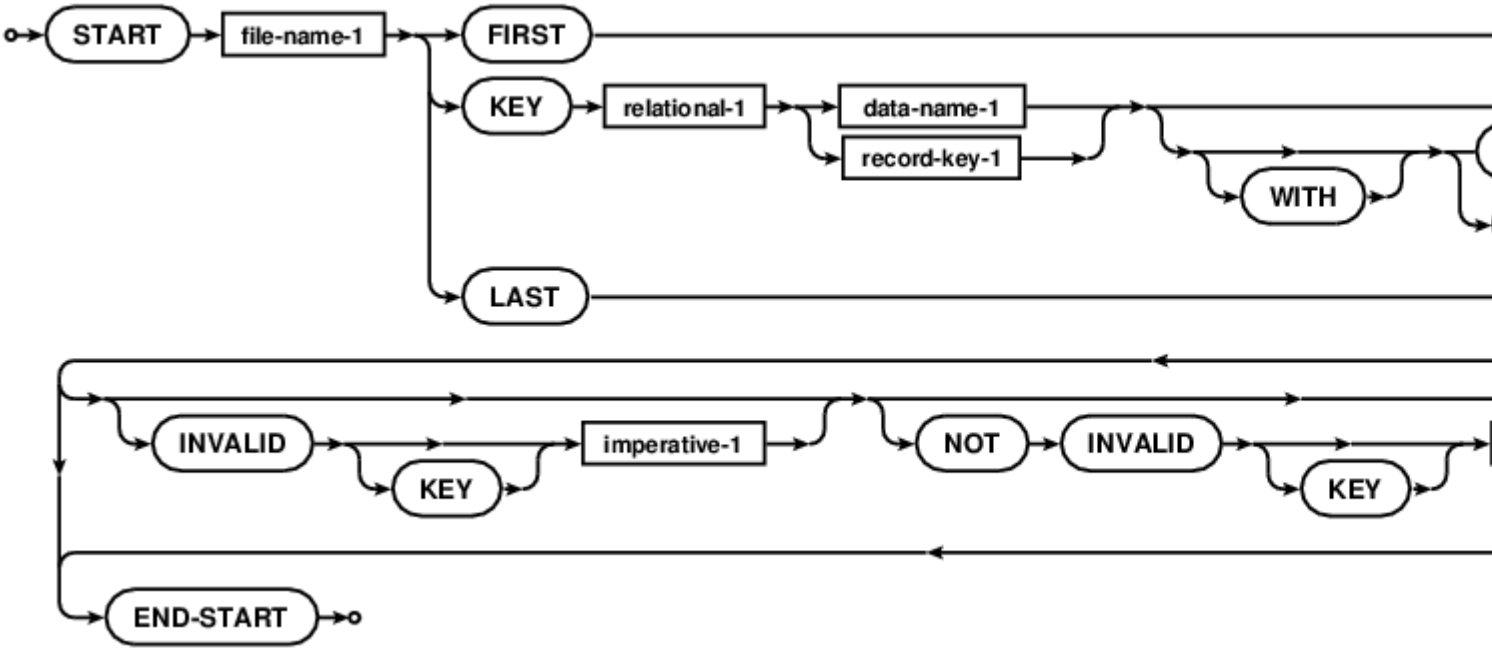
```
GCobol* GnuCOBOL SORT verb example using standard in and standard out
       identification division.
       program-id. sorting.

       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.
```

Read SORT statement online: https://riptutorial.com/cobol/topic/7463/sort-statement

Chapter 42: START statement

Remarks

The `START` statement provides a way to position a read in a file for subsequent sequential retrieval (by key).



The key relational can include (but is not limited to):

- KEY IS GREATER THAN
- KEY IS >
- KEY IS LESS THAN
- KEY IS <
- KEY IS EQUAL TO
- KEY IS =
- KEY IS NOT GREATER THAN
- KEY IS NOT >
- KEY IS NOT LESS THAN
- KEY IS NOT <
- KEY IS NOT EQUAL TO
- KEY IS NOT =

- KEY IS <>
- KEY IS GREATER THAN OR EQUAL TO
- KEY IS >=
- KEY IS LESS THAN OR EQUAL TO
- KEY IS <=

Examples

START example

```
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
```

Read START statement online: https://riptutorial.com/cobol/topic/7464/start-statement

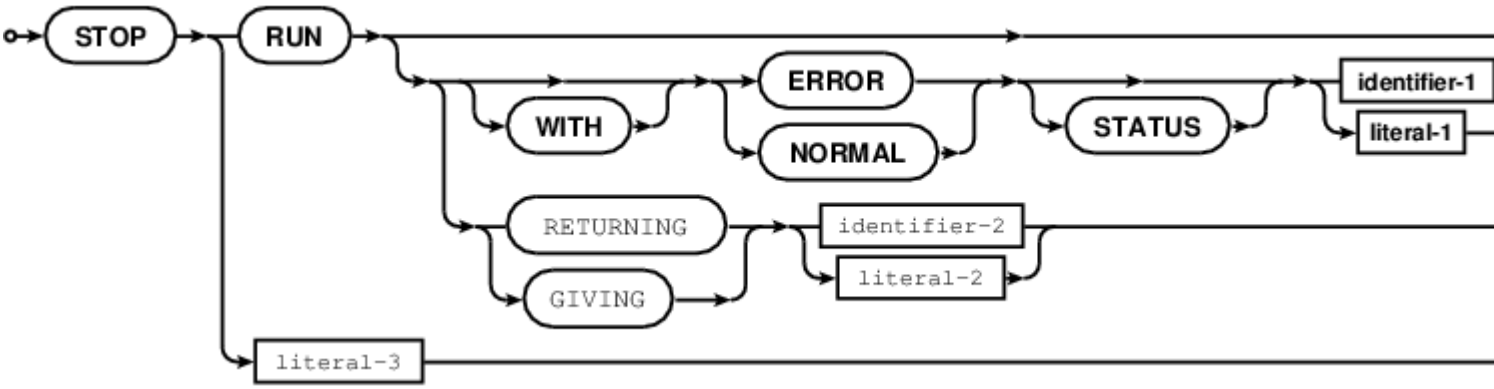
Chapter 43: STOP statement

Remarks

The `STOP` statement terminates the current run unit.

A now deemed obsolete extension to `STOP RUN` is `STOP literal`, which will pause a program until a response from the console is given, at which point execution will resume. This could be handy for things like, "go get the big box of paper and load up the special printer".

`STOP` is a hard program end, `GOBACK` is a slightly nicer way of returning to the operating system or caller module, especially in subroutines that may have no business terminating a run.



Examples

STOP RUN

```
STOP RUN
```

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

Chapter 44: String

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.
```

Not an example, but ....

seemed the only way to add a comment. One thing that's easy to forget is that if you string some variables like the example above, and the resulting length is SHORTER than what was originally in the receiving variable (o- string above) then"trailing"characters are left in place.

For example, if o- string contained "the string contains this data" and you string together "fred & Bert", then o- string will contain "fred & Bertontains this data" (if I counted right).

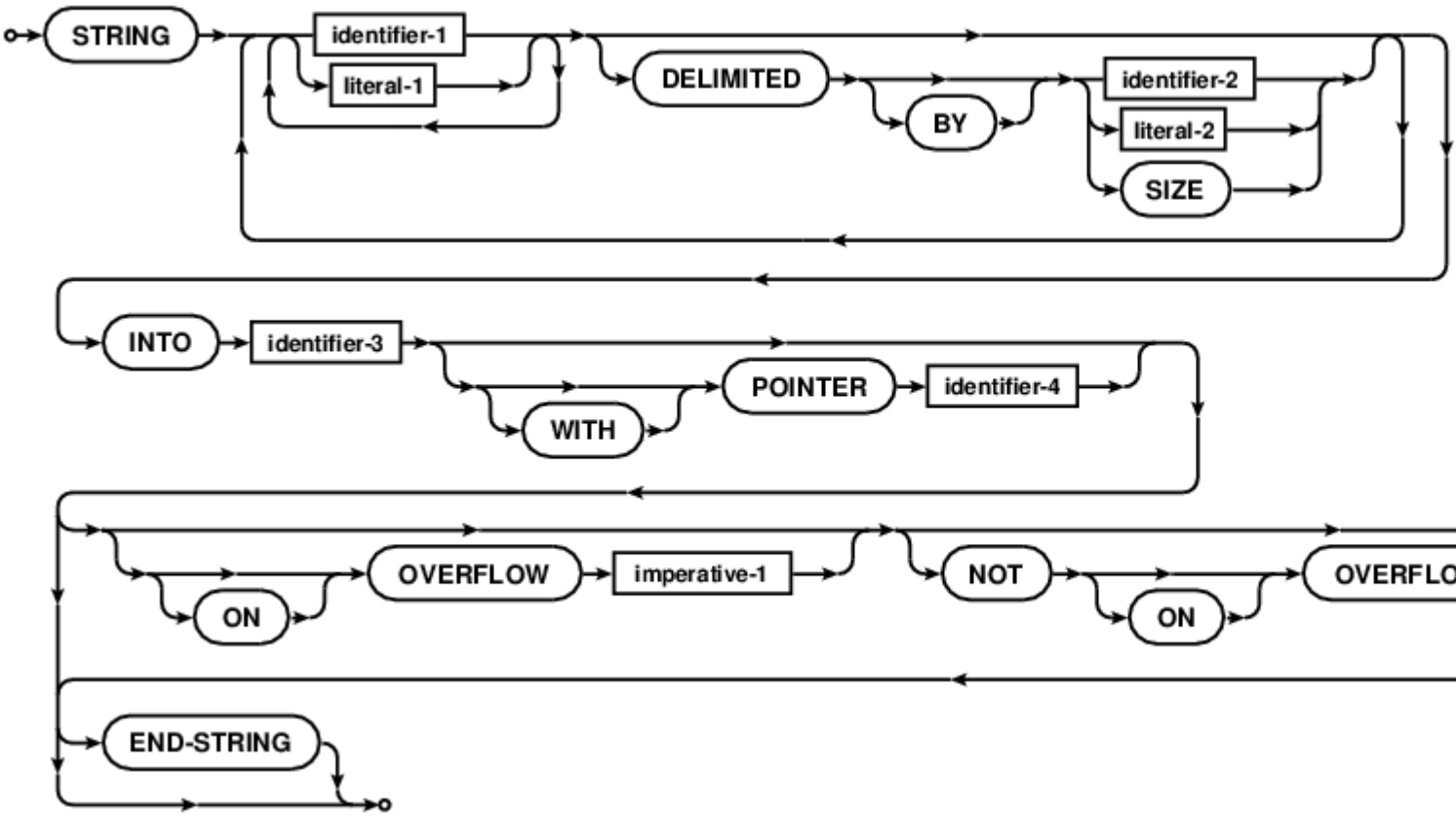
Summa summary, get into the habit of ALWAYS moving spaces into your receiving variable before you start stringing.

Read String online: https://riptutorial.com/cobol/topic/7039/string

Chapter 45: STRING statement

Remarks

The `STRING` statement concatenates the partial or complete contents of multiple fields into a single result.



Examples

STRING example for C strings

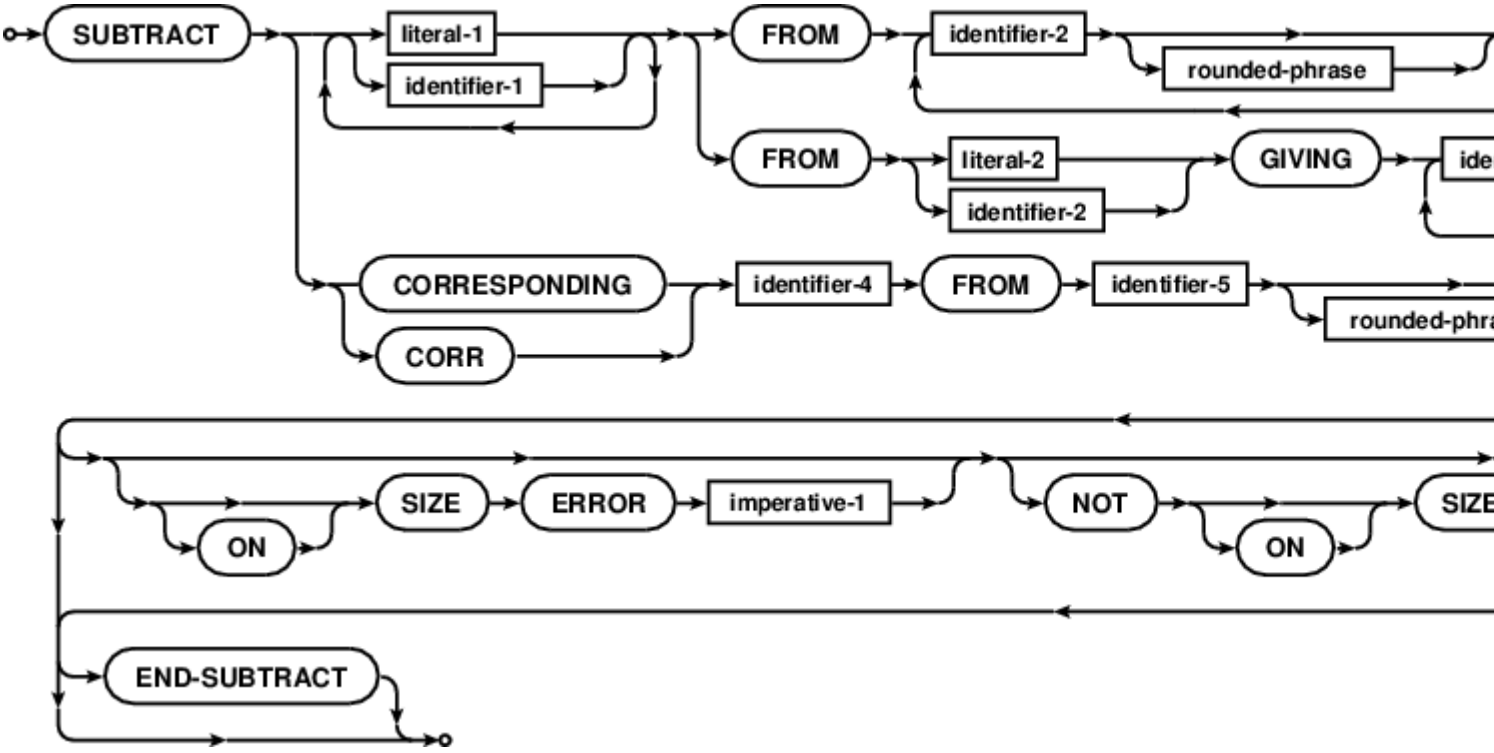
```
*> Strip off trailing zero bytes
STRING c-string DELIMITED BY LOW-VALUE INTO working-store
```

Read STRING statement online: https://riptutorial.com/cobol/topic/7468/string-statement

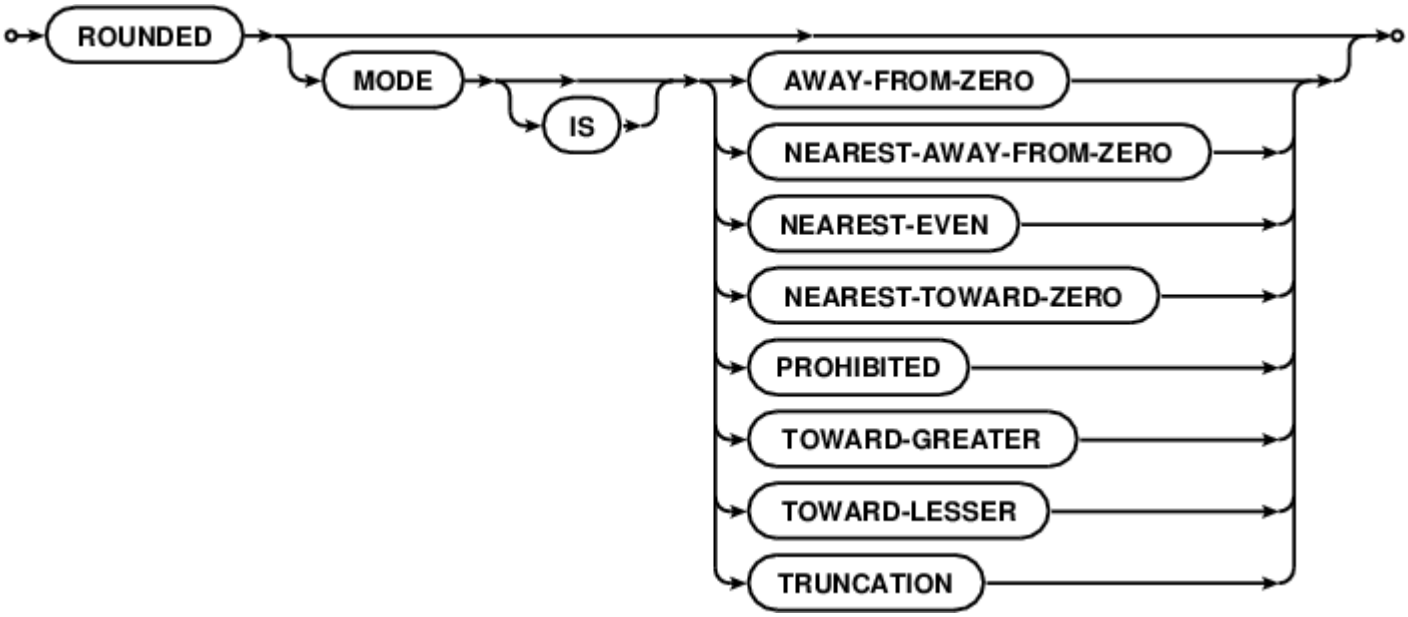
Chapter 46: SUBTRACT statement

Remarks

The `SUBTRACT` statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more identifiers to the result.



rounded-phrase



Examples

SUBTRACT example

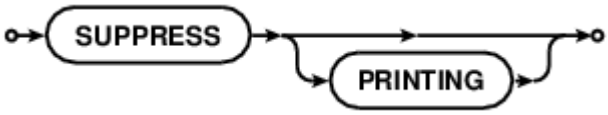
```
SUBTRACT item-a item-b item-c FROM account-z ROUNDED MODE IS NEAREST-EVEN
    ON SIZE ERROR
        DISPLAY "CALL THE BOSS, Account `Z` is OUT OF MONEY" END-DISPLAY
        PERFORM promisary-processing
    NOT ON SIZE ERROR
        PERFORM normal-processing
END-SUBTRACT
```

Read SUBTRACT statement online: https://riptutorial.com/cobol/topic/7465/subtract-statement

Chapter 47: SUPPRESS statement

Remarks

The `SUPPRESS` statement inhibits the printing of a report group. COBOL Report Writer feature.



Examples

SUPPRESS example

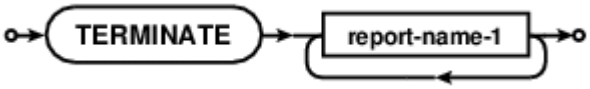
```
SUPPRESS PRINTING
```

Read SUPPRESS statement online: https://riptutorial.com/cobol/topic/7470/suppress-statement

Chapter 48: TERMINATE statement

Remarks

The `TERMINATE` statement is a COBOL Report Writer feature. Terminates the processing on the given report names.



Examples

TERMINATE example

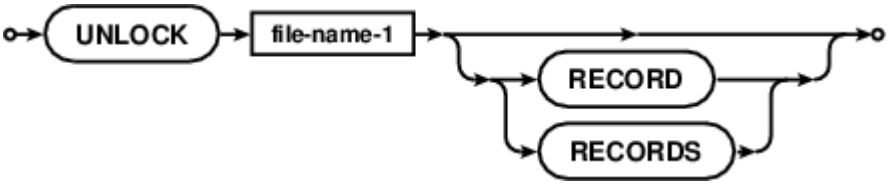
```
TERMINATE report-1 report-2 report-summary
```

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

Chapter 49: UNLOCK statement

Remarks

The `UNLOCK` statement explicitly releases any record locks associated with a file connector.



Examples

UNLOCK record from a file connector

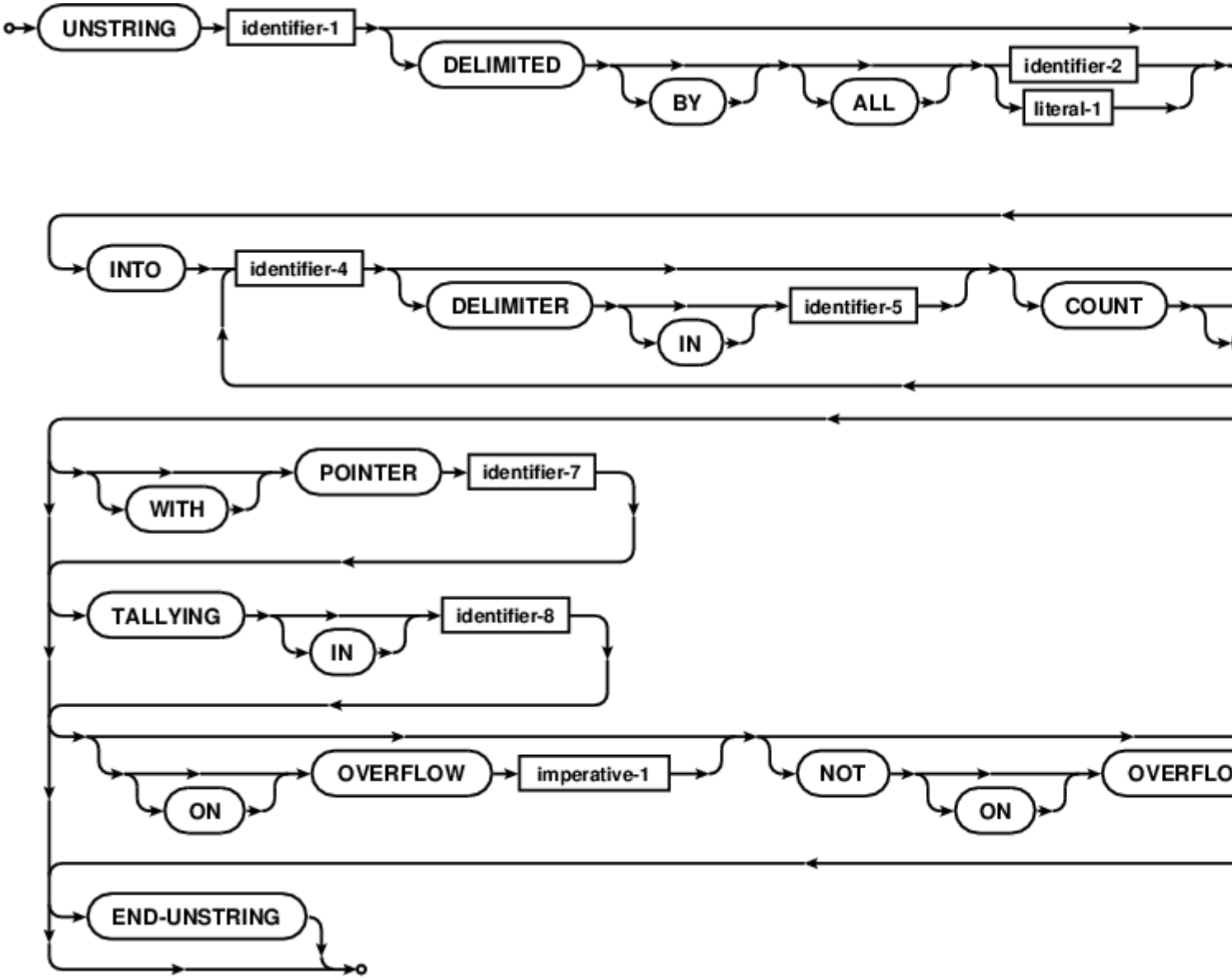
```
UNLOCK filename-1 RECORDS
```

Read UNLOCK statement online: https://riptutorial.com/cobol/topic/7471/unlock-statement

Chapter 50: UNSTRING statement

Remarks

The `UNSTRING` statement separates a sending field and places results in multiple receiving fields.



Examples

UNSTRING example

```
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
```

Read UNSTRING statement online: https://riptutorial.com/cobol/topic/7581/unstring-statement

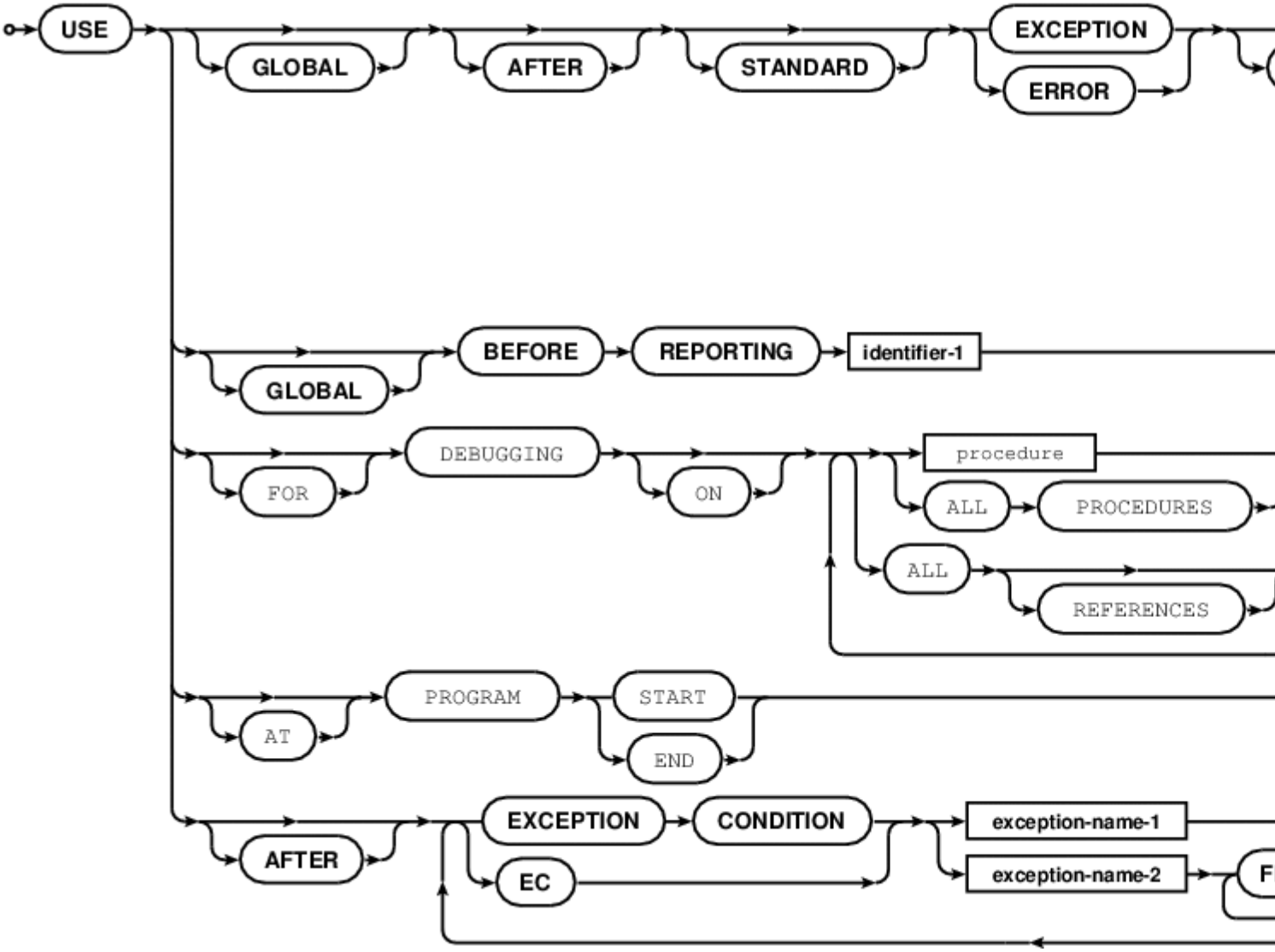
Chapter 51: USE statement

Remarks

The `USE` statement specifies procedures to be used

- for error and exception handling in addition to those provided by other facilities
- before printing of a designated report group
- after detection of designated exception conditions

Obsolete usage includes specifying procedures to be used during `DEBUGGING`, and extensions include adding interstitial procedures for program start and end.



Examples

USE statement with 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.
```

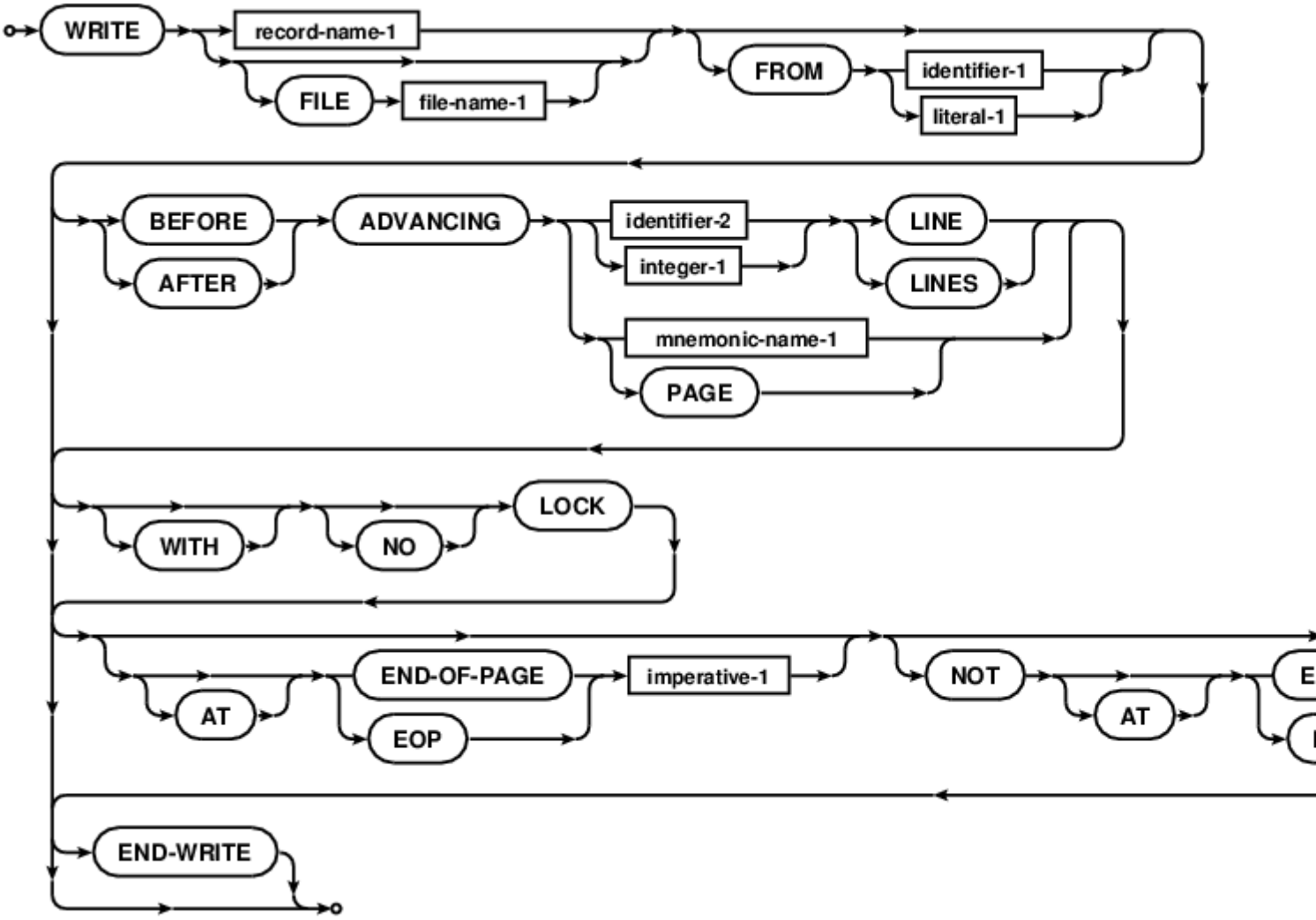
Read USE statement online: https://riptutorial.com/cobol/topic/7582/use-statement

Chapter 52: WRITE statement

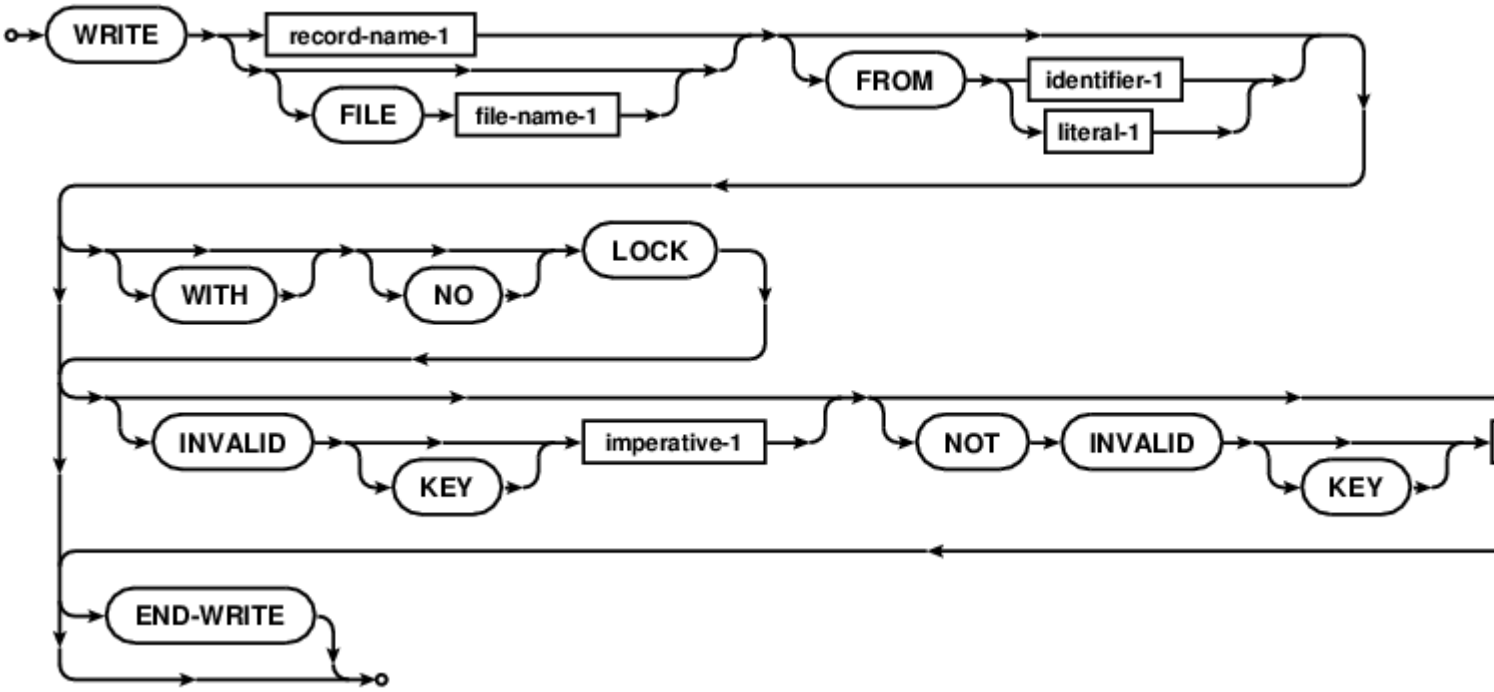
Remarks

The `WRITE` statement releases logical records to an `output` or `input-output` storage resource, and for logical positioning of lines within a logical page.

WRITE sequential



WRITE random



Examples

WRITE examples

```
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
```

Read WRITE statement online: https://riptutorial.com/cobol/topic/7583/write-statement

Credits

S. No	Chapters	Contributors
1	Getting started with cobol	4444, Abhishek Jain, Bharat Anand, Brian Tiffin, Community, Joe Zitzelberger, ncmathsadist
2	ACCEPT statement	Brian Tiffin
3	ADD statement	Brian Tiffin
4	ALLOCATE statement	Brian Tiffin
5	ALTER statement	Brian Tiffin
6	CALL statement	4444, Bill Woodger, Brian Tiffin, infoRene, Jeffrey Ranney, Joe Zitzelberger, Simon Sobisch
7	CANCEL statement	Brian Tiffin
8	COMMIT statement	Brian Tiffin
9	COMPUTE statement	Brian Tiffin
10	CONTINUE statement	Brian Tiffin
11	COPY directive	Brian Tiffin
12	Data division	Bulut Colak
13	DELETE statement	Brian Tiffin
14	DISPLAY statement	Brian Tiffin
15	DIVIDE statement	Brian Tiffin
16	EVALUATE statement	Brian Tiffin
17	EXIT statement	Brian Tiffin
18	FREE statement	Brian Tiffin
19	GENERATE statement	Brian Tiffin

20	GnuCOBOL installation with GNU/Linux	Brian Tiffin
21	GO TO statement	Brian Tiffin
22	GOBACK statement	Brian Tiffin
23	How does the computational work in cobol?	Bruce Martin, Bulut Colak
24	IF statement	Brian Tiffin
25	INITIALIZE statement	Brian Tiffin
26	INITIATE statement	Brian Tiffin
27	INSPECT statement	Brian Tiffin
28	Intrinsic Functions	Brian Tiffin, MC Emperor
29	MERGE statement	Brian Tiffin
30	MOVE statement	Brian Tiffin
31	MULTIPLY statement	Brian Tiffin
32	OPEN statement	Brian Tiffin
33	PERFORM statement	Brian Tiffin
34	READ statement	Brian Tiffin
35	RELEASE statement	Brian Tiffin
36	REPLACE directive	Brian Tiffin
37	RETURN statement	Brian Tiffin
38	REWRITE statement	Brian Tiffin
39	SEARCH statement	Brian Tiffin
40	SET statement	Brian Tiffin
41	SORT statement	Brian Tiffin

42	START statement	Brian Tiffin
43	STOP statement	Brian Tiffin
44	String	Jeffrey Ranney, Michael Simpson
45	STRING statement	Brian Tiffin
46	SUBTRACT statement	Brian Tiffin
47	SUPPRESS statement	Brian Tiffin
48	TERMINATE statement	Brian Tiffin
49	UNLOCK statement	Brian Tiffin
50	UNSTRING statement	Brian Tiffin
51	USE statement	Brian Tiffin
52	WRITE statement	Brian Tiffin