COBOL

DIVISIONS IN COBOL:

1) **IDENTIFICATION DIVISION** – Identifies just the Program Name (any valid meaningful name), Author (your name or your company name or any name), Date written (can be todays date) and the Date the latest version is compiled. It's the mandatory division as COBOL compiler will expect this division to be present. All the fields can start in Area A (Cols 8-11).

```
IDENTIFICATION DIVISION.      -> MANDATORY
PROGRAM-ID. FIRSTPGM.         -> MANDATORY
AUTHOR. VENKAT.               -> OPTIONAL
```

2) **ENVIRONMENT DIVISION –** It's an Optional division. But usually it will be present as 99% of the programs uses files and the file definition will be given in this division. All Division and Section Headers can be in Area A.

It has 2 Sections:- (a) Configuration Section.
(b) INPUT-OUTPUT Section.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   < computer name> -> NOT REQUIRED
OBJECT-COMPUTER.   < computer name> -> NOT REQUIRED
SPECIAL-NAMES.    <special-names-entry> -> NOT USED REGULARLY

INPUT-OUTPUT SECTION.
FILE-CONTROL. <SELECT Statements defn links with DDNAMES in JCL>
I-O-CONTROL.  < NOT USED REGULARLY>
```

(a) Configuration Section will identify the Computer name in which the program was written and the Computer name in which it is compiled. These are not coded in new programs nowadays. So, better to avoid.

(b) INPUT-OUTPUT Section will have **Select Statement** entries under FILE-CONTROL header. SELECT Statements are used to link the DDNAMES of Datasets in Run JCL with that of filenames used in the program.

3) **DATA DIVISION –** Consists of all DATA Entries. It's an Optional Division for COBOL compiler but usually 99.9% of programs requires DATA DIVISION. It has 3 Sections as given below:-

a. **FILE SECTION:** Describes the structure (Record Length, Record format) of the PS files or VSAM datasets whether it is used as an Input to this program or Output out of this program.

b. **WORKING STORAGE SECTION:** Declaration of Temporary Data names, be it an Alphabetic (PICTURE Clause A), Alphanumeric

(PICTURE Clause X), NUMERIC (PICTURE Clause 9). There is no known restriction for defining Alphabetic and Alphanumeric. In Numeric, we could not go beyond 9(18).

Usually Input file record is READ and MOVED to working storage, Processing is done and MOVE to OUTPUT Record and WRITTEN to Output file.

C. **LINKAGE SECTION:** This Section will have field variables declared

a) when you need to send DATA to another program from the current program using CALL PGM-NAME in current Program.
b) when you need to receive a parameter from Jcl thru PARM keyword, written as given below.

```
//step01  exec  pgm=reportwr,parm='2015'
```

c) When you wanted to get data from SYSIN of a Run JCL
d) When you wanted to get IMS-DLI execution information in Batch
e) When you wanted to get data from a CICS program in Online CICS region

4) **PROCEDURE DIVISION:** This is a mandatory division as all the logic will be written here.

HOW MVS OS Executes a Program Load Module:-
MVS OS will execute line by line of the procedure division and start from the first line until it meets the statement (STOP RUN or GOBACK) to stop the execution. When it meets the sequence control statements like PERFORM (Perform PARAs or SECTIONS and come back), GO TO (GO to a Para or an exit para in the same PARA or SECTION), EXIT (Just Exits and it is used usually in EXIT-PARA, when used here it doesn't make any difference in the sequence of execution),

COBOL CODING RULE:

1) IDENTIFICATION (PROGRAM-ID) and PROCEDURE Divisions are Mandatory.
2) You are supposed to put the command **'NUM ON STD COBOL'** in Command line in your PDS member in EDIT mode, so that the member repositions the column such that 1-6 Line numbers goes to the Left hand side.
3) Column 7 will accept only the following values
   a. *** - Comment Line is starting, Compiler will not execute this line. Comment line can be given anywhere and there is no restriction in number of comment lines**
   b. **Space – If a SPACE is found, Compiler will assume a valid COBOL statement is on that line and tries to compile. One Sentence denotes a valid COBOL command verb with literals or fields or constants or any operators until it find a SCOPE TERMINATOR '.'**

c. '-' – Continuation line, Usually it is avoided.
d. '/' – Comment line, When Program is printed to a mainframe printer. It instructs the printer to denote a new page.
e. 'D' – Debugging. Usually DISPLAY statements are used only for testing.In order to avoid these statements to print in spool in Production environment, we add 'D' in Column 7 for those lines. So, 'D' acts as a '*' (comment) in Production (Different COBOL compiler) while it acts as ' ' in Test Environment.

4) AREA A (8-11)
   a. All Division, Section and PARA Headers, FD and SD entries of FILE Section, Level no's 01 and 77
   b. Must be blank for a continuation line
5) AREA B (12-72)
   All other Coding will be done here.

WORKING STORAGE SECTION LEVEL NUMBERS

Every program will be allocated a Temporary memory to store all Data which comes as an input to the program or goes out of the program and also any temporary variable required for any logical processing during the execution of the program. Working Storage variables will be stored in this temporary memory which lives only during the execution of the program.

Working Storage variables are required for the following:-
a) Bringing in a copy of record/Data from FILE/Database (or) to a Temporary Memory Storage for processing inside the Program
b) For any Logical processing inside the Program
c) For Preparing O/p record to be written to an O/p FILE or to a Database

Level Numbers denote the hierarchical structure of a Working Storage variable.

Level numbers vary from 01 till 49, 66, 77, 88

01 will either be a standalone/elementary level number with one variable defined (or) a group variable with as many subgroup variables defined under it. It is denoted in Area A.

```
01  WS-INPUT-RECORD            PIC X(200).

01  WS-EMPL-REC.
    05   WS-EMPL-NO            PIC X(06).
    05   WS-EMPL-NAME          PIC X(20).
    05   WS-EMPL-ADDRESS       PIC X(20).
    05   WS-EMPL-PHONE         PIC X(10).

01 WS-BIRTH-DATE.
```

```
       05 WS-BIRTH-DATE-DD       PIC  9(02).
       05 WS-BIRTH-DATE-MM       PIC  9(02).
       05 WS-BIRTH-DATE-YY       PIC  9(02).
```

77 will only be an elementary variable. It is defined in area A. It cannot have any subgroups. It is normally used to declare any constants. It is similar to 01 that is used for defining constants.

```
  77  WS-COUNTER                PIC 9(05).
  77  WS-SUB                    PIC 9(01).
```

66 Renames (Will be discussed later)

88 Conditional Names.

- It is given under a group variable. It has only values and not a PIC clause.
- It is widely used inside the program for better program readability.
- Usually conditional expression like IF or EVALUATE usually uses this.
- VALUE or VALUES clause are used to define 1 or more values.

```
  01  WS-GENDER                  PIC  X(1).
      88 WS-VALID-GENDER   VALUE 'M' 'F'.
      88 MALE              VALUE 'M'.
      88 FEMALE            VALUE 'F'.

  01  STUDENT-MARKS              PIC 9(03).
      88 PASS              VALUE 040 THRU 100.
      88 FAIL              VALUE 000 THRU 039.
```

In procedure Division, it is used like this,

```
     MOVE WS-STUDENT-MARKS TO STUDENT-MARKS.

     IF PASS
          DISPLAY ' STUDENT HAVE PASSED'
     ELSE
          DISPLAY ' STUDENT HAVE FAILED'
     END-IF.
```

```
02  WS-CLASS          PIC 99.
       88  WS-PRIMARY      VALUE IS 1 THRU 5.
       88  WS-SECONDARY    VALUE IS 6 THRU 8.
       88  WS-HIGH         VALUE IS  9  THRU 10.
       88  WS-HIGHER-SEC   VALUE IS 11 THRU 12.

     IF    WS-PRIMARY
           PERFORM  PRM-ROUTINE
     ELSE   IF   WS-SECONDARY
           PERFORM  SEC-ROUTINE
     ELSE   IF   WS-HIGH
```

```
           PERFORM  HIGH-ROUTINE
      ELSE   IF   WS-HIGHER-SEC
           PERFORM  HIGHER-ROUTINE.
```

PICTURE CLAUSE

Picture clause denotes the Format and Length of the variable to declared using level numbers.

9 – Numeric (Storage of 1 Byte) S – Signed (Sign for a Numeric) (No Extra Storage) V – Implied Decimal X – Alpha Numeric A – Alphabetic (Used very rarely, Usually X will be used for Alphabetic too)

Examples:-
PIC 9(2)V99S, PIC S9(5)V99, PIC S9(9), X(20000), A(5)

VALUE CLAUSE

Value clause is used to initialize a variable (say a Numeric or Alphanumeric) with SPACES/ZEROES or any valid data. It follows a PIC clause.

```
01 WS-COUNTER    PIC 9(04) VALUE ZEROES.
01 WS-RECORD     PIC X(50) VALUE SPACES.

01 WS-SUB        PIC 9(01) VALUE 1.

01 WS-GENDER                    PIC  X(1).
   88 MALE                VALUE 'M'.
   88 FEMALE              VALUE 'F'.
```

FILLER CLAUSE

FILLER Clause is used as a dummy variable in COBOL. It can have a VALUE clause. It is used to indicate spaces between variables in a Group or at the end of the record.

```
01    EMPLOYEE-REC-IN.
        05 EMPLOYEE-ID-IN   PIC 9(7).
        05 FILLER           PIC X(2)
        05 EMPLOYEE-NAME-IN PIC X(25).
        05 FILLER           PIC X(2).
        05 HOURS-WORKED-IN  PIC 9(5).
        05 FILLER           PIC X(45).

01    CURR-DATE.
      02  DD     PIC  9(2).
      02  FILLER PIC  X(1)  VALUE  '/'.
      02  MM     PIC  9(2).
      02  FILLER PIC  X(1)  VALUE '/' .
```

```
      02  YY     PIC  9(2).
```

PROCEDURE DIVISION COMMAND VERBS

DISPLAY

Displays a Literal (Numeric or Alphanueric) or a Data in a Working Storage Field (Variable) in SYSOUT.

- Used to display successful START and END of the Programs, Section, PARAs
- **Used for Debugging purposes to display values in a variable or any other field that aid to find the coding error in Logic**
- To display Error Message in Table Overflow, Arithmetic Expression size error
- Can also be used to create reports thru SYSOUT and sent to Printers

DISPLAY is a statement and it can have upto 132 chars for a single display line in SYSOUT.

```
DISPLAY 'PROGRAM STARTED EXECUTION'.

DISPLAY 'VALUES BEFORE PERFORM LOOP'.
DISPLAY 'WS-EMP-ID (SUB)' WS-EMP-ID (SUB).
DISPLAY ALL ' '.  (DISPLAY ENTIRE LINE WITH SPACES)
DISPLAY ALL '*'.
DISPLAY WS-A  WS-B WS-C WS-A WS-B WS-C
        WS-A1 WS-B2 WS-C3 WS-A4 WS-B2 WS-C1
        WS-D WS-E WS-F WS-G WS-H WS-I.
```

ACCEPT

Used to accept records from a SYSIN of a JCL (or) to accept system DATE (YYMMDD) , TIME (HHMMSSSS), DAY (YYDDD) , DAY-OF-WEEK(D).

```
ACCEPT WS-CUSTOMER-DETAILS. (Get a record from sysin)

01 WS-CURR-DATE PIC 9(6).
01 WS-CURR-TIME PIC 9(8).
01 WS-CURR-DAY  PIC 9(5).
01 WS-CURR-DAY-OF-WEEK PIC 9(1).

ACCEPT WS-CURR-DATE FROM DATE.
ACCEPT WS-CURR-TIME FROM TIME.
ACCEPT WS-CURR-DAY  FROM  DAY. (Accepts a Julian date from system)

ACCEPT WS-CURR-DAY-OF-WEEK  FROM  DAY-OF-WEEK.
IF WS-CURR-DAY-OF-WEEK = 1
      DISPLAY 'MONDAY'.
```

2- Tuesday, 3 – Wednesday, 4 – Thursday, 5 – Friday, 6 – Saturday, 7 – Sunday

MOVE CLAUSE

Move statement assigns a value (numeric/literal) to a variable/field.

```
MOVE ZEROES TO WS-SUBSCRIPT.
MOVE 999     TO WS-MAX-COUNTER.

MOVE 'SUMMARY REPORT' TO WS-REPORT-HEADER.
```

Move statement also can be used to move contents of a field to another field. It can be either numeric or alphanumeric. **We cannot move a Alphanumeric or Alphabetic variable to a Numeric variable. While moving make sure that the receiving field has enough space to hold the value moved to avoid any truncation.**

```
MOVE WS-A   TO WS-B.
```

Following table denotes the allowable moves.

Sending Field	Receiving Field			
	Numeric	Alphabetic	Alphanumeric	Group item
Numeric	√	x	√*	√
Alphabetic	X	√	√	√
Alphanumeric	X	√	√	√
ZEROS	√	x	√	√
SPACES	X	√	√	√
Group item	X	√	√	√

MOVE CORR

To move the contents of some or all those data items, of one group, to the corresponding identically named data items, of another group

```
MOVE {CORRESPONDING } identifier-1  TO  identifier-2.

01  WS-DATE-SYSTEM.
      05    WS-YY       PIC 9(2).
      05    WS-MM       PIC 9(2).
      05    WS-DD       PIC 9(2).

01  WS-DATE-PRINT.
      05    WS-DD       PIC 99.
      05    FILLER      PIC X   VALUE  "/".
      05    WS-MM       PIC 99.
      05    FILLER      PIC X   VALUE  "/".
      05    WS-YY       PIC 99.
```

MOVE CORR WS-DATE-SYSTEM TO WS-DATE-PRINT.
is equivalent to the following three MOVE statements

(OR)

```
MOVE  WS-DD   OF  WS-DATE-SYSTEM  TO  WS-DD   OF  WS-DATE-PRINT.
MOVE  WS-MM   OF  WS-DATE-SYSTEM  TO  WS-MM   OF  WS-DATE-PRINT.
MOVE  WS-YY   OF  WS-DATE-SYSTEM  TO  WS-YY   OF  WS-DATE-PRINT.
```

COMMON CODING ERRORS

1) **Truncation:** While moving a variable to another variable say a Numeric or Alphanumeric, we need to make sure that the receiving field has enough space to hold the input value.
2) During Arithmetic Operation like ADD, Multiply or Compute make sure that the receiving field is declared properly

- Say if you are adding 10 9(3) variables, then receiving variable should be declared as 9(4) to hold the max added value.

HOW TO ANALYZE A REQUEST BEFORE STARTING CODING

- *Any Business Request will be in plain English or in Business terms.*
- *Understand the request first thoroughly. Read multiple times if you want. If you have any questions regarding the request, don't hesitate to ask before proceeding.*
- *Spend enough time to understand the request and start planning how to proceed(only if you are ready without any questions).*
- *Understand all the data given as an Input for the Program. Understand the layout (fields record, starting position and length, type of data (numeric or alpha), valid values for fields) and the amount of data it contains and what data it contains.*
- *Understand the O/p from the Program*
- *See if the data provided to you is sufficient to bring out the Output else ask questions to the person who gave the request*
- *Work out the logic on how to get the Output data with the given Input data.*

LOGIC

- Break down the request into smaller parts and try to understand the request if the request is complex
- Don't worry about any other divisions and work only on Procedure Division as it holds the logic
- Any working storage required can be defined later, but first work on the Procedure division logic
- Add AS MANY Display statements in your program to see how your Program works and it is also helpful if there is a coding issue

A Program usually has 7 Logical parts

- INITIALIZE THE VARIABLES USED IN YOUR PROGRAM
- ACCEPT ANY SYSTEM VARIABLES IF REQUIRED
- OPEN THE INPUT FILES (PS OR VSAM) /
  DECLARE AND OPEN ANY DB2 CURSORS
- READ INPUT FILES/DATABASES RECORD ONE BY ONE
- **PROCESS THE RECORD (ALL LOGICAL PROCESSING)**
- WRITE THE OUTPUT RECORDS FOR PS FILES
  REWRITE (OR) INSERT (OR) DELETE RECORDS FOR VSAM FILES
  INSERT (OR) REWRITE (OR) DELETE DB2/Any DB Records

- CLOSE/COMMIT THE FILES

- Classify everything into PARAs
- MAIN PARA THAT CALLS MAIN PROCESSING PARA AND FINAL STATEMENT AS GOBACK
- INITIALIZE, ACCEPT, OPEN IN ONE PARA
- READ TO BE DONE IN SEPARATE PARAS
- MAIN PROCESSING IN ONE PARA
  CALL INDIVIDUAL LOGIC PARAS FROM MAIN PROCESSING PARA
- WRITE TO BE DONE IN SEPARATE PARAS
- FINAL CLOSING OF FILES IN SEPARATE PARAS

Example:-

```
PROCEDURE DIVISION.
0000-MAINLINE.
      DISPLAY 'PROGRAM STARTED SUCCESSFULLY '.

      PERFORM 0100-INIT-PARA THRU 0100-EXIT.
      PERFORM 0500-PROCESS-PARA THRU 0500-EXIT.
      PERFORM 9999-CLOSE-PARA THRU 9999-EXIT.

      DISPLAY 'PROGRAM ENDED SUCCESSFULLY '.
      GOBACK.

0100-INIT-PARA.
     DISPLAY 'INSIDE INIT PARA'
     ACCEPT WS-DATE FROM DATE.
0100-EXIT.
      EXIT.

0500-PROCESS-PARA.
     DISPLAY 'INSIDE PROCESS PARA'
     PERFORM 0300-READ-PARA THRU 0300-EXIT.
     PERFORM 0400-MAIN-LOGIC THRU 0400-EXIT.
     PERFORM 0450-WRITE-PARA THRU 0450-EXIT.

0500-EXIT.
```

```
      EXIT.

0300-READ-PARA.
     READ FILES ….
0300-EXIT.
      EXIT.

0400-MAIN-LOGIC.
      ALL LOGICAL PROCESSING (FOCUS ON THIS PARA)
      FOR COMPLEX PROGRAMS, TRY TO BREAK DOWN MAJOR LOGIC INTO
      SUBLOGICS AND WORK ONE BY ONE.

0400-EXIT.
      EXIT.

0450-WRITE-PARA.
     WRITE FILES ….
0450-EXIT.
      EXIT.

9999-CLOSE-PARA.
     DISPLAY 'INSIDE CLOSE PARA'.
9999-EXIT.
      EXIT.
```

SIGNED AND DECIMAL VARIABLES

Representation of a Signed Variable

```
01 WS-AMOUNT              PIC    9(6)V99   VALUE 1999.99.
01 WS-AMOUNT-A            PIC    9(6)V9    VALUE 1999.0.
01 WS-AMOUNT-B            PIC   S9(6)V9    VALUE -1999.0.
01 WS-AMOUNT-C            PIC   9(6)V9S    VALUE 1999.0-.
01 WS-AMOUNT-D            PIC   S9(9)V9(9) VALUE 999999.90.
01 WS-AMOUNT-E            PIC   S9(9)V9(9) VALUE -999999.90.
```

EDITABLE CHARACTERS/DISPLAY CHARACTERS

```
WORKING-STORAGE  SECTION.
      01 WS-AMOUNT              PIC  9(6)V99.
      01 WS-PRINT-AMOUNT        PIC ZZZ,ZZ9.99.
```

```
PROCEDURE DIVISION.
      MOVE WS-AMOUNT          TO WS-PRINT-AMOUNT.
      DISPLAY WS-PRINT-AMOUNT.
```

	WS-AMOUNT	**WS-PRINT-AMOUNT**
$ZZZ,ZZZ.ZZ	009999.90	$9,999.9
ZZZ,ZZZ.ZZ$	000123.00	123$
******9.99	000156.90	***156.90
ZZZZZ9.99	001467.80	1467.80

-ZZZZZ9.99	-000090.00	-90.00
ZZZZZ9.99-	-001000.00	1000.00-

SEQUENCE CONTROL

Program sequence can be altered with the help of a sequence control verb

Sequence Control Verbs:

EXIT - Provides a common end point to a set of statements. It is usually used to exit PARAs or Sections

```
Ex: - 0100-MAIN-LINE-PARA.
           PERFORM  0200-INITIALIZE-PARA THRU 0200-EXIT.
           PERFORM  0300-CALCULATE-PARA THRU 0300-EXIT.

           GOBACK.

      0200-INTIALIZE-PARA.

      0200-EXIT.
            EXIT.

      0300-CALCULATE-PARA.

      0300-EXIT.
            EXIT.
```

GO TO -

Transfers control abruptly to the specified PARA or Section. This is not usually allowed in a program and allowed only if the program execution is transferred to the EXIT para of the same PARA.

```
Ex:-     0050-COMPUTE-SALARY.

                GO TO 0050-EXIT.
         0050-EXIT.
               EXIT.
```

GO BACK - Go back marks the end of the program and release any VSAM/Database resources allocated to the program. It is usually used in COBOL programs to stop the execution of the program.

STOP RUN - Stops the execution of the program

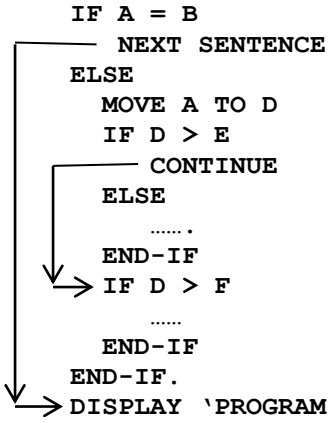
CONTINUE – It's a dummy or a no operation statement. It indicates that no executable instruction is present.

```
DISPLAY IF A > B BUT LESS THAN C
    IF A > B
      IF A > C
```

```
      CONTINUE
    ELSE
      DISPLAY 'VALUE IN A IS ' A
      DISPLAY 'A IS GREATER THAN B BUT LESS THAN C'
    END-IF
  END-IF.
```

NEXT SENTENCE – Transfers control to the next logical COBOL statement i.e., statement following next period.

IF A IS NOT EQUAL TO B process the logic ELSE come out of the loop.



REDEFINES

- Reformats a Group or subgroup variable (01 …. 49 Level numbers)
- REFINES level number should be same as the Group variable it is redefining
- Refers to the same storage area by different names with different formats and sizes

```
01   WS-TELEPHONE-NUMBER            PIC X(10).
01   WS-R-TEL-NUMBER       REDEFINES      WS-TELEPHONE-NUMBER.
      05   WS-AREA-CODE              PIC 9(3).
      05   WS-CODE1                  PIC 9(3).
      05   WS-CODE2                  PIC 9(4).


01   WS-FULL-NAME       PIC X(20).
01   WS-NAME-REDEF          REDEFINES      WS-FULL-NAME.
      05   WS-FIRST-NAME             PIC X(10).
      05   WS-SECOND-NAME            PIC X(10).


01   WS-STORED-AREAS.
      05  WS-DATE       PIC X(8).
      05  WS-DATE-X   REDEFINES   WS-DATE.
            10  WS-YEAR PIC 99.
```

```
10  FILLER  PIC X  VALUE "/".
10  WS-MONTH      PIC 99.
10  FILLER  PIC X  VALUE "/".
10  WS-DATE PIC 99.
```

OPERATORS

As usual, operators to compare two variable values or against a constant.

RELATIONAL OPERATORS:-

Operator	Operator Equivalent in words	Examples
>	IS GREATER THAN	IF A > B, IF A > 0, IF A > ZEROES, IF A > 1000, IF A > SPACES
>=	IS GREATER THAN OR EQUAL TO	IF A >= B, IF A >= 999
<	IS LESS THAN	IF A < 0, IF A < B
<=	IS LESS THAN OR EQUAL TO	IF A <= 0, IF A <= B
=	IS EQUAL TO	IF A = 0, IF A = B
NOT =	IS NOT EQUAL TO	IF A NOT = ZEROES, IF A NOT = B

LOGICAL OPERATORS:-

Operator	Description	Examples

AND	Operator is used to check whether all of the given conditions are satisfied	IF (A > B) AND (A > C)
		DISPLAY 'A IS GREATER THAN B AND C'.
		IF (A>B) AND (A>C) AND (A>D)
		DISPLAY 'A IS GREATER'.

OR	Operator is used to check whether any of the condition is satisfied	IF GENDER = 'M' OR GENDER = 'F' DISPLAY 'VALID GENDER'. IF (A = B) OR (A = C) OR (A = E) DISPLAY 'A IS EQUAL TO EITHER B,C,E'
NOT	Negate Condition	IF A IS NOT NUMERIC DISPLAY 'INPUT FIELD IS NOT NUMERIC' IF WS-COUNT IS NOT ZEROES MOVE ZEROES TO WS-COUNT.

ARITHMETIC OPERATIONS

We can do all arithmetic expressions in COBOL. We have ADD, SUBTRACT, MULTIPLY, DIVIDE and COMPUTE Command verbs to do this. There are many ways and the common ways are defined here. Compiler accepts ' ' or a ',' between variables.

Here, LEFTHAND side variables doesn't undergo change and the variables on RIGHTHAND side undergoes changes.

```
ADD 1 TO A                  -> A = A + 1
ADD A TO B                  -> B = B + A
ADD A TO B GIVING C         -> C = A + B
ADD A B  C GIVING D         -> D = A + B + C
ADD A B  C D E GIVING F     -> F = A + B + C + D + E

SUBTRACT 1 FROM A           -> A = A - 1
SUBTRACT A FROM B           -> B = B - A
SUBTRACT A FROM B GIVING C  -> C = B - A

MULTIPLY A BY B             -> B = B * A
MULTIPLY A BY B GIVING C    -> C = A * B

DIVIDE   A BY B GIVING C REMAINDER D

                            -> C = QUOTIENT OF A/B
                            -> D = REMAINDER OF A/B
```

We can give any Arithmetic calculation involving + - * or / with proper brackets as given below

```
COMPUTE WS-TOTAL = ((A + B)/ (C – D)) * (E+ (1 /F))
```

If there is Truncation in Arithmetic expression happens, there is a way to route the logic to a PARA or a simple DISPLAY statement as given above. Please note the placement of Arithmetic verbs ADD, MULTIPLY scope terminator at the end of error message or routine.

```
ADD A TO B GIVING C
    ON SIZE ERROR
       DISPLAY 'ADD ERROR – TAKE APPROPRIATE ACTION'.
```

('ADD ERROR – TAKE APPROPRIATE ACTION' will be displayed in Output spool if PIC clause of C variable is not sufficient to hold the output of A and B)

```
MULTIPLY A BY B GIVING C
    ON SIZE ERROR
       DISPLAY 'MULTIPLY ERROR – TAKE APPROPRIATE ACTION'.
```

('MULTIPLY ERROR – TAKE APPROPRIATE ACTION' will be displayed in Output spool if PIC clause of C variable is not sufficient to hold the output of A and B)

```
COMPUTE  AMOUNT-B =  AMOUNT-A * AMOUNT-C
                ON SIZE ERROR
     PERFORM ERR-PARA THRU ERR-EXIT.
```

For Rounding of Decimal variable, we use rounded keyword in Arithmetic expressions.

```
COMPUTE X ROUNDED = A * B
```

ADD A TO B ROUNDED C

CONDITIONAL EXPRESSIONS

Major Conditional Expressions are
1) IF
2) EVALUATE
3) PERFORM

IF STATEMENT

- We can check more than one condition connecting using a logical operator such as AND, OR, NOT
- We can have any number IF ELSE END-IF within a IF statement
- ELSE is Optional
- A IF loop ends at the Scope terminator say '.'

IF condition format	**Example 1**	**Example 2**

IF THEN
END-IF
 	
IF
END-IF
(THEN is Optional)
	IF A = B
	COBOL STATEMENTS
	END-IF
	IF A > 0 THEN
	COBOL STATEMENTS
	END-IF
		IF (A = B) AND (A > C)
		COBOL STATEMENTS
		END-IF
		IF (A > 0) AND (A > CNT)
		COBOL STATEMENTS
		END-IF

IF
ELSE
END-IF.
	IF A > 0
	DISPLAY 'A IS POSITIVE'
	ELSE
	DISPLAY 'A IS NEGATIVE'
	END-IF.
		IF (A > B)
		DISPLAY 'A IS GREATER'
		ELSE
		DISPLAY 'B IS GREATER'
		END-IF.

NESTED IFs IF ELSE IF ELSE IF ELSE END-IF.	IF TOTAL > 90 DISLAY 'A+ GRADE' ELSE IF TOTAL > 70 DISPLAY 'B GRADE' ELSE IF TOTAL > 40 DISPLAY 'PASS' ELSE DISPLAY 'FAIL'	IF A = B NEXT SENTENCE ELSE MOVE A TO D IF D > E CONTINUE ELSE ……. END-IF IF D > F …… END-IF END-IF.

FIELD CHAR TYPE CHECK USING COBOL VERBS

OPERATION	Description	Examples
NUMERIC	Used to check whether a variable is NUMERIC. Usually checked before an Arithmetic operation to make sure the field does not	IF WS-QTY IS NUMERIC MULTIPLY WS-QTY BY WS-PRICE GIVING WS-TOTAL.

	contain any junk characters and contains Numeric only	IF WS-QTY IS NOT NUMERIC MOVE ZEROES TO WS-QTY.
POSITIVE	To check whether a field value is Positive or not	IF WS-SALARY IS POSITIVE IF WS-QTY IS NOT POSITIVE
NEGATIVE	To check whether a field value is Negative or not	IF WS-DEBITS IS NEGATIVE
ALPHABETIC	To check whether a field value is Alphabetic or not	IF NAME IS ALPHABETIC

PERFORM

PERFORM does the following

a) OUTLINE PERFORM: Used to call PARAs or Sections with or without a condition
b) INLINE PERFORM: Used to execute a set of statements in a loop a set number of times or until a condition gets satisfied. For any looping

OUTLINE PERFORM

```
PERFORM 0100-PROCESS-PARA THRU 0100-EXIT.

PERFORM 0500-PARA THRU 0500-EXIT UNTIL WS-SWITCH = 'Y'.
```

INLINE PERFORM

PERFORM Loop - FIXED Number of Times

Syntax	Example	Example
PERFORM N TIMES Cobol Statements END-PERFORM. (LOOP starts from PERFORM and ends with END-PERFORM)	PERFORM 5 TIMES Cobol Statements END-PERFORM. (Performs this Loop 5 times, you can give any valid number)	MOVE 100 TO COUNTER. PERFORM COUNTER TIMES Cobol Statements END-PERFORM. (Performs 100 times as Counter is 100)

PERFORM LOOP - Unlimited times until a Condition is satisfied

Syntax/Example	Explanation
MOVE 1 TO I. PERFORM UNTIL I > MAX-COUNTER Cobol statements ADD 1 TO I END-PERFORM.	Assign a value of 1 for I variable. (Note: Variable I should have been already defined such that it should atleast match WS-MAX-COUNTER PIC clause) Performs this Loop until I is greater than MAX-COUNTER. If WS-MAX-COUNTER is 100 (say). Then this loop will execute for 100 times. Make sure you

	increment I by 1 inside the loop otherwise the loop will be infinite as I will never by greater than 100.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > MAX-COUNTER Cobol statements END-PERFORM.	Assigning I to 1 and Incrementing (Adding) I during each iteration is taken care here automatically. So MOVE 1 TO I and ADD 1 TO I is not required for this type of loop. Performs this loop until I reaches atleast 1 greater than MAX-COUNTER

EVALUATE

Evaluate is usually used
- for better understanding.
- if many conditional checks are present and if NESTED-IFs are required.
- WHEN OTHER is Mandatory.

It is usually used with Conditional Names.

Evaluate with One & two conditions	Examples	Examples

EVALUATE TRUE
  WHEN COND-1
  STATEMENT1
  WHEN COND-2
  STATEMENT2
  WHEN OTHER
  STATEMENT3
END-EVALUATE.
	EVALUATE TRUE
	 WHEN (A > 100) AND
	  (A <= 500)
	  COBOL STATEMENTS
	 WHEN (A > 100) AND
	  (A <= 500)
	  COBOL STATEMENTS
	END-EVLAUTE.
		01 GENDER PIC X.
		88 MALE VALUE 'M'.
		88 FEMALE VALUE 'F'.
		
		MOVE 'M' TO GENDER.
		EVALUATE TRUE
		 WHEN MALE
		  COBOL STATEMENTS
		 WHEN FEMALE
		  COBOL STATEMENTS
		END-EVLAUTE.

EVALUATE TRUE ALSO TRUE
  WHEN COND-1 ALSO
  COND-2
  STATEMENT1
  WHEN COND-3 ALSO
  COND-4
  STATEMENT2
  WHEN OTHER
  STATEMENT3
END-EVALUATE.
	EVALUATE TRUE
	  ALSO TRUE
	 WHEN (A > 100)
	  ALSO A <= 500)
	  COBOL STATEMENTS
	 WHEN (A > 100)
	  ALSO A <= 500)
	  COBOL STATEMENTS
	END-EVLAUTE.	

TABLES

- Set of values stored in consecutive storage locations

- We will have a Table name and a Subscript. Table name under 01 Level number
- A Table name will be a group variable. A sub group variable will just have the OCCURS clause and the final subgroup variables will have the actual fields with the PIC clause
- These are also called as ARRAYS. Multi dimensional arrays are possible. Usually we don't specify beyond a second subscript as the program will tend to become complex.

The Use of Table successfully eliminates the need for declaring new variables for repeated items.

Steps in a Table:-

If a Table is required in your program, do the following steps. We could create as many tables as you require.

1) Define the Table in Working Storage with fixed number of Occurrences (with either Subscript or Index).
2) Load the complete Table with the use of Subscript or Index with data from a SYSIN or an Input PS/VSAM file or hardcode in Working storage.
3) Refer to the Table values in your program logic wherever required with the use of Subscript.

SUBSCRIPTING

Declaration

```
01  SAMPLE-TABLE-ONE.
     05  EMPLOYEE-DETAILS OCCURS 5 TIMES.
          10 EMPID         PIC  X(6).
          10 EMPNAME       PIC  X(20).
```

OCCURS clause specifies number of table occurrences (we need to decide on the number based on Input data that we are going to load) called as a SUBSCRIPT. SUBSCRIPT should be defined in Working storage and PIC clause with sufficient storage.

```
Say for a table upto 99 entries,
   01 WS-SUB                  PIC 9(02).
Say for a table upto 9999 entries,
   01 WS-SUB                  PIC 9(04).
```

Subscript is used to store and refer data in the program.

OCCURS clause in the above examples starts from 1 and goes upto 5.

EMPID (1) – will have the emp no of first record
EMPID (2) – will have the emp no of Second record…and so on

EMPNAME (1) – will have the emp name of first record
EMPNAME (2) – will have the emp name of Second record…and so on

EMPNAME (5) – will have the emp name of fifth record…and so on

OCCURS clause may go upto any valid number that system supports. Even it could go upto 999999999.

LOAD DATA INTO A TABLE (For SUBSCRIPT)

In Procedure Division, we need to load this Table with table from a file (preferably) or with records from a SYSIN however we couldn't manually type data into JCL SYSIN every time so it is not used.

1) Data is accepted via a SYSIN or READ from a PS file and loaded into a TABLE.

Working storage definition

```
01  SAMPLE-TABLE-ONE.
   05  EMPLOYEE-DETAILS OCCURS 5 TIMES.
         10 EMPID          PIC  X(6).
         10 EMPNAME        PIC  X(20).
```

01 ws-sub 9(3). --→ Declaration of subscript is reqd

Procedure Division code to Load table

```
LoadCountryTable.
    OPEN INPUT EMPLOYEEFILE.

    READ EMPLOYEEFILE
         AT END SET EndOfEMPLOYEE TO TRUE
    END-READ.

    PERFORM VARYING WS-SUB FROM 1 BY 1 UNTIL EndOfEMPLOYEE
        MOVE EMP-ID TO EMPID(WS-SUB)
        MOVE EMP-NM TO EMPNAME(WS-SUB)
        READ EMPLOYEEFILE
           AT END SET EndOfEMPLOYEE TO TRUE
        END-READ
    END-PERFORM.
```

INDEXING

Tables can also be defined with an Index instead of a subscript incase if you want to use the table for any SEARCH operation. We need not declare the Index variable as the program will take care of the definition automatically based on the number of occurrences we give.

Indexed table had to be loaded either in ascending or descending order of the specified key in declaration otherwise we will get an error while loading.

Indexed table records normally comes from an Input file in JCL. We never hardcode in program in working storage or get from sysin.

Also, as a programmer we need to make sure that the OCCURS clause given here is sufficient for today and for future use.

Table Declaration with Index

```
01 WS-ITEM.
   05 WS-ITEM-TABLE   OCCURS 1000 TIMES INDEXED BY ITEM-INDX.
      10  WS-ITEM-NO            PIC X(06).
      10  WS-ITEM-DESC          PIC X(20).
      10  WS-ITEM-UNIT-PRICE    PIC 9(06).

01 WS-ITEM.
   05 WS-ITEM-TABLE   OCCURS 1000 TIMES INDEXED BY ITEM-INDX
                      ASCENDING KEY IS WS-ITEM-NO.
      10  WS-ITEM-NO            PIC X(06).
      10  WS-ITEM-DESC          PIC X(20).
      10  WS-ITEM-UNIT-PRICE    PIC 9(06).
```

(WITH KEY AS ASCENDING OR DESCENDING. IF KEY IS USED, REMEMBER TO SORT RECORDS FIRST AND THEN LOAD INTO THE TABLE)

LOAD DATA INTO A TABLE (For Index)

In Procedure Division, we need to load this Table with table from a file (preferably). There are other ways to declare data in Working Storage or to get data thru ACCEPT statement via sysin, but these are not normally followed as hard coding is not allowed and also we couldn’t manually type data into JCL SYSIN every time.

Table definition starts from a 01 Level number. The next lower subgroup will have the OCCUR clause and an optional PIC clause (if only one value is present) else it still breaks down into subgroups with actual fields with PIC clause.

1) Hardcoding in Working Storage

```
01 WEEKDAY-Table.
   02 TableValues.
      03 FILLER       PIC X(11) VALUE "01MONDAY   ".
      03 FILLER       PIC X(11) VALUE "02TUESDAY  ".
      03 FILLER       PIC X(11) VALUE "03WEDNESDAY".
      03 FILLER       PIC X(11) VALUE "04THURSDAY ".
      03 FILLER       PIC X(11) VALUE "05FRIDAY   ".
      03 FILLER       PIC X(11) VALUE "06SATURDAY ".
      03 FILLER       PIC X(11) VALUE "07SUNDAY   ".
   02  WEEK-TABLE REDEFINES TableValues.
      03 WEEK-NM OCCURS 07 TIMES.
        05 WEEK-NUMBER   PIC  X(02).
        05 WEEK-NAME     PIC  X(09).
```

(HERE WEEK-NM is the Table Name and WEEK-NAME (1) is MONDAY, WEEK-NAME (2) is TUESDAY and so on)

2) Data is accepted via a SYSIN or READ from a PS file and loaded into a TABLE.

Working storage definition

```
01 CountryTable.
   02 Country OCCURS 100 TIMES INDEXED BY Cidx.
      03 CountryCode    PIC XX.
      03 CountryName    PIC X(26).
```

Procedure Division code to Load table

```
LoadCountryTable.
    OPEN INPUT CountryFile.
    READ CountryFile
         AT END SET EndOfCountryFile TO TRUE
    END-READ.
    PERFORM VARYING Cidx FROM 1 BY 1 UNTIL EndOfCountryFile
        MOVE CountryRec TO Country(Cidx)
        READ CountryFile
           AT END SET EndOfCountryFile TO TRUE
        END-READ
    END-PERFORM.
```

3) REFER TABLE VALUES INSIDE THE PROGRAM OR USING SEARCH TABLE VALUES WHEN THE TABLE IS INDEXED

SEARCHES a Table by supplying value for the Indexed variable and getting results.

We have SERIAL and BINARY searches. Both does the same processing while the way it process is different.

SEARCH COMMAND – SERIAL SEARCH:

```
SET Cidx TO 1
SEARCH Country
   AT END MOVE "Code not found" TO CountryNameSF
   WHEN CountryCode(Cidx) = CountryCodeWF
        MOVE CountryName(Cidx) TO CountryNameSF
END-SEARCH
```

For Table occurrences less than or equal to 100 use SERIAL search else BINARY search.

For SERIAL search, set Index value to 1 and issue the SEARCH statement.

SEARCH COMMAND – BINARY SEARCH:

```
SEARCH ALL Country
   AT END MOVE "Code not found" TO CountryNameSF
   WHEN CountryCode(Cidx) = CountryCodeWF
        MOVE CountryName(Cidx) TO CountryNameSF
END-SEARCH
```

For Table occurrences > 100 use BINARY search. We need not set the Index for Binary search.

MULTI-DIMENSIONAL TABLE

Below Table denotes how a 2 dimensional table is used. Here, the table can store upto 2000 Employee records, while each employee record can internally store upto 25 records.

```
01 EMPLOYEE-TABLE.
      05 EMPLOYEE-RECORD  OCCURS  2000 TIMES.
         10 EMP-ID            PIC 9(03).
         10 EMP-NAME          PIC X(20).
         10 EMP-ADDRESS PIC X(30).
         10 EMP-PHONE-NO      PIC 9(10).
         10 EMP-DOJ           PIC 9(06).
         10 EMP-SALARY    OCCURS  25 TIMES.
            15 EMP-CC-YY      PIC 9(02).
            15 EMP-CC-MM      PIC 9(02).
            15 EMP-SALARY     PIC 9(07).
```

FILE HANDLING

PS and VSAM files can be used as an Input or Output to a COBOL program. We can use any number of files in a Program. Files are given as DDnames in JCL.

```
//STEP01  EXEC    PGM=REPORT
//STEPLIB    DD   DSN=WPR085.COBOL.LOADLIB,DISP=SHR
//EMPFILEI   DD   DSN=WPR085.EMPLOYEE.FILE,DISP=SHR
//*
//REPORTO    DD   DSN=WPR085.EMLOYEE.REPORT,
//                DISP=(NEW,CATLG,DELETE),
//                DCB=(LRECL=132,RECFM=FB,BLKSIZE=13200)
//SYSOUT     DD   SYSOUT=*
```

Here EMPFILEI is the Input file and REPORTO is Output file of the COBOL program 'REPORT'.

Following are done to define the filenames in Program:-

- In ENVIRONMENT DIVISION, under INPUT-OUTPUT Section / FILE-CONTROL, SELECT clauses are used to connect DDNAMEs in JCL
- In DATA DIVISION, under FILE-SECTION FD Entries are given
- FILES are OPENed, READ (or WRITTEN) Record by Record until end of File reached for processing of all INPUT records and CLOSED properly.

FILE Declaration in Environment Division:

ESDS/PHYSICAL SEQUENTIAL FILE

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
      SELECT   file-name       ASSIGN  [TO]    {DDNAME IN JCL}
            [ ORGANIZATION  IS  SEQUENTIAL]
            [ ACCESS  MODE  [IS]  SEQUENTIAL]
            [ FILE  STATUS  IS  data-name-1].
```

KSDS/RRDS

```
ENVIRONMENT   DIVISION.
INPUT-OUTPUT   SECTION.
FILE-CONTROL.
      SELECT   file-name ASSIGN  [TO]  DISK
            ORGANIZATION  IS  INDEXED
            ACCESS  MODE  IS  [ Dynamic / Random /  Sequential ]
            RECORD  KEY  IS  data-name-1
            [ ALTERNATE RECORD KEY IS data-name-2]
            [ FILE  STATUS  IS data-name-3]
```

Type of File	ORGANIZATION	ACCESS MODE	FILE STATUS
Physical Sequential	SEQUENTIAL	SEQUENTIAL	FILE-STAT
ESDS	SEQUENTIAL	SEQUENTIAL	FILE-STAT
KSDS	INDEXED	SEQUENTIAL, RANDOM, DYNAMIC	FILE-STAT
RRDS	RELATIVE	SEQUENTIAL, RANDOM, DYNAMIC	FILE-STAT

FILE-STAT is a Two byte field Alphanumeric field to be defined in Working Storage. This contains the status of any operation of a file.

Example PS/ESDS :-

```
SELECT EMPL-FILE ASSIGN TO EMPFILEI.
SELECT REPORT-FILE ASSIGN TO REPORTO.
```

FILE Declaration in Data Division:

```
DATA   DIVISION .
FILE   SECTION .
FD file-name.
      [LABEL  { RECORDS ARE/ RECORD IS} {STANDARD /   OMITTED / data-name}]
      [RECORD CONTAINS  integer-1 CHARACTERS]
      [BLOCK CONTAINS  integer-2 RECORDS]
      [RECORDING MODE IS  {F/V}]
```

Example PS/ESDS:-

```
DATA   DIVISION .
FILE   SECTION .
FD EMPL-FILE.
       LABEL RECORDS ARE STANDARD.
       RECORDING MODE IS F.
       RECORD CONTAINS 80 CHARACTERS.
01 EMPL-RECORD PIC X(80).

FD REPORT-FILE.
       LABEL RECORDS ARE STANDARD.
       RECORDING MODE IS F.
       RECORD CONTAINS 80 CHARACTERS.
01 REPORT-RECORD PIC X(132).
```

Now, the declaration are over, we need to just have working storage record to receive input file, process and move to O/p record.

FILE OPEN/CLOSE

```
OPEN INPUT FILENAME.
OPEN OUTPUT FILENAM1 FILENAM2.
OPEN I-O FILENAM3.
OPEN EXTEND FILENAM4.
```

```
CLOSE FILNAME FILENAM1.
```

Type of File	INPUT	OUTPUT	I-O	EXTEND
Physical Sequential	READ Records Sequentially	WRITE a Single Record	READ/ REWRITE a Record (Read should be done before doing REWRITE)	APPEND Records at the End
ESDS	READ Records Sequentially	WRITE a Single Record	-	-
KSDS/RRDS	READ Records Sequentially/ Random/Dynamic	WRITE a Single Record	READ/ WRITE/ REWRITE/DELETE a Record	-

For KSDS/RRDS, To Load records for the first time, we need to open in OUTPUT mode.

FILE OPERATION COMMANDS

READ:- Reads a single Record from an Input file for Sequential Processing.

```
READ file-name-1 [INTO-identifier-1]
   AT END
      --imperative-statement-1
   NOT AT END
      --imperative-statement-2
END-READ.
```

```
Examples:-
READ EMPL-FILE INTO WS-EMP-REC
 AT END
    MOVE 'Y' TO EOF-SW
 NOT AT END
    ADD 1    TO WS-RECS-READ
END-READ.
```

WRITE:-Writes a Single Record to Output file.

```
WRITE  record-name      [ FROM   rec  ]
```

```
Examples:-
WRITE REPORT-RECORD FROM WS-REP-REC.
```

REWRITE:- Rewrites a Record. Remember the Key cannot be replaced. We need to issue READ first, make sure the current record to be rewritten and then issue a REWRITE command to rewrite the record.

```
REWRITE  record-name          [ FROM   rec  ] .
```

```
Example:-
```

```
    MOVE SP-Student-Number TO SM-Student-Number.
    READ Student-Master-File
        KEY IS SM-Student-Number
        INVALID KEY DISPLAY "invalid read FS = " SM-File-Status
    END-READ.


    ADD SP-Payment TO SM-Amount-Paid.
    REWRITE SM-Rec
        INVALID KEY DISPLAY "Rewrite error fs = " SM-File-Status
    END-REWRITE.
```

DELETE:- Deletes a Record. We need to issue READ first, make sure the current record to be deleted and then issue a DELETE command to delete the record.

```
DELETE  filename RECORD.
```

Example:-

```
    MOVE SP-Student-Number TO SM-Student-Number.
    READ Student-Master-File
        KEY IS SM-Student-Number
        INVALID KEY DISPLAY "invalid read FS = " SM-File-Status
    END-READ.
    DELETE Student-Master-File RECORD
        INVALID KEY DISPLAY "Delete error fs = " SM-File-Status
    END-REWRITE.
```

66 – RENAMES

66 Level numbers are used for Renames. Here shows an example of how it can be used. RENAMES may not be used at all as we may not find it's usage necessary.

```
01 WS-PAYROLL.
      05 WS-BASIC-PAY   PIC 9(07).
      05 WS-HRA         PIC 9(07).
      05 WS-DA          PIC 9(07)
      05 WS-PF          PIC 9(07).
      05 WS-DEDUCTIONS  PIC 9(07).
         10 WS-INCOME-TAX PIC 9(07)
         10 WS-ESI        PIC 9(07).
66 WS-PF-INCOME-ESI RENAMES WS-PF THRU WS-ESI.

DISPLAY WS-PF-INCOME-ESI.
```

STRING HANDLING FUNCTIONS

STRING, UNSTRING, INSPECT and REFERENCE MODIFICATION

STRING:

```
01 WS-FIRST-NAME PIC X(20)VALUE 'SURESH'.
01 WS-LAST-NAME  PIC X(20) VALUE 'KUMAR'.

STRING WS-FIRST-NAME DELIMITED BY SPACES
       ' '           DELIMITED BY SIZE
       WS-LAST-NAME  DELIMITED BY SPACES
               INTO  WS-FULL-NAME.
```

UNSTRING:

```
01 WS-STRING-1 PIC X(40) VALUE 'WELCOME TO UST GLOBAL'.

UNSTRING WS-STRING-1 DELIMITED BY SPACE
    INTO WS-STR1, WS-STR2, WS-STR3, WS-STR4.

UNSTRING WS-STRING-1 DELIMITED BY 'UST'
    INTO WS-STR1.
```

INSPECT:

```
01 WS-STRING PIC X (20) VALUE 'USTGLOBAL'.
01 WS-COUNT  PIC 9 (03).
```

```
INSPECT <WS-STRING> TALLYING <WS-COUNT> FOR ALL CHARACTERS.
INSPECT <WS-STRING> TALLYING <WS-COUNT> FOR ALL 'L'.


01 WS-STRING-VAR PIC X (40) VALUE 'MARY.HAD.A.LITTLE.LAMB'.
INSPECT <WS-STRING-VAR> REPLACING ALL "." BY " ".

INSPECT <WS-STRING-VAR> REPLACING ALL "a" BY "A"
                                      "b" BY "B"
                                        …
                                      "Z" BY "Z".
```

REFERENCE MODIFICATION

```
01 WS-STRING PIC X (40) VALUE 'ENTER THE DRAGON'.

MOVE WS-STRING (1:10)  TO WS-STRING-1.
MOVE WS-STRING (11:20) TO WS-STRING-2.
DISPLAY WS-STRING-1.
DISPLAY WS-STRING-2.
```

EXERCISE:

1. Write a COBOL program to accept the string values for First-name, Middle-name and Last-name concatenate them and store in a data item Name. Use STRING verb.
2. Write a COBOL program to separate each word from a string say 'ALL IS WELL' and store in separate fields using UNSTRING verb.
3. Write a program to count the no. occurrence of a 'O' in the given input string.

   01 WS-STRING-1 PIC X(40) VALUE 'GODS OWN COUNTRY'
4. Write a program to replace '.' with a space in the given input string

   01 WS-STRING-2 PIC X(40) VALUE 'GODS.OWN.COUNTRY'

Remove leading blanks from a string

```
*   Solution - Use the inspect to count the leading blanks and reference
*   modification to get the substring from the point indicated by CharCount
*   and for FullStrLength - CharCount characters.
    MOVE 1 TO CharCount.
    INSPECT xStr TALLYING CharCount FOR LEADING SPACES
    DISPLAY "Task5 =" xStr(CharCount: 50 - CharCount)
```

REMOVE TRAILING BLANKS FROM A STRING

```
    MOVE 0 TO CharCount
    INSPECT FUNCTION REVERSE(xStr) TALLYING CharCount
        FOR LEADING SPACES
```

```
    DISPLAY "Task4 After = "xStr(1:50 - CharCount) "<<<<<<<".


    PERFORM VARYING CharCount FROM 50 BY -1
           UNTIL xStr(CharCount:1) NOT = SPACE
    END-PERFORM
    DISPLAY "Task4 After = "xStr(1:CharCount) "<<<<<<<".


*   Find the location of the first occurrence of substring yStr in xStr.
*   Solution - Use the INSPECT..TALLYING to count the characters before
*   the first occurrence of the substring. CharCount has the location.
*   In this example we get the position of the substring "source".
    MOVE 1 TO CharCount
    INSPECT xStr TALLYING CharCount for CHARACTERS
        BEFORE INITIAL "source".
    DISPLAY "Task6 First occurrence is in char position " CharCount
```

INTRINSIC FUNCTIONS

RELATIONAL - MIN, MAX, SUM

Ex:- a) Compute M = Function Max(X Y Z),

b) Compute M = Function SUM(A B C D)

SCEINTIFIC – SIN, COS, TAN, LOG, SUM, FACTORIAL

Ex:- COMPUTE A = Function Log10(x)

STRING MANIPULATION – LOWER-CASE, UPPER-CASE, REVERSE

Ex:- a) Move function Upper-case (field-variable) to <Field-Variable>.

b) Comparing two fields, if we are unsure of the case,

IF FUNCTION UPPER-CASE (FIELD1) = FUNCTION UPPER-CASE (FIELD2)

c) MOVE FUNCTION REVERSE (FLDA) TO FLDB.

DATE MANIPULATION –

CURRENT-DATE (YYYYMMDDHHMMSSTT+HHMM),

DATE-OF-INTEGER (YYYYMMDD), DAY-OF-INTEGER (YYYYDDD)

EX:- MOVE FUNCTION (CURRENT-DATE) TO WS-CURRENT-DATE-DATA

STATISTICAL ACCOUNTING - MEAN, MEDIAN, MODE, RANGE, STANDARD-DEVIATION, VARIANCE

OTHER – INTEGER, LENGTH

Ex:- a) COMPUTE LEN1 = FUNCTION LENGTH (FLD-IN)

(Length of Non-Numeric fields)

b) COMPUTE WS-INT = FUNCTION INTEGER(222.233)

EXERCISE:-

1. Accept Maths, Physics, Chemistry and Biology marks of 2 Students and display the Student Name who got more marks between the two.
2. Reverse the following string and store in a variable in Capital letters 'abcdefghijklmnopqrstuvwxyz'
3. Accept Current Date and Display in Spool