Cobol Tutorial

Basic Procedure Division Commands

Introduction

Aims

The PROCEDURE DIVISION contains the code used to manipulate the data described in the DATA DIVISION. This tutorial examines some of the basic COBOL commands used in the PROCEDURE DIVISION.

In the course of this tutorial you will examine how assignment is done in COBOL, how the date and time can be obtained from the computer and how arithmetic statements are written.

Objectives

By the end of this unit you should -

1. Know how to get data from the keyboard and write it to the screen.
2. Know how to get the system date and time.
3. Understand how the MOVE is used for assignment in COBOL.
4. Understand how alphanumeric and numeric moves work.
5. Be able to use the arithmetic verbs to perform calculations.

Prerequisites

Introduction to COBOL

Declaring data in COBOL

To top of page

Basic User Input and Output

Introduction

In COBOL, the ACCEPT and DISPLAY verbs are used to read from the keyboard and write to the screen. Input and output using these commands is somewhat primitive

because they were originally designed to be used in a batch programming environment to communicate with the computer operator.

In recent years many vendors have augmented the ACCEPT and DISPLAY syntax to facilitate the creation of on-line systems by allowing such things as: cursor positioning, character attribute control and auto-validation of input.

In this tutorial we will examine only the standard ACCEPT and DISPLAY syntax.

The DISPLAY verb

In the COBOL syntax diagrams (the COBOL metalanguage) upper case words are keywords. If underlined, they are mandatory.
{ } brackets mean that one of the options must be selected
[ ] brackets mean that the item is optional
ellipses (...) mean that the item may be repeated at the programmers discretion.

The symbols used in the syntax diagram identifiers have the following significance:-
$ signifies a string item,
is numeric item,
i indicates that the item can be a variable identifier
l indicates that the item can be a literal.

```
DISPLAY OutputItem1$#il [OutputItem2$#il] ...
        [UPON Mnemonic-Name] [WITH NO ADVANCING]
```

The DISPLAY verb is used to send output to the computer screen or to a peripheral device.

As you can see from the ellipses (...) in the metalanguage above a single DISPLAY can be used to display several data-items or literals or any combination of these.

DISPLAY notes
After the items in the display list have been sent to the screen, the DISPLAY automatically moves the screen cursor to the next line unless a WITH NO ADVANCING clause is present.

Mnemonic-Names are used to make programs more readable. A Mnemonic-Name is a name devised by the programmer to represent some peripheral device (such as a serial port) or control code. The name is connected to the actual device or code by entries in the ENVIRONMENT DIVISION.

When a Mnemonic-Name is used with the DISPLAY it represents an output device (serial port, parallel port etc).

If a Mnemonic-Name is used output is sent to the device specified; otherwise, output is sent to the computer screen.

DISPLAY examples

```
DISPLAY "My name is " ProgrammerName.
DISPLAY "The vat rate is " VatRate.
DISPLAY PrinterSetupCodes UPON PrinterPort1.
```

The ACCEPT verb

```
Format1. ACCEPT Identifier [FROM Mnemonic-Name]

Format2. ACCEPT Identifier FROM { DATE
                                { DAY
                                { DAY-OF-WEEK
                                { TIME
```

The ACCEPT verb is used to get data from the keyboard, a peripheral device, or certain system variables.

ACCEPT notes
When the first format is used, the ACCEPT inserts the data typed at the keyboard (or coming from the peripheral device), into the receiving data-item.

When the second format is used, the ACCEPT inserts the data obtained from one of the system variables, into the receiving data-item.

Using the ACCEPT to get the system date and time

The second format of the ACCEPT allows the programmer to access the system date and time (i.e. the date and time held in the computer's internal clock). The system variables provided are -

- Date
- Day of the year
- Day of the week
- Time

The declarations and comments below show the format required for data-items receiving each of the system variables.

```
01 CurrentDate         PIC 9(6).
* CurrentDate is the date in YYMMDD format


01 DayOfYear           PIC 9(5).
* DayOfYear is current day in YYDDD format


01 Day0fWeek           PIC 9.
* DAY-OF-WEEK is a single digit where 1=Monday


01 CurrentTime         PIC 9(8).
* CurrentTime is the time in HHMMSSss format where s = S/100
```

New formats for the ACCEPT

The problem with ACCEPT ..FROM DATE and ACCEPT..FROM DAY is that since they hold only the year in only two digits, they are subject to the millennium bug. To resolve this problem, these two formats of now take additional (optional) formatting instructions to allow the programmer to specify that the date is to be supplied with a 4 digit year.

The syntax for these new formatting instructions is:

ACCEPT DATE [YYYYMMDD]
ACCEPT DAY [YYYYDDD]

When the new formatting instructions are used, the receiving fields must be defined as;

```
01 Y2KDate PIC 9(8).
* Y2KDate is the date in YYYYMMDD format

01 Y2KDayOfYear PIC 9(7).
* Y2KDayOfYear is current day in YYYYDDD format
```

ACCEPT and DISPLAY example program

This example program uses the ACCEPT and DISPLAY to get a student record from the user and display some of its fields. It also demonstrates how the ACCEPT can be used to get the system date and time.


```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
```

```
* 4 digit year.  If not used, the year supplied by the system
* will only contain two digits which may cause a problem
* in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
   DISPLAY "Enter student details using template below".
   DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
   DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
   ACCEPT  StudentDetails.
   ACCEPT  CurrentDate FROM DATE YYYYMMDD.
   ACCEPT  DayOfYear FROM DAY YYYYDDD.
   ACCEPT  CurrentTime FROM TIME.
   DISPLAY "Name is ", Initials SPACE Surname.
   DISPLAY "Date is " CurrentDay SPACE CurrentMonth
           SPACE CurrentYear.
   DISPLAY "Today is day " YearDay " of the year".
   DISPLAY "The time is " CurrentHour ":" CurrentMinute.
   STOP RUN.
```

Results of running ACCEPT.CBL

```
Enter student details using template below
Enter - ID,Surname,Initials,CourseCode,Gender
SSSSSSSNNNNNNNNIICCCCG
9923453Power   NSLM51F
Name is NS Power
Date is 01 03 1999
Today is day 060 of the year
The time is 14:41
```

To top of page

Assignment using the MOVE verb

Introduction

In strongly typed languages like Modula-2, Pascal or ADA the assignment operation is simple because assignment is only allowed between data items with compatible types. The simplicity of assignment in these languages, is achieved at the cost of having a large number of data types.

In COBOL there are basically only three data types;

- Alphabetic (PIC A)
- Alphanumeric (PIC X)
- Numeric (PIC 9)

But this simplicity is achieved only at the cost of having a very complex assignment statement.

In COBOL, assignment is achieved using the MOVE verb.

The MOVE verb

MOVE Source$#il TO Destination$#i ...

As we can see from the syntax metalanguage above, the MOVE copies data from the source identifier or literal to one or more destination identifiers.

Although this sounds simple, the actual operation of the MOVE is somewhat more complicated and is governed by a number of rules.

MOVE rules
In most other programming languages, data is assigned from the source item on the right to the destination item on the left (e.g. Qty = 10;) but in COBOL the MOVE assigns data from left to right. The source item is on the left of the word TO and the receiving item(s) is on the right.

The source and destination identifiers can be group or elementary data-items.

When data is moved into an item, the contents of the item are completely replaced.

If the number of characters in the source data-item is less than the number in the destination item, the rest of the destination item is filled with zeros or spaces.

If the source data-item is larger than the destination item, the characters that cannot fit into the destination item will be lost. This is known as truncation.

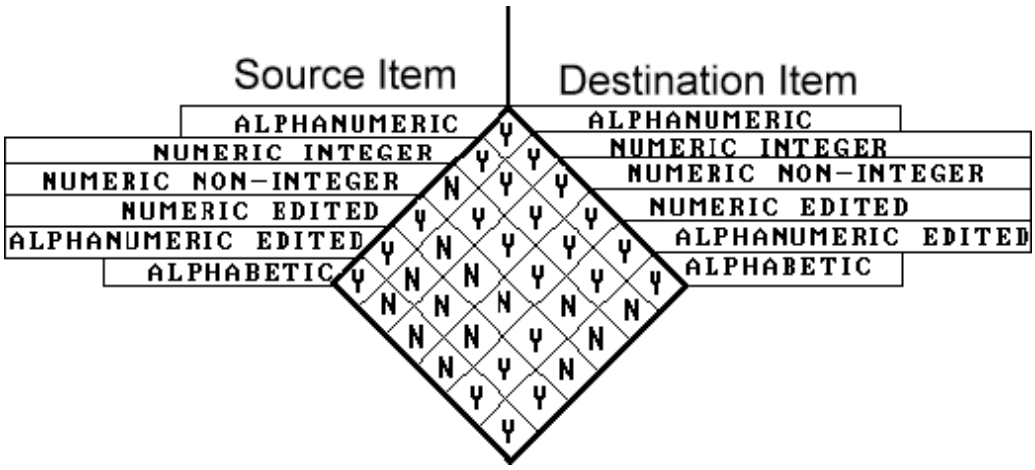
When the destination item is alphanumeric or alphabetic (PIC X or A), data is copied into the destination area from left to right with space filling or truncation on the right.

When the destination item is numeric, or edited numeric, data is aligned along the decimal point with zero filling or truncation as necessary.

When the decimal point is not explicitly specified in either the source or destination items, the item is treated as if it had an assumed decimal point immediately after its rightmost character.

MOVE combinations
Although COBOL is much less restrictive in this respect than many other

languages, certain combinations of sending and receiving data types are not permitted and will be rejected by the compiler. The valid and invalid MOVE combinations are shown in the diagram below:



MOVE examples

Examine the examples in the animation below in combination with the MOVE rules above. Make sure you understand the why the moves produce the results shown.

To top of page

Arithmetic in COBOL

Introduction

Most procedural programming languages perform computations by assigning the result of an arithmetic expression or a function to a variable. In COBOL the COMPUTE verb is used to evaluate arithmetic expressions, but there are also specific commands for adding, subtracting, multiplying and dividing.

Data Movement

In a MOVE operation data is moved from a source item on the left to the destination item(s) on the right. Data movement is from left to right. The same direction of data movement can be observed in the COBOL arithmetic verbs.

All the arithmetic verbs, except the COMPUTE, assign the result of the calculation to the rightmost data-items.

General Rules

All the arithmetic verbs move the result of a calculation into a receiving data-item according to the rules for a numeric move; that is, with alignment along the assumed decimal point and with zero-filling or truncation as necessary.

All arithmetic verbs must use numeric literals or numeric data-items (PIC 9) that contain numeric data. There is one exception. Identifiers that appear to the right of the word GIVING may refer to numeric data-items that contain editing symbols.

When the GIVING phrase is used, the data-item following the word GIVING is the receiving field of the calculation but it is **not** one of the statement operands (does not contribute to the result). The original values of all the items before the word GIVING are left intact.

If the GIVING phrase is not used, the data-item(s) after the word TO, FROM, BY or INTO both contribute to the result and are receiving field for it.

The maximum size of each operand is 18 digits.

The ROUNDED option

All the arithmetic verbs allow the ROUNDED phrase.

The ROUNDED phrase takes effect when, after decimal point alignment, the result calculated must be truncated on the right hand side. The option adds 1 to the receiving item when the leftmost truncated digit has an absolute value of 5 or greater.

ROUNDED examples			
Receiving Field	**Actual Result**	**Truncated Result**	**Rounded Result**
PIC 9(3)V9	**123.25**	**123.2**	**123.3**
PIC 9(3)V9	**123.247**	**123.2**	**123.2**
PIC 9(3)	**123.25**	**123**	**123**

ON SIZE ERROR

When a computation is performed it is possible for the result to be too large or too small to be contained in the receiving field. When this occurs, there will be truncation of the result. The ON SIZE ERROR phrase detects this condition.

ON SIZE ERROR notes
All the arithmetic verbs allow the ON SIZE ERROR phrase.

A size error condition exists when, after decimal point alignment, the result is truncated on either the left or the right hand side.

If an arithmetic statement has a ROUNDED phrase then a size error only occurs if there is truncation on the left-hand side (most significant digits) because if we specify the ROUNDED option we indicate that we know there will be truncation on the right and are specifying rounding to deal with it.

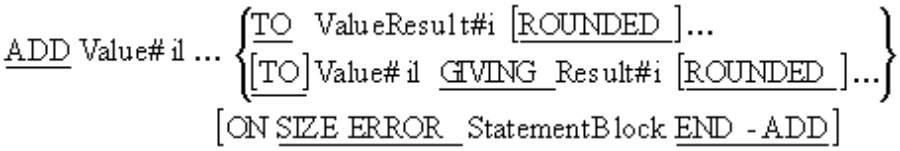
Division by 0 always causes a SIZE ERROR.

ON SIZE ERROR examples

Receiving Field	Actual Result	Truncated Result	Size Error?
PIC 9(3)V9	**245.96**	**245.9**	**YES**
PIC 9(3)V9	**3245.9**	**245.9**	**YES**

PIC 9(3)	**324**	**324**	**NO**
PIC 9(3)	**5324**	**324**	**YES**
PIC 9(3)V9 not Rounded	**523.35**	**523.3**	**YES**
PIC 9(3)V9 Rounded	**523.35**	**523.4**	**NO**
PIC 9(3)V9 Rounded	**3523.35**	**523.4**	**YES**

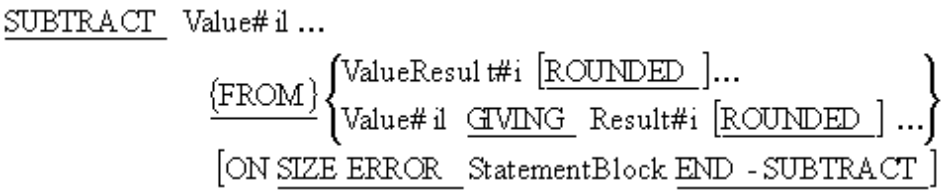
ADD verb



If the GIVING phrase is used, everything before the word GIVING is added together and the **combined result** is moved into each of the Result#i items.

If the GIVING phrase is not used, everything before the word TO is added together and the **combined result** is then added to **each** of the ValueResult#i items in turn.

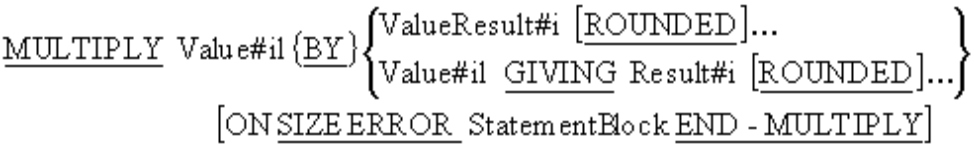
SUBTRACT verb



If the GIVING phrase is used, everything before the word FROM is added together and the **combined result** is subtracted from the Value#il item after the word FROM and the result is moved into each of the Result#i items.

If the GIVING phrase is not used everything before the word FROM is added together and the **combined result** is then subtracted from **each** of the ValueResult#i items after the word FROM in turn.

MULTIPLY verb



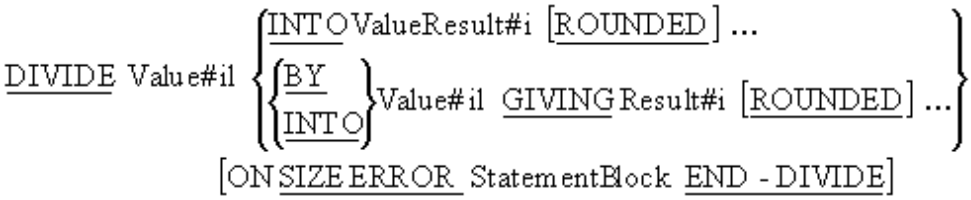
If the GIVING phrase is used, then the item to the left of the word BY is multiplied by the Value#i item to the right of the word BY and the result is moved into **each** of the Result#i items.

If the GIVING phrase is not used, then the Value#il to the left of the word BY is multiplied by **each** of the ValueResult#i items. The result of each calculation is placed in the ValueResult#i involved in the calculation.

DIVIDE verb

The Divide has two main formats. One produces a remainder and the other does not.

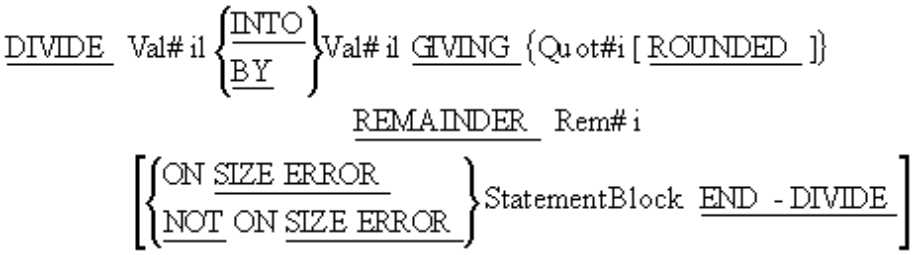
Format1



In the GIVING phrase is used, the Value#il to the left of BY or INTO is divided by or into the Value#il to the right of BY or INTO and the result of the calculation in moved into **each** of the Result#i items in turn.

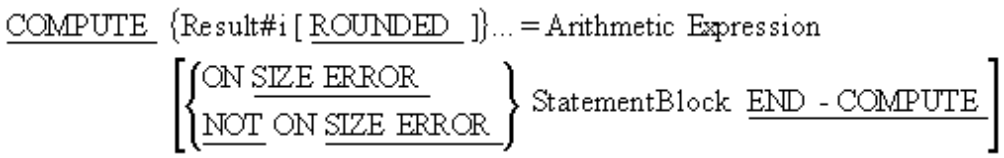
If the GIVING phrase is not used, the item to the left of the word INTO is divided into each of the ValueResult#i items in turn. The result of each calculation is placed in the ValueResult#i involved in the calculation.

Format2



In this format the Val#il to the left of BY or INTO is divided by or into the Val#il to the right of BY or INTO. The quotient part of the computation is assigned to Quot#i and the remainder is assigned to Rem#i.

COMPUTE verb



The COMPUTE assigns the result of an arithmetic expression to a data-item. The arithmetic expression is evaluated according to the normal arithmetic rules. That is, the expression is normally evaluated from left to right but bracketing and the precedence rules shown below can change the order of evaluation.

Precedence	Symbol	Meaning
1.	**	Power
2.	*	multiply
	/	divide
3.	+	add
	-	subtract

Note that unlike some other programming languages COBOL provides the ** expression symbol to represent raising to a power.

Arithmetic examples

The animation below contains examples of each of the arithmetic verbs. The arithmetic statement shows the contents of the variables before the statement executes. Initially the contents of the variables after execution are hidden; but you can display them by clicking with the mouse.

Before you display the contents of the variables, try to figure out what they are going to be. If you get the wrong answer, make sure you understand why the statement produces the answer shown.

To top of page

Copyright Notice

Last updated : March 1999

e-mail : CSISwebeditor@ul.ie